

Programming Windows Applications for Multiple Languages

*Jean-Paul Bleau
Ordinateurs Laval Inc.*

*CA-World 2000
eBusiness Solutions in Internet Time
JV165SN*

Introduction

When developing software, the designer has to think “international” to extend his market and get a bigger customer potential. The International Customer will look for features like local language support, support for Windows International Settings and ability to support more than one language at a time.

This session will look at how to use stored text strings and messages, maintain different language data and the user interface. We will also have a look at the Windows settings and Unicode or double-byte characters (DBCS).

When designing your application, keep the application and user interface components separate, because independent components make the application easier to localize and maintain. For example, with separate components, you do not have to browse the source code to localize interface elements. You can reduce the cost of developing an international application and bring it to market more quickly by designing it as an international application initially, rather than modifying it for international use later.

Using Data Driven Labeling

Data driven labeling has been the standard method to create multi-lingual CA-Clipper[®] applications. This method can still be used for Windows applications, but the string tables are most effective and have a better integration into the Windows platform.

Why data driven over string table?

The string table is for storing string text, that’s all. If you need to store other information with your string, the data driven approach becomes more profitable. If you think of languages that use the right-to-left writing, you need to define different coordinates for the start of the label. Also, using data driven logic you can group all your strings in one record with as many fields as you have languages defined.

Store the text and the coordinate for the labels.

In CA-Clipper[®], using coordinates within the data driven definition was quite easy in the following form:

```
@ FIELD->Lin, FIELD->Col SAY FIELD->Msg
```

With CA-Visual Objects[®], it is quite different, because you have to deal with the resources inside your window. You have to retrieve your data and coordinates at instantiation of the windows. It would look like this:

```
p := Point{ _FIELD->X, _FIELD->Y }
d := Dimension{ _FIELD->SzX, _FIELD->SzY }
cMsg := _FIELD->Msg
oMyFT := FixedText{ SELF, 1, p, d, cMsg }
```

Here you can see a new parameter that can be saved in your data driven file: the size (Dimension) of the text. This can be important if you use some language that translates to very long string that cannot fit inside your defined window. You just have to draw it smaller to fit the space allocated.

Another way to store information along your text inside the string table is to use embedded data inside your message. To retrieve your message you will need to use a user defined function (UDF) to control the way data is stored into your string. You can use separators inside your string to separate the different information like position and size. Then your function could retrieve the first 3 characters and translate them to the X dimension parameter. The same can be done with all the other parameters that have been stored within the string. The most important thing is to define your own standards as to how to store the string and always keep these standards when creating your data.

Creating constants to identify your labels.

You need a way to access your strings. It could be a string table DLL or a database containing your text; you still need a key to find it. With the database approach, you need the key to search for the desired string using a seek method within an indexed file. Numeric constants could be used as in string tables, but I suggest storing your numeric keys in a character field using STR() for indexing reasons. If you want to use a numeric constant as with DLL string tables, you can create defines for your constants for readability within your code. Put all the defines in a single library; each database or DLL should have its own definitions for the constants. Then attach the definitions' library to your application.

Writing generic functions to retrieve the labels.

In order to work with your string table or your database containing your text, you have to create your own generic functions to read the information and return it to your application. If you keep a certain standard in your applications, you will be able to use the same functions to retrieve your strings in all of them. These functions can be created inside a personal library of functions you will use in all your future development. You need to create a function where you will send a key (the constant) and a language, so the function will return the desired string. It could be something like this:

```
FUNCTION StrSearch(nKey, cLanguage)
LOCAL cRetString := "" AS STRING
LOCAL nX := 0 AS INT
LOCAL nY := 0 AS INT
SY_SELECT("strings") // open the file
strings->(DBSEEK(STR(nKey)))
```

```

IF strings->(FOUND())
  nX := strings->Xvalue
  nY := strings->Yvalue
  DO CASE
    CASE cLanguage == "F"
      cRetString := strings->MsgFrench
    CASE cLanguage == "E"
      cRetString := strings->MsgEnglish
    CASE cLanguage == "S"
      cRetString := strings->MsgSpanish
    ENDCASE // any other language...
  ENDF
RETURN {nX, nY, cRetString}

```

This function can be personalized to suit your specific needs. You can add as many languages as you need. You could also use more generic fields like:

```
cRetString := strings->Msg1
```

This way language 1 can be any language. You could add X and Y language specific coordinates. When you have such functions, your biggest problem will be to build and maintain the string database.

The previous example is good for multiple languages during the same session or location. If you only need one language at all time, you only need 1 message field and create many DBF specific to the language. All depends on your needs and on the application's specifications.

Placing the label on the window.

In a generated window, the fixed texts are defined within a Windows Resource definition. In our case, we need to create the fixed text after the instantiation of a window. It could be placed inside the PostInit() method of the window. This method is called at the end of the Init() method. It is acting as a subclass of the window. It could look like this:

```

METHOD PostInit() CLASS MyWin
LOCAL p AS POINT
LOCAL d:=Dimension{10,10} AS DIMENSION
LOCAL aMsg := {} AS ARRAY
aMsg := StrSearch(I_MYWIN_FT1, cLang)
p := Point{aMsg[1],aMsg[2]}
oMyFT := FixedText{SELF,1,p,d,aMsg[3]}
RETURN self

```

In this example, the I_MYWIN_FT1 is the constant you must define as a numeric that represents the key field to search for in the strings' database. You can then add other functionality like dimensions. In this example, the variable cLang is a global representing the user language. This variable can be stored in the user's .ini file and then retrieved directly by the StrSearch() function, eliminating a global and a parameter to the function call.

Using DLL String Tables

In CA-Visual Objects the visual editors support reading the HyperLabel properties, Caption, Description, and HelpContext, from a string table. Instead of entering strings for these properties, you can enter a parameter string enclosed in angle brackets that will generate a lookup operation in a string table based on a unique ID. The string takes on the following form (note that the angle brackets are required):

```
<[Default Value], StringID, [Module]>
```

In this case, Module means the Windows EXE or DLL file within which the string resource resides. If Module is omitted, then the current EXE is used. For example, you might enter the following for the Caption of the File menu:

```
<"&File", MyFileString, MyLanguage>
```

The generated code, when you use a parameter string, includes a call to the LoadResString() function. For the example given above, the call is:

```
LoadResString("&File", MyFileString, MyLanguage)
```

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).

This example illustrates a technique for looking up an object's 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:

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

Then, in your start module, add the following:

```
GLOBAL hLibHndl AS PTR
GLOBAL MyLanguage AS STRING
METHOD Start() CLASS App
...
  IF ... // it's French
    MyLanguage := "MYFRENCH.DLL"
  ELSEIF ... // it's German
    MyLanguage := "MYGERMAN.DLL"
  ENDIF
  hLibHndl := LoadLibrary(MyLanguage)
...
RETURN NULL_STRING
```

Then, in the closing of the apps, do not forget to put a call to:

```
FreeLibrary (hLibHndl)
```

Then create a library just for the defines. This library will be attached to the apps and to each string table DLL. See the section “*Creating constants to identify your string*”.

Format of a string table DLL.

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

```
RESOURCE French STRINGTABLE
BEGIN
    MyFileString, "&Fichier"
    MyCloseString, "&Fermer"
    MyNameString, "Nom"
    ...
END
```

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

```
RESOURCE English STRINGTABLE
BEGIN
    MyFileString, "&File"
    MyCloseString, "&Close"
    MyNameString, "Name"
    ...
END
```

Note: The RC compiler does not offer a line-continuation symbol for strings in string tables. To force a carriage return into a long line of text, use one of the methods described below.

One method is to force the carriage return using \012\015 (octal representation of line feed / carriage return control characters). The following example demonstrates the use of \012\015 and should be considered to be on one continuous line:

```
RESOURCE English STRINGTABLE
BEGIN
    IDSLONGSTRING, "This is a long line of text so I would like \012\015 to force a
    carriage return."
END
```

Another method of forcing a carriage return is to press ENTER and continue the line on the next line. The following example will force a carriage return after the word "like". The string should be considered to be on one continuous line except for the return after the word “like”:

```
RESOURCE English STRINGTABLE
BEGIN
    IDSLONGSTRING, "This is a long line of text so I would like
    to force a carriage return"
END
```

Note: There is a 255-character limit (per string) in a string table. If you try to use the \n or other \ characters, the RC compiler will ignore them.

Storing your string in different DLLs.

In order to distribute your language specific application, you have to create one string table DLL for each language. The string table DLL can be created by following these steps:

1. Click on the “New Application” button and choose the “Next” button.
2. In the Application Name and Type fields, enter your DLL name (like “English”), click on the radio button for DLL, and then click on the “Next” button.
3. In the User Interface window, click “Yes” for "Do you want to provide your own user interface code?" Then answer “No” for the Terminal window. Here you do not need the GUI Classes or the Terminal Classes. Click “Next”.
4. In the Data Access window leave all buttons blank; it will not access data at runtime. Click “Next”.
5. For Language Style, I suggest choosing Strict and leaving the Debug button blank. Click “Next”.
6. In the Path for .EXE or .DLL field, enter the full path and file name you want for your DLL. Ex.:C:\DEV\English.DLL. Click “Next”.
7. Leave What Libraries Will Your Application Use? blank for now, unless you already created your Defines library, and then add it to the left in Include in my Application. Click “Next”.
8. In the User Defined Commands dialog, just click on “Finish” to continue. You now have a new DLL called English (or whatever name you gave it).
9. You will now open “Module 1,” or create a module if it is not created, and give it a significant name like “English String Table”. Then open the module by right-clicking on it and choose “Edit all source in module”.
10. Inside the Source Code Editor, enter your string table using the following example format:

```
RESOURCE English STRINGTABLE
BEGIN
    MyFileString, "&File"
    MyCloseString, "&Close"
    MyNameString, "Name"
    ...
END
```

11. When you are finished entering the strings, save your work and exit the Source Code Editor. If your Defines library is not created, wait until you have it before compiling and linking your DLL. You have to return to the DLL properties and add the Defines library. Look in the next section for information on defining constants.

Creating constants to identify your strings.

With a string table, you absolutely need a numeric constant, but you can use defined constants for readability inside your code. A string table is stored in a DLL. Each of the DLLs should have an external library for the definition of the constants. This way, you use the same defines for the DLL and the application. You just have to include it inside the library's property sheet for the application or the DLL property dialog. Only after this library is included in your application and the string table DLL will you be able to compile and link without defines errors.

The steps for creating a Defines library are as follows:

1. Click on the “New Application” button and choose the “Next” button.
2. In the Application Name and Type fields, enter your library name (like “String Table Defines”), click on the radio button for Library, and then click on the “Next” button.
3. In the User Interface window, click “Yes” for “Do you want to provide your own user interface code?” Then answer “No” for the Terminal window. Here you do not need the GUI Classes or the Terminal Classes. Click “Next”.
4. In the Data Access window, leave all buttons blank; it will not access data at runtime. Click “Next”.
5. For Language Style, I suggest choosing Strict and leaving the Debug button blank. Click “Next”.
6. What Libraries Will Your Application Use? should always be blank. Click “Next”.
7. In the User Defined Commands dialog, just click on “Finish” to continue. You now have a new library called String Table Defines (or whatever name you gave it).
8. You will now open “Module 1,” or create a module if it is not created, and give it a significant name, like “Defines”. Then open the module by right-clicking on it and choose “Edit all source in module”.
9. In the Source Code Editor, enter the defines using the following example format. Each define MUST be unique, having a different number. The number itself is not important, its uniqueness within the file is.

```
DEFINE MyFileString := 100
DEFINE MyCloseString := 101
DEFINE MyNameString := 500
```
10. When you are finished entering the defines, save your work and exit the Source Code Editor. You have to return to the application and the DLL property sheet and add the Defines library. You will then be able to compile and link your DLL and application.

Using the strings inside a window.

The CA-Visual Objects® Window Editor lets you load text from a string table. To load text from a string table rather than hard coding the text by entering a string in the Caption property, enter:

```
<[Default Value], StringID, [Module]>
```

The angle brackets are required as part of the entry. This causes the Window Editor to generate code to load the caption from a string table, using the LoadResString() function. The Default Value is a string that's displayed if the StringID cannot be found in the string table. If this parameter is omitted, it is the same as passing a NULL-STRING. The StringID is the unique identifier in a string table. The Module parameter is the DLL or EXE file that contains the string table. If this parameter is omitted, LoadResString() assumes the string table is in the current EXE or DLL file. Anytime < ... > is used for a text string, the Window Editor generates the code to read the text from a string table.

If you want to manually create the code, just assign the return value of the LoadResString() function to the caption or fixed text like this:

```
Self:Caption := LoadResString("", ,StringID, StrTableDLL)
```

Using the strings inside a menu.

The CA-Visual Objects® Menu Editor also lets you load text from a string table. To load menu captions from a string table rather than hard coding them by entering a string in the Caption property, enter:

<[Default Value], StringID, [Module]>

The angle brackets are required as part of the entry. This causes the Menu Editor to generate code to load the caption from a string table, using the LoadResString() function. The Default Value is a string that's displayed if the StringID cannot be found in the string table. If this parameter is omitted, it is the same as passing a NULL-STRING. The StringID is the unique identifier in a string table. The Module parameter is the DLL or EXE file that contains the string table. If this parameter is omitted, LoadResString() assumes the string table is in the current EXE or DLL file. Anytime < ... > is used for a text string, the Menu Editor generates the code to read the text from a string table.

String Tables to reduce your code size.

When you are storing strings in external DLL's, it provides a much greater ease of converting text into other languages. It also helps reduce the sizes of the code and data sections in the CA-Visual-Objects® application. If all your strings are stored inside a DLL, the .EXE file will be much smaller. This is true if you clear or replace all the default values in your call to LoadResString() with a null string.

Using DLL string tables slightly slows down the initial loading of forms, and the overall performance of your program might suffer if you remove all strings from it. For example, performance might suffer if the program searches for strings while inside a loop. This is the price to pay for internationalization. This is related to the speed of the CPUs, which are faster and faster these days.

Getting Help for Translations

If you live in the United States and you do not know the language or the culture of the country in which you want to sell your software, you will need help from somebody who knows the language, but also the culture, politics and religions, which are very important. You also have to be careful when developing the application and follow these rules:

- When you create messages in your application, English text strings are usually shorter than equivalent text strings in other languages. The following table shows the additional average growth for strings, based on their initial length.

English length (in characters)	Additional growth for localized strings
<i>1 to 10</i>	<i>200%</i>
<i>11 to 20</i>	<i>100%</i>
<i>21 to 30</i>	<i>80%</i>
<i>31 to 50</i>	<i>60%</i>
<i>51 to 70</i>	<i>40%</i>
<i>over 70</i>	<i>30%</i>

- Also, when designing messages, avoid constructing strings dynamically. For example, if your program displays this:

There are 10 records in the report

do not place two substrings in the table and build the message like this:

```
cmessage := cSubStr1 + Str(nRecs) + cSubStr2
```

This can cause headaches in languages where the word order is different from English. This might be a better solution:

Number of records in the report: 10

- For the same reason, do not generate plurals by adding “s” to the singular.
- Also, be careful *not* to translate strings that are used only internally, like field names, the names of files, and the keywords in INI files. Translating these items can cause all kinds of problems if files are shared between different versions of the program.

Doing all translations simultaneously.

The best way to handle the translation is to build the application using a master string table with your native language, and then copy this string table to other files to submit them for translation. If you try to do it during development, you can get in trouble synchronizing translations and coding changes. One easy way to do it is the following:

1. Select your string table module and right-click on it. Then, select Edit All Source in Module.
2. From the Source Code Editor, select File and Export.
3. In the Save As... dialog, enter “English” in the File Name SLE. Press the Save button.
4. Repeat steps 2 and 3 with each language name as file name, for as many languages as you want to translate.
5. You now have text files you can submit to a translator. He can replace the English text inside the quotation marks with the translated text. Be sure to instruct the translator not to remove the quotations marks and not to translate the constant defines. Be sure also that the translator uses the good codepage while writing.
6. Once you have your translated table, you can import it into a new DLL inside CA-Visual Objects. Refer to “Storing your string in different DLLs.” for directives, and just replace step 10 with the following: Select Edit All Source in Module, then File and Import your new file. If your codepage is different from the one used in the country, some characters may appear “bizarre”. This is normal, but the good character will be displayed when running on the good codepage.
7. You can now add the Defines library to the DLL property, compile and link this DLL. You now have a DLL to distribute with your application.

Ask the end users for the best translation.

Sometimes, it is hard to find good translations in your local premises. You could try to find beta testers in the target country to help you fix translation errors. This way, you could be sure your program fits into cultural, political and local religious issues. You have to be sure that no part of your application conflicts with the local culture or religion or politics. You can get into big trouble and lose sales if you shock your customers. Be sure to have more than one person involved in translation and that they are not related. Some big companies have had some big troubles with subversive translations in the past.

Using CompuServe or the Internet to get help.

One other great source of help is online services where so many people are ready to help you. People from around the world can give you hints on cultural or local issues and marketing ideas. Just find a good forum or newsgroup and ask.

Getting it from a translation professional.

One more expensive way to get translations done is with the help of professional translators. You could also ask your local language schools or university where to find resources. Sometimes they can suggest students that could handle the job for less money than professionals. You can also find translation professionals on CompuServe, where a lot of them have accounts.

Windows International Settings

Each version of Windows has its own configuration for international issues. In Windows 3.1x[®], the settings are stored in the file WIN.INI. In Windows 95[®], they can be accessed in the WIN.INI file or from the Windows Registry. In Windows 95[®] or NT[®], they can be accessed via the Regional Icon in the Control Panel. The icon looks like the planet Earth. When you double-click on it, you discover the following:

1. **Regional Settings.** In the list box, you can select the language and country you want to set as regional default settings for this computer. Defaults will change the way programs display and sort dates, times, currency and numbers.
2. **Number.** In this property sheet, you can change the default set by the language you selected. You can change the decimal symbol, the number of digits after decimal, the digit grouping symbol (thousands), number of digits in a group, the negative sign symbol, the negative number format, the display leading zeroes, the measurement system and the list separator.
3. **Currency.** In this property sheet, you can change the default set by the language you selected. You can change the currency symbol, the position of the currency symbol, the negative number format, the decimal symbol, the number of digits after decimal, the digit grouping symbol (thousands) and the number of digit in the group (thousands). These are mostly the same properties as in the Number property sheet.
4. **Time.** In this property sheet, you can change the default set by the language you selected. You can change the time style, the time separator character, the AM symbol and the PM symbol.
5. **Date.** In this property sheet, you can change the default set by the language you selected. You can change the short date style, the date separator character, and the long date style.

On all the property sheets, you have a sample line to show you the result of your settings. All of these settings can be found in the WIN.INI file under the [Intl] section with the appropriate Entry name for it.

WIN.INI entries in the International section.

Entries in the [International] Section in WIN.INI.

Entry	Description
icountry	Country code (usually the same as the country's telephone code). Canada is the only exception since it has the same telephone country code as the U.S. The country code for Canada is 2, U.S. is 1.
scountry	Country name as a string.
slanguage	Three-letter code indicating the National language code selected by the user (for example, FRA = French, ENU = US English).
sShortDate	Short date format (for example, mm-dd-yy). M = 1-12 MM = 01-12 D = 1-31 DD = 01-31 YY = 00-99 YYYY = 1900-2099
sLongDate	Long date format (like sShortDate, but also supports days of week, month names, and other elements). M = 1-12 MM = 01-12 MMM = Jan-Dec MMMM = January-December D = 1-31 DD = 01-31 DDD = Mon-Sun DDDD = Monday-Sunday YY = 00-99 YYYY = 1900-2099
itime	Time format (0 = 12-hour clock, 1 = 24-hour clock).
stime	Character used to separate the hours and minutes, and the minutes and seconds in the time string (:)
sll59	Trailing string used to denote times before noon (for example, "AM").
s2359	Trailing string used to denote times after noon (for example, "PM"). If the 24-hour clock is in force, the string can be used to denote the time zone (for example, "PST").
iTLZero	If 0, leading zero in hours field is suppressed.
scurrency	Currency symbol for the selected language.
Icurrency	Position of currency symbol. 0 = before the amount "\$1" 1 = after the amount "1\$" 2 = before the amount with an intervening space "\$ 1" 3 = after the amount with an intervening space "1 \$"
iCurrDigits	Number of digits after the decimal point in a currency amount.

Entry	Description
iNegCurr	Format for negative currency amounts. 0 = “{(\$)}” 1 = “-.\$1” 2 = “\$.-1”
sthousand	Thousand separator. (,)
sdecimal	Decimal point character.
idigits	Number of digits after the decimal point in a number.
iLZero	If 0, zero to the left of the decimal point is suppressed in numbers between -1.0 and +1.0.
slist	This is the character to be used to separate different elements in a list. This value MUST be different from the one for the decimal separator in the sDecimal entry.
imeasure	Measurement system. 0 = Metric 1 = English

Regarding sLongDate, CA-Visual Objects® does not support this setting, so if you want to display dates in this way, you will have to read sLongDate and do the formatting yourself.

Windows 95® is available in the following local languages. The value column represents the 3-letter codes for the sLanguage variable in WIN.INI:

Value	Description
DAN	Danish
NLD	Dutch
DEU	German
ENG	English (International)
ENU	English (U.S.)
ESN	Modern Spanish
ESP	Spanish
PTG	Portuguese
FIN	Finnish
NOR	Norwegian
SVE	Swedish
FRA	French
FRC	Canadian French
ISL	Icelandic
ITA	Italian
	Arabic

Value	Description
	Czech
	Hungarian
	Basque
	Polish
	Catalan
	Greek
	Japanese
	Turkish
	Chinese
	Hebrew
	Korean
	Russian
	Thai

These are the possible Values of sLanguage in WIN.INI. These settings indicate the driver used for collation purposes, not the language in which Windows itself is running.

Any changes in WIN.INI under the [Intl] section will be in effect for all versions of Windows 3.1x[®] and Windows 95[®] that continue to use this file for configuration purpose. You will have to restart Windows for the new settings to take effect.

Registry sections related to International Settings.

Although the Registry replaces the basic function of the initialization files used in earlier versions of Windows, the SYSTEM.INI, WIN.INI, and WINFILE.INI files still appear in the Windows directory. These files continue to be used for compatibility with earlier Windows-based applications and device drivers. For example, entries in WIN.INI and SYSTEM.INI created by Win16-based applications are not updated in the Registry, because such applications do not know how to access the Windows 95[®] Registry.

If you install Windows 95[®] as an upgrade to Windows 3.1[®], some INI file settings are copied into the Registry, including settings from CONTROL.INI, PROGMAN.INI, SYSTEM.INI, and WIN.INI.

Some INI file entries are not moved to the Registry, but remain in the INI file for compatibility with Win16-based applications. Most of these entries can be changed without editing the INI files by using the graphical tools provided with Windows 95[®]. However, some INI entries cannot be set using the Windows 95[®] user interface. These entries are required for some applications to function properly, but should not need direct modification by users.

When you use the Regional settings in the Control Panel, the registry and WIN.INI are updated automatically. The registry entry for International Settings is:

```
Hkey_Current_User\Control Panel\International
```

Under this key, you can find the default country coded under the entry "locale". If you change any of the default values in the Regional settings, they will be reflected under the same entry name as in WIN.INI. If you use the default, they do not appear and they are stored internally with the "locale" country code.

Note: Modifying the Registry is hazardous; please be sure that you know what you are doing when changing the values and be sure to do a backup of two particular registry files before experimenting. These two files are USER.DAT and SYSTEM.DAT and they have System, Hidden and Read-only attributes.

Dates and time format.

In America, we use the familiar mm-dd-yy format for dates, while dd-mm-yy is the norm in Europe and in the Province of Quebec. Some other countries prefer yy-mm-dd. The separator symbol varies, too. For example, some regions use a forward slash rather than a hyphen.

If SetInternational(#Windows) is in force, you do not need to take any special action to handle these differences. All CA-Visual Objects® date-handling functions will recognize the sShortDate entry in WIN.INI, which in turn determines the local date format.

CA-Visual Objects® time handling functions are all sensitive to the WIN.INI settings (itime, stime, sl159, and s2359). For example, Time() returns the system time in 12-hour or 24-hour format, depending on the setting of itime.

Currencies, numbers and amounts.

If English-speaking countries all use a period as a decimal point and a comma to separate thousands, many countries do the reverse. There are also countries that use a space as the thousand's separator. The easiest way of handling these variations is with picture clauses, which are used with the Transform() function and FieldSpec properties. Consider the following code:

```
nValue := 12345.6789  
cVar := Transform(nValue, "99,999.99")
```

In the U.S., this would display "12,345.68" but in Europe the result would be "12.345,68".

This behavior depends on three settings: sThousand, sDecimal, and iDigits. You can override them by calling the CA-Visual Objects® functions shown in "Overriding the Windows settings."

When dealing with amounts of money, you can determine the local currency symbol by reading `sCurrency` from `WIN.INI`. Keep in mind that it might consist of two or even three characters (for example, "Esc" is the Portuguese Escudos). Also, it might appear either before or after the amount and it might be separated from the amount by a space. To confuse matters further, some countries use parentheses to indicate negative values, others use a minus sign; some place the minus sign before the currency symbol and others insert it between the symbol and the number. Unfortunately, CA-Visual Objects® will give you no help in dealing with these exceptions. You will have to read the `iCurrency` and `iNegCurr` entries from `WIN.INI` and handle the necessary formatting in your code using user-defined functions.

For example, a valid picture format would be:

999,999,999.99

Picture template strings in this format will allow the end user to alter the entries in Windows International Settings that the developer cannot control. This makes the actual thousand and decimal separators used in the Picture Template string simply place holders. These are not the actual characters that will be used. It will use the character specified in the Windows settings.

Measurement system.

The setting for the Measurements system is used mainly to determine what the user wants as a unit of measure for length or for capacity. For example, you should use this setting to set a ruler in your application. A setting for Metric will put centimeters on the ruler. A setting for English will put inches. It is useful information for applications using this type of data.

Overriding the Windows settings.

CA-Visual Objects® has several functions for overriding the time and amount-related `WIN.INI` settings. You can also use these functions to interrogate the settings without having to read `WIN.INI` directly.

Function	Overrides
<code>SetAMExt()</code>	<code>s1159</code>
<code>SetAMPM()</code>	<code>itime</code>
<code>SetDecimal()</code>	<code>idigits</code>
<code>SetDecimalSep()</code>	<code>sdecimal</code>
<code>SetPMExt()</code>	<code>s2359</code>
<code>SetThousandSep()</code>	<code>sthousand</code>
<code>SetTimeSep()</code>	<code>stime</code>

CA-Visual Objects® functions for overriding `WIN.INI` settings.

All of these functions, except `SetAMExt()` and `SetPMExt()`, also return the current setting of the corresponding item. To find the current settings of `s1159` and `s2359`, use `GetAMExt()` and `GetPMExt()`, respectively.

Configuring and Using the International Support of Windows 2000 and the Windows 2000 MultiLanguage Version

The Microsoft Windows 2000 family of operating systems offers support for the languages and cultural conventions used in approximately 120 international locales. No matter whether you're using the English version or the Arabic version, you'll find you can still input, process and display text in Greek, Japanese, Korean or any of the other languages supported in Windows 2000.

What is the Windows 2000 MultiLanguage Version?

The Windows 2000 MultiLanguage Version is a separate, standalone release of Windows 2000, which will ship as both Professional and Server editions. It allows the user interface language of the operating system to be changed according to the preferences of individual users. This allows large corporations to roll out Windows 2000 worldwide with a single installation job. Local users can then select the user interface language, or it can be set by Group Policy for Organizational Units.

The Windows 2000 MultiLanguage Version allows users of different languages to share the same workstation; one user might choose to see system menus, dialogs and other text in Japanese, while another user logging onto the same system might prefer to see the corresponding text in French.

Note: The Windows 2000 MultiLanguage Version is **NOT**:

- A replacement for completely localized different language versions of Windows 2000 (the installation language is English, so files such as INFs and the registry remain in English).
- The only version of Windows 2000 to allow the input, processing and display of other languages in applications like word processors. Rather, this ability is a core feature of every Windows 2000 version, including the Windows 2000 MultiLanguage Version.
- A retail product. It is only sold through Volume Licensing programs like Microsoft Open License Program (MOLP / Open), Select, and Enterprise agreements.

The following text will help guide you through the process of setting up and configuring your Windows 2000 system to use whichever languages and regional settings you need.

Basic concepts.

This document introduces some of the key concepts you'll need to understand as you configure the multilingual and international settings of the Windows 2000 family of operating systems.

Language Groups

Language groups are central to the international support provided in Windows 2000. There are 17 language group modules, each of which is a collection of all the files and settings necessary to support a given set of languages and locales. Some language groups, such as Greek, contain support for only one locale / language. Others, such as the Western Europe & US language group, provide support for many different languages and locales. Only the Western Europe & US language group is installed by default (the other modules can be installed according to demand).

Language Groups determine which locales and languages can be used on a system. The appropriate language group must be installed before a language can be input, processed or displayed, and before locale settings can be specified. You can find a complete chart of the Windows 2000 language groups and the languages / locales they support in our List of Language Groups.

Locales

In Windows 2000, locales are settings which reflect languages and cultural conventions as they are used in different parts of the world -- "German_Swiss" perhaps, as opposed to "German_Standard". Several different kinds of locales are supported:

User Locale

The user locale is a per user setting which determines the formats used to display dates, times, currency, and numbers, and the sorting order of text. A user locale is specified for each and every account created on a machine. During unattended installations the user locale is specified as a four-digit hexadecimal value: 0411 or 0c0a, for example.

System Locale

The system locale is a system-wide setting which affects all users of a machine. It determines which codepages and associated bitmap font files are used as defaults for the system. These codepages and fonts enable non-Unicode applications to run as they would on a system localized to the language of the system locale. ONLY non-Unicode applications are affected by this setting. During unattended installations the system locale is specified as a four-digit hexadecimal value: 0411 or 0c0a, for example.

Input Locale

Input locales are pairings of a language with an input method (which might be a particular keyboard layout, an Input Method Editor, or speech-to-text converter). An input locale describes the language being entered, and how it is being entered. During unattended installations input locales are specified as a pairing of a four-digit hexadecimal locale value with a keyboard layout: `0436:00000409`, for example.

User Interface (Menus and Dialogs) Languages

The Windows 2000 MultiLanguage Version includes user interface language (MUI) modules in addition to the standard Windows 2000 Language Group files. The MUI modules, one for each of the 24 user interface languages supported, contain the resources necessary to display the menus, dialogs and Help files of the system in the given language. Before a user interface language can be used the appropriate Language Group must be installed.

ID Values

In Windows 2000 and the Windows 2000 MultiLanguage Version all the Language Groups, locales and user interface languages are represented by ID numbers, rather than by names. When installing the system in unattended mode, you should refer to the ID for each entity. You can find lists of all the IDs supported in Windows 2000 linked to in the sidebar at right.

Configuring the system during setup.

How to Install the Windows 2000 MultiLanguage Version from the Command Line

To facilitate in quiet mode installations, the MUI setup program (muisetup.exe) accepts parameters entered at the command line. This can be useful either during an unattended installation of the Windows 2000 MultiLanguage Version, or simply during the addition and/or removal of user interface languages on the system.

To use these command line parameters, use the command line to navigate to the directory containing the muisetup program, and enter **muisetup.exe** followed by any of the following accepted switches:

- i (specifies the user interface language(s) to be installed)
- d (specifies the default user interface language (applied to all new user accounts and used in places such as the Winlogon screen))
- u (specifies the user interface language(s) to be uninstalled)
- r (specifies that the reboot message should not be displayed)
- s (specifies that the installation complete message should not be displayed)

The -i, -d and -u switches take languages as four-digit hexadecimal LANGID values. LANGIDs should be separated by a space as in the following example:

```
e:\muisetup.exe -i 0411 0409 0c0a -d 0411 -u 0414 040c
```

See the List of LANGIDs for all the four-digit hexadecimal LANGID values supported by the Windows 2000 MultiLanguage Version.

Configuring the system post setup.

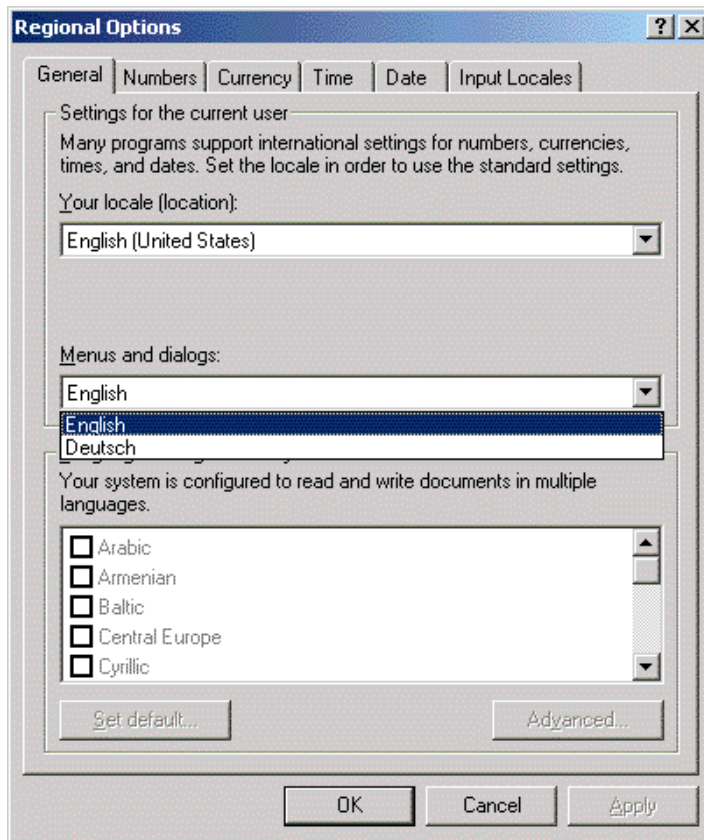
How to Change the Language Used to Display the Menus and Dialogs

[Windows 2000 MultiLanguage Version only]

The Windows 2000 MultiLanguage Version makes it easy for users to change the user interface (menus and dialogs) into the language that best suits them. Up to 24 different languages can be installed on the machine by an administrator (any user with administrative privileges) using the muisetup.exe program.

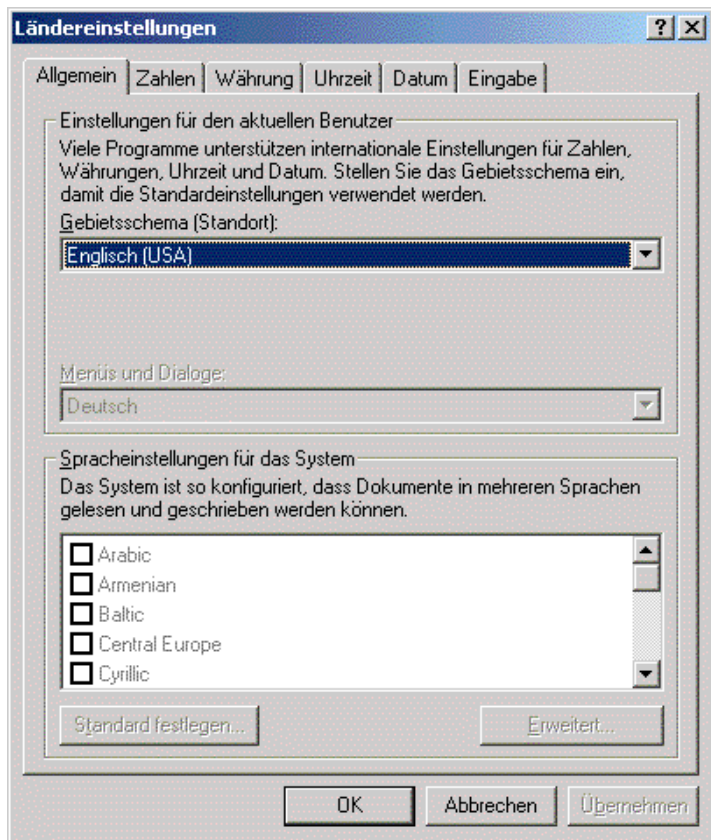
Users select any one of the installed user interface languages* by opening the Regional Options control panel and choosing one of the languages displayed in the 'Menus and dialogs' control.

Changing the user interface language has the effect of displaying menus, dialogs and Help files in the specified language. Changes to the user interface language only become effective after logging off and logging on again.



In this example, German and English are the only user interface languages installed on the system. The user can select either of them, according to preference or requirements.

* Note that network or system administrators can restrict the choice of user interface language through the use of Group Policy. If policy is set for a particular user, that user will find the 'Menus and dialogs' control disabled in the Regional Options control panel:



Here a Group Policy applied to the user account has restricted the user to the German user interface language.

Restrict the User's Choice of the Language Used to Display Menus and Dialogs

[Windows 2000 MultiLanguage Version only]. Please see the following WEB pages:
<http://www.microsoft.com/globaldev/win2k/setup/restrict.asp>

For additional information of this subject, please visit the following WEB site for the complete documents:
<http://www.microsoft.com/globaldev/default.asp>

Data

- What data is acceptable?

To decide what data is acceptable, first consider the country in which the application will be used. The country will determine the cultural content of the data as well as the languages in which the data is prepared.

You will also have to specify if pick list data will be translated or use as is. Proper names, addresses, and cities also are written differently within the same country. In Belgium, for example, the City of Brussels is written differently in French than in Dutch. This could lead to serious problems, even inside the same country. You need to create a special pick list that holds both translations and relates the address to the town using a key code. You should never let a user type in the town directly without going through a pick list of predefined towns. These are directives to follow at the user level. What about international branches that will exchange data between offices. Will they need to translate this data before sending it? You have to establish rules that the users will follow according to your software specifications and requirements.

In addition, the languages will affect the codepage with which the data is prepared. A *codepage* is a character set that a computer uses to display data properly, often to handle international characters. International characters include characters that have an accent, tilde, umlaut, etc. The most common special characters are the grave accent (` as in è), acute accent (´ as in é), circumflex (^ as in ê), tilde (~ as in ã), umlaut (¨ as in ä), ring (° as in å), and slash (/ as in ø), all used in conjunction with vowels. There is also the cedilla used with the c letter (as in ç or ÷). Note that most of these letters also need the equivalent in uppercase.

Character sets or codepages.

A character set is a mapping of characters to their identifying numeric values. Most of the character sets commonly used in computers are single-byte character sets in which each character is identified by a value one-byte wide. The large number of characters in Asian languages led to the development of multi-byte character sets, in particular the double-byte character set (DBCS). Microsoft® Windows NT® incorporates a new global standard for character encoding: Unicode.

With an international keyboard, you can display international characters by simply pressing the keys dedicated to those characters. If your keyboard does not have keys for international characters, you can enter such characters by using the character map provided with Windows or by pressing the ALT key in conjunction with keys on the numeric keypad. Depending on the codepage in use, the international characters can have different codes; you should utilize the codepage in use in the target country to test the output.

Data stored in Windows is tagged with a codepage, which is a table of characters and corresponding numbers in memory that Windows uses to display data properly. For example, if you enter the letter A in a .DBF file, the letter is stored on your hard disk as the number 65. When you open the file, CA-Visual Objects® inspects the loaded codepage to find the character corresponding to the number 65, and then displays the character (A) on your monitor.

Codepages correspond roughly to different alphabets. For example, Windows supplies codepages for English, German, French, Scandinavian languages, Asian languages, etc. By using a different codepage, applications can properly display characters from these different alphabets.

When working with codepages, be sure to test that the user interface and data display correctly by using the codepage designed for a particular country. If you see unexpected characters on the screen, check the underlying codepage. It may be that you do not have the good codepage loaded on your station.

Here are examples of the most popular codepages:

ANSI Code-Page Identifier examples

Identifier	Meaning
874	Thai
932	Japan
936	Chinese (PRC, Singapore)
949	Korean
950	Chinese (Taiwan, Hong Kong)
1200	Unicode (BMP of ISO 10646)
1250	Windows 3.1 Eastern European
1251	Windows 3.1 Cyrillic
1252	Windows 3.1 Latin 1 (US, Western Europe)
1253	Windows 3.1 Greek
1254	Windows 3.1 Turkish
1255	Hebrew
1256	Arabic
1257	Baltic

OEM Code-Page Identifier examples

Identifier	Meaning
437	MS-DOS United States
708	Arabic (ASMO 708)
709	Arabic (ASMO 449+ BCON V4)
710	Arabic (Transparent Arabic)
720	Arabic (Transparent ASMO)
737	Greek (formerly 437G)
775	Baltic
850	MS-DOS Multilingual (Latin I)
852	MS-DOS Slavic (Latin II)
855	IBM Cyrillic (primarily Russian)
857	IBM Turkish
860	MS-DOS Portuguese
861	MS-DOS Icelandic
862	Hebrew
863	MS-DOS Canadian-French
864	Arabic

Identifier	Meaning
865	MS-DOS Nordic
866	MS-DOS Russian (former USSR)
869	IBM Modern Greek
874	Thai
932	Japan
936	Chinese (PRC Singapore)
949	Korean
950	Chinese (Taiwan Hong Kong)
1361	Korean (Johab)

Language ID in Windows 2000.

An unattended or silent installation of Windows 2000 MultiLanguage Version requires the appropriate 4-digit LangID language setting to be specified on the command line. Windows 2000 MultiLanguage Version supports the following language IDs and language group IDs.

Language IDs

Language	LangID	Language Group
English (default)	0409	Western Europe and U.S.
French	040c	Western Europe and U.S.
Spanish	0c0a	Western Europe and U.S.
Italian	0410	Western Europe and U.S.
Swedish	041D	Western Europe and U.S.
Dutch	0413	Western Europe and U.S.
Brazilian	0416	Western Europe and U.S.
Finnish	040b	Western Europe and U.S.
Norwegian	0414	Western Europe and U.S.
Danish	0406	Western Europe and U.S.
Hungarian	040e	Central Europe
Polish	0415	Central Europe
Russian	0419	Cyrillic
Czech	0405	Central Europe
Greek	0408	Greek
Portuguese	0816	Western Europe and U.S.
Turkish	041f	Turkish
Japanese	0411	Japanese
Korean	0412	Korean
German	0407	Western Europe and U.S.
Chinese (Simplified)	0804	Simplified Chinese
Chinese (Traditional)	0404	Traditional Chinese
Arabic	0401	Arabic
Hebrew	040d	Hebrew

Group language IDs are not specified as a command-line parameter with muisetup.exe, but are used in the [RegionalSettings] section of the Unattend.txt file.

Group Language IDs

Language Group	ID	Supported Locales
Western Europe and United States	1	Afrikaans Basque Catalan Danish Dutch_Standard Dutch_Belgian English_United_States English_United_Kingdom English_Australian English_Canadian English_New_Zealand English_Irish English_South_Africa English_Jamaica English_Caribbean English_Belize English_Trinidad English_Zimbabwe English_Philippines Faeroese Finnish French_Standard French_Belgian French_Canadian French_Swiss French_Luxembourg French_Monaco German_Standard German_Swiss German_Austrian German_Luxembourg German_Liechtenstein Icelandic Indonesian Italian_Standard Italian_Swiss Malay_Malaysia Malay_Brunei_Darussalam Norwegian_Bokmal Norwegian_Nynorsk Portuguese_Brazilian Portuguese_Standard Spanish_Traditional_Sort Spanish_Mexican Spanish_Modern_Sort Spanish_Guatemala Spanish_Costa_Rica Spanish_Panama Spanish_Dominican_Republic

Language Group	ID	Supported Locales
		Spanish_Venezuela Spanish_Colombia Spanish_Peru Spanish_Argentina Spanish_Ecuador Spanish_Chile Spanish_Uruguay Spanish_Paraguay Spanish_Bolivia Spanish_El_Salvador Spanish_Honduras Spanish_Nicaragua Spanish_Puerto_Rico Swahili Swedish Swedish_Finland
Central Europe	2	Albanian Croatian Czech Hungarian Polish Romanian Serbian_Latin Slovak Slovenian
Baltic	3	Estonian Latvian Lithuanian
Greek	4	Greek
Cyrillic	5	Azeri_Cyrillic Belarusian Bulgarian Kazakh Macedonian Russian Serbian_Cyrillic Tatar Ukrainian Uzbek_Cyrillic
Turkic	6	Azeri_Latin Turkish Uzbek_Latin
Japanese	7	Japanese
Korean	8	Korean

Language Group	ID	Supported Locales
Traditional Chinese	9	Chinese_Taiwan Chinese_Hong_Kong Chinese_Macau
Simplified Chinese	10	Chinese_PRC Chinese_Singapore
Thai	11	Thai
Hebrew	12	Hebrew
Arabic	13	Arabic_Saudi_Arabia Arabic_Iraq Arabic_Egypt Arabic_Libya Arabic_Algeria Arabic_Morocco Arabic_Tunisia Arabic_Oman Arabic_Yemen Arabic_Syria Arabic_Jordan Arabic_Lebanon Arabic_Kuwait Arabic_UAE Arabic_Bahrain Arabic_Qatar Farsi Urdu
Vietnamese	14	Vietnamese
Indic	15	Hindi Konkani Marathi Sanskrit Tamil
Georgian	16	Georgian
Armenian	17	Armenian

Data indexing.

Most European nations have in their alphabets a variety of accents, tildes, and umlauts. Windows supports these diacritical markings by means of 64 special characters in the ANSI character set. Your users can enter these characters, and your program can display them, without any special action on your part.

CA-Visual Objects[®] character functions, such as Upper(), IsUpper(), and IsAlpha(), are all sensitive to the presence of these international characters. So Upper("été") correctly returns ÉTÉ, and IsAlpha("é") correctly returns TRUE. This behavior does not depend on SetInternational(), and it applies whether or not CA-Clipper[®] compatibility is in force.

String comparisons work in the same way. "ÉTÉ" is considered to be less than "FTF," for example. If you want to compare strings in a more conventional way, insert the following code before the comparison:

```
SetCollation(#Clipper)
```

This will cause characters to be compared on the basis of their numeric values, with all the normal Latin letters being evaluated lower than all the international ones. This also implies that the loaded Nation Module in CA-Visual Objects[®] will use the same collation as the language specific CA-Clipper[®] data.

To switch off this behavior, issue this function call:

```
SetCollation(#Windows)
```

SetCollation() also affects the case-sensitivity of normal alphabetical comparisons. With the #Windows setting, "a" is always less than "B"; with the #Clipper settings, "B" is less than "a."

The default for SetCollation() is whichever value is currently in force for SetInternational(). Changing SetInternational() automatically changes SetCollation().

Finally, be careful when working with indexes that contain international characters. The exact collating sequence of the index depends on the language that was in force when it was created. If you update the index when a different language setting is in force, it could become corrupted. To be safe, you might want to read the sLanguage entry in WIN.INI and disallow the updating of the index if it has changed.

If you are sharing data with a CA-Clipper[®] application or want to use the language-specific collation for indexes, you can use nation-specific files with CA-Visual Objects[®] (Nation Modules). These modules, once installed and activated using SET COLLATION TO CLIPPER, change character handling to the country specified. Your version of CA-Clipper[®] MUST also have the same collation (country specific) as the one in the CA-Visual Objects[®] Nation Module.

With CA-Visual Objects[®] 2.5x, you will need to replace CAVONT20.DLL in your CA-Visual Objects[®] directory (usually C:\CAVO25) with the nation module that you require. After a backup of the original file, this can be achieved by using the File Manager or by using the DOS COPY command, for example:

```
COPY C:\CAVO25\SPANISH.DLL C:\CAVO25\CAVONT20.DLL
```

Do not attempt to do this while a CA-Visual Objects® application is running. Only one language may be used at any one time.

Available Nation modules for CA-Visual Objects 2.5x.

Language	Filename
Croatian ASCII	CROATIA.DLL
Czech-852 ASCII	CZECH852.DLL
Czech-895 ASCII	CZECH895.DLL
Danish ASCII	DANISH.DLL
Dutch ASCII	DUTCH.DLL
Finnish ASCII	FINNISH.DLL
French ASCII	FRENCH.DLL
German ASCII	GERMAN.DLL
Hungarian-852 ASCII	HUNG852.DLL
Hungarian-CWI ASCII	HUNGCWI.DLL
Italian ASCII	ITALIAN.DLL
Norwegian ASCII	NORWEGN.DLL
Polish-852 ASCII	POL852.DLL
Polish-ISO ASCII	POL-ISO.DLL
Polish-Mazvia ASCII	POL-MAZ.DLL
Portuguese Brazilian	BRAZIL.DLL
Portuguese-850 ASCII	PORT850.DLL
Portuguese-860-ASCII	PORT860.DLL
Romania-437 ASCII	ROMANIA.DLL
Russian ASCII	RUSSIAN.DLL
Slovakia-852 ASCII	SLOV852.DLL
Slovakia-895 ASCII	SLOV895.DLL
Slovenia-W95 ASCII	SL-W-95.DLL
Slovenia-AS7 ASCII	SL-W-AS7.DLL
Slovenia-895 ASCII	SL-W-EE.DLL
Spanish ASCII	SPANISH.DLL
Swedish ASCII	SWEDISH.DLL

With CA-Visual Objects® 2.5x, you can use a function to activate a new DLL for nation-dependent operations and messages. This function is SetNatDLL(). The default library for nation-dependent operations is CAVONT20.DLL. SetNatDLL() disables the default DLL and loads the specified DLL as a parameter to the function that is active until the next call to SetNatDLL().

The next example sets DLL CAVONT20.GER for nation-dependent operations:

```
? GetNatDLL()           // CAVONT20.DLL
? SetNatDLL("CAVONT20.GER") // .T.
? GetNatDLL()           // CAVONT20.GER
```

The function GetNatDLL() returns the current DLL for nation-dependent operations and messages.

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 (CAVONT20.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 CAVONT20.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(), 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.

Multilingual data.

In some countries, it is better to leave proper names and addresses in the native language. Sometimes it could be useless and troublesome to translate them. There are some exceptions though; in some countries, there is more than one official language. In these, the developer will have to take care of every exception. For example, as mentioned earlier, in Belgium, the City of Brussels is written differently in French than in Dutch. One suggestion would be to create pick lists for street and city names, with both translations in the description, and then store this information as a code to connect the master table and the detail's table. You should never let a user type in the town directly, as you will end up with duplication in your tables if you rely on this information for a uniqueness check. The search will not find your record if the user enters the search key in the other language. You, as a developer, have the responsibility to build software that will take care of the exceptions and you will have to define the rules when these exceptions have no work around.

If your end users are with multinational enterprises, maybe some data will be exchanged from one branch to other branches or to the main office. If your data has to be stored in a local language using local codepages, this could lead to problems when you merge the data with other languages and codepages. In this case, your communication module will have to translate this data into another codepage or Unicode before sending it; there are many CA-Visual Objects[®] functions or Windows API functions to do this. This part of the development process is more related to database architecture, but it will change the structure of your program. So, this subject is very important and must be defined before the development cycle. Once the rules are established, you will have fewer undesired surprises when the alpha testing cycle will start in another country.

The best way to avoid such codepage problems would be to use Unicode.

Unicode and Double-Byte Characters

Some languages, such as Chinese, Korean, and Japanese, use *DBCS* (double-byte character sets) to represent their data. If your application might run in these environments, you might need to use special string-handling functions and collation sequences for the application to work properly.

Definition.

The first and most prominent character standard in use by computers today is ASCII. This format is adequate for western languages, but as computers became more popular in all countries, the limitations of ASCII became clear.

In an effort to overcome some of these imitations, the International Standards Organization (ISO) established a new standard called Latin-1 that defined European characters that were omitted from ASCII.

Microsoft Windows[®] modified the Latin-1 standard even further and called the character set Windows ANSI. However, by continuing use of an 8-bit coding scheme, ASCII is only capable of representing 256 unique symbols -- considerably less than the 10,000 symbols that are common in such languages as Cyrillic, Chinese, Korean, and Japanese.

In addition to the language barriers, as the capabilities of computers broaden beyond uppercase, monospaced fonts, the requirements for a large set of unique characters (for example, letters, punctuation, mathematical and technical symbols, and publishing characters) have also grown far beyond the capabilities of 8-bit text.

The lowest level of localization (adaptation to a particular language) is the actual binary representation of characters: the code set. To overcome the limitations of the other coding methods, several major computer companies, including Apple Computer Inc.[®], Sun Microsystems Inc.[®], Xerox Corp.[®], and IBM[®] (International Business Machines Corp.), formed Unicode Inc.[®], a non-profit consortium, to set out to define a new standard for international character sets. At the same time, the ISO began developing a standard. Eventually, these standards merged and became Unicode. Unicode is published as The Unicode Standard, Worldwide Character Encoding.

Unicode employs a 16-bit coding scheme that allows for 65,536 distinctive characters--more than enough to include all languages in use today. In addition, it supports several archaic or arcane languages such as Sanskrit and Egyptian hieroglyphs. Unicode also includes representations for punctuation marks, mathematical symbols, and dingbats, with room left for future expansion. Because it establishes a unique code for each character in each script, Windows NT[®] can insure that the character translation from one language to another is accurate.

ANSI strings use one byte per character if they do not contain any DBCS characters. UNICODE strings use two bytes per character.

Single-Byte Character Set

A single-byte character set is a mapping of 256 individual characters to their identifying numeric values. The character codes 0x20 through 0x7E represent standardized displayable characters, but the characters represented by the remaining codes vary among character sets. The ASCII character set covers the range 0x00 through 0x7F.

In Windows, the ANSI character set is used in window manager and graphics device interface (GDI), but the MS-DOS file allocation table (FAT) file system uses a character set called the original equipment manufacturer (OEM) character set. Variations on the character sets, called codepages, include different special characters, typically customized for a language or group of languages. The OEM codepage generally used in the United States is codepage 437.

Applications that use the Microsoft[®] Win32[®] application programming interface (API) can use Unicode to avoid the inconsistencies of varied codepages and as an aid in developing easily localized applications.

If we have a look at the Windows API, your application can use the GetACP() function to retrieve the ANSI code-page identifier for the system or use the GetOEMCP() function to retrieve the OEM code-page identifier.

The OemToChar() and OemToCharBuff() functions allow an application to convert a character or string from the OEM codepage to either the ANSI codepage or Unicode. To convert in the other direction, you can use either the CharToOem() or CharToOemBuff() function. In addition, an application can use the MultiByteToWideChar() and WideCharToMultiByte() functions to map single-byte character set (SBCS) strings to Unicode and Unicode strings to SBCS strings.

The GetCPInfo() function fills a CPINFO structure with information that includes the size, in bytes, of the largest character in the codepage and the default character used when a character code is entered that has no corresponding entry in the codepage.

In CA-Visual Objects[®], the built in functions are: Ansi2Oem(), Ansi2OemA(), Ansi2OemBuff(), Oem2Ansi(). They are mostly conversion functions from OEM to ANSI and vice-versa. They act as wrappers for the API functions. The API functions may be better because they are new to the 32-bit API and replace the old 16 bits that are now obsolete.

Double-Byte Character Set (DBCS)

The double-byte character set (DBCS) is called an expanded 8-bit character set because its smallest unit is a byte. It can be thought of as the ANSI character set for some Asian versions of Windows (particularly the Japanese version). However, unlike the handling of Unicode, DBCS character handling requires detailed changes in the character-processing algorithms throughout an application's source code.

An application can use the `IsDBCSLeadByte()` function to determine whether a given character is the first byte in a 2-byte character; this helps identify double-byte character sets. In addition, an application can use the `MultiByteToWideChar()` and `WideCharToMultiByte()` functions to map DBCS strings to Unicode and Unicode strings to DBCS strings.

Unicode

Unicode is a worldwide character-encoding standard. Windows NT/2000 uses it exclusively at the system level for character and string manipulation. Unicode simplifies localization of software and improves multilingual text processing. By implementing it in an application, a developer can enable the application with universal data exchange capabilities for global marketing, using a single binary file for every possible character code. Windows 9x only offers very limited support for Unicode. Therefore CA-Visual Objects 2.x does not support development of native Unicode applications.

Unicode defines semantics for each character, standardizes script behavior, provides a standard algorithm for bi-directional text, and defines cross-mappings to other standards. Among the scripts supported by Unicode are Latin, Greek, Han, Hiragana, and Katakana. Supported languages include, but are not limited to, German, French, English, Greek, Chinese, and Japanese.

Unicode can represent all the world's characters in modern computer use, including technical symbols and special characters used in publishing. Because each Unicode character is 16 bits wide, it is possible to have separate values for up to 65,536 characters. Unicode-enabled functions are often referred to as "wide-character" functions.

Win32 functions support applications that use either Unicode or the regular ANSI character set. Mixed use in the same application is also possible. Adding Unicode support to an application is easy, and a developer can even maintain a single set of sources from which to compile an application that supports either Unicode or the Windows ANSI character set.

Win32 functions support Unicode by assigning its strings a specific data type and providing a separate set of entry points and messages to support this new data type.

Unicode and your operating system.

Unicode in Windows 2000®

Windows 2000 supports Unicode Version 2.0. If you run a program that uses Unicode on a Windows 2000-based computer, you can input text from another language. For example, a program that uses the English language would support Japanese text. Windows 2000 APIs are fully functional in every language edition of the operating system.

You need to restart your Windows 2000-based computer when changing program environments if they use a specific codepage. For example, you need to restart your computer when you switch between a multi-language program that uses codepage 932 for Japanese and a Russian program that uses codepage 932. However, Unicode-based programs do not require that you restart your computer.

Microsoft Windows 95/98, Microsoft Windows NT 4.0 and Windows 2000 contain tables that convert text between ANSI character encodings and Unicode. A software developer can add a conversion table to allow Windows to use programs that do not use Unicode, including UNIX and Macintosh character encodings.

Windows 95/98 does not contain native support for Unicode, but does support several wide character API's, such as the TextOutW API.

In Windows NT, IME APIs were either unavailable or added onto non-Asian language editions of Windows NT. A program that does not use Unicode can add code to trap IME messages to use Japanese text.

For information about Unicode version 2.0, please visit the following Web site:

<http://www.unicode.org/>

Unicode in Windows NT®

Windows NT® is the first widely available operating system to be built upon the Unicode character encoding. Almost all of the strings used in the system have 16bits reserved for each character. However, Windows NT® does not yet realize the Unicode ideal of offering an editor capable of handling one document containing all of the languages of the world.

Because the industry is moving to Windows NT®, and Windows NT® supports Unicode, many software vendors want to convert existing ASCII (or ANSI) help and resource files into Unicode.

Unicode is the native code set of Windows NT®, but the Win32 subsystem provides both ANSI and Unicode support. Character strings in the system, including object names, path names, and file and directory names are represented with 16-bit Unicode characters. The Win32 subsystem converts any ANSI characters it receives into Unicode strings before manipulating them. It then converts them back to ANSI, if necessary, upon exit from the system.

Some API functions exist only in wide-character versions and can be used only with the appropriate data type.

In Windows, the ANSI character set is used in the window manager and GDI; the MS-DOS FAT file system uses the OEM character set. Windows applications that create MS-DOS files have sometimes had to use the CharToOem() and OemToChar() functions to translate between these character sets. However, the New Technology File System (NTFS) is capable of storing filenames in Unicode; no translation is necessary with NTFS.

With Unicode implementations of the file-system functions, it is not necessary to perform translations to and from ANSI and OEM character sets.

The special filename characters in MS-DOS are unchanged in Unicode filenames:

“\”, “/”, “:”, “?”, “*”

These special characters are in the ASCII range of characters (0x00 through 0x7F) and their Unicode equivalents are simply the same values in a 2-byte form: 0x0000 through 0x007F.

Unicode and Character Set Functions

The following are the character set functions. On errors, some of these functions set an extended error value. Use the GetLastError() function to retrieve this value.

GetTextCharset	This function obtains a character-set identifier for the font that is currently selected for a specified device context. If the function succeeds, the return value identifies the character set of the font that is currently selected for the specified device context.
GetTextCharsetInfo	This function obtains information about the character set of the font that is currently selected for a specified device context. If the function succeeds, the return value identifies the character set of the font currently selected for the specified device context; but it also stores info about the font to a structure (font signature)
IsDBCSLeadByteEx	This function determines whether a character is a lead byte. That is, the first byte of a character in a double-byte character set (DBCS). If the function succeeds, the return value is nonzero.
TranslateCharsetInfo	This function translates based on the specified character set, codepage, or font signature value, setting all members of the destination structure to appropriate values.
WideCharToMultiByte	This function maps a wide-character string to a new character string. The new character string is not necessarily from a multi-byte character set.

These functions can be used to work with Unicode strings. When converting ASCII to Unicode, remember that the entire set of ASCII characters maps perfectly to the first characters in Unicode. You need only add a second null bit and a 0x00 in the high byte.

Supported or unsupported?

The Rich Edit control included with Windows 95® and Windows NT® version 3.51 does not support Unicode. For a Unicode application to use the Rich Edit control, it must convert any strings passed to the control to ASCII text.

Unicode support in Windows NT®:

- All Windows USER objects support Unicode strings.
- The Win32 console is Unicode enabled.
- NTFS supports Unicode filenames.
- All of the information strings in the registry are Unicode.
- The L_10646.TTF (Lucida Sans Unicode) font covers over 1300 Unicode characters.
- Most of the TrueType fonts include a Unicode encoding table.

Unicode features missing from Windows NT®:

- There is no font support for all of the Unicode characters.
- The FAT and HPFS file systems do not support Unicode filenames. (Nor will they in the future; to accomplish this, use NTFS.)

Unlike Windows NT®, Windows 95® does not implement the Unicode (or wide character) version of most Win32 functions that take string parameters.

With some exceptions, these functions are implemented as stubs that simply return success without modifying any arguments.

In general, Windows 95® implements the ANSI version of these functions.

One major exception to this rule is OLE. All native 32-bit OLE APIs and interface methods use Unicode exclusively. For more information on this, please see the OLE documentation.

Excluding OLE, Windows 95® supports the wide character version of the following functions:

- EnumResourceLanguages()
- EnumResourceNames()
- EnumResourceTypes()
- ExtTextOut()
- FindResource()
- FindResourceEx()
- GetCharWidth()
- GetCommandLine()
- GetTextExtentExPoint()
- GetTextExtentPoint32()
- GetTextExtentPoint()
- lstrlen()
- MessageBoxEx()
- MessageBox()
- TextOut()

In addition, Windows 95® implements the following two functions for converting strings to or from Unicode:

- MultiByteToWideChar()
- WideCharToMultiByte()

Double byte languages.

If you are working in a DBCS environment, you can use an Input Method Editor to input characters into the application. The IME is an application provided with your environment that allows you to type characters on the keyboard to display a selection of international characters and then choose the specific character you want. For example, an IME for Chinese might allow you to enter a Pinyin representation of a Chinese word and then display a list of characters that match the representation. When you select the character you want, the IME pastes it into the application.

Double Byte and CA-Visual Objects®

CA-Visual Objects® includes functions for manipulating character expressions containing any combination of single-byte or double-byte characters. By using DBCS string functions, you can develop applications without having to write extra code that tests for double-byte characters when counting, locating, inserting, or removing characters in a string.

Most DBCS functions are equivalent to their single-byte counterparts except that they are named with an MB prefix to distinguish them. A few other functions help you work with strings specifically in double-byte environments, like JCDoW(), which is the equivalent of DoW() and extracts the Japanese name of the day of the week from a date.

Double-byte (DBCS) specific functions

Functions	Description
DBToSB()	Convert double-byte kana characters in a string to their single-byte equivalents.
IsKanji()	Determine if the first character of a string is a kanji character.
JCDoW()	Extract the Japanese name of the day of the week from a date.
JCMonth()	Extract the Japanese name of the month from a date.
JCYear()	Extract the wareki, or Japanese year, from a date.
JNToCDoW()	Convert the number that identifies a day into the Japanese name of the day.
JNToCMonth()	Convert the number that identifies a month into the Japanese name of the month.
JNToCYear()	Convert a year specified as a number to its wareki, or Japanese year, equivalent.
MBAAllTrim()	Remove leading and trailing spaces--including double-byte spaces--from a string.
MBAAt() MBAAt2() MBAAt3()	Return the position of the first occurrence of a substring within a string--both the substring and the string can contain double-byte characters.
MBAAtC() MBAAtC2()	Return the position of the first occurrence of a substring within a string, without regard for case--both the substring and the string can contain double-byte characters.
MBAAtLine() MBAAtLine2()	Return the line number of the first occurrence of a substring within a multiple line string--both the substring and the string can contain double-byte characters.
MBLeft()	Return a substring beginning with the first character of a string containing double-byte characters.

Functions	Description
MBLen()	Return the length of a string containing double-byte characters or an array.
MBLTrim()	Remove leading spaces--including double-byte spaces--from a string.
MBPszLen()	Return the length of a PSZ containing double-byte characters.
MBRAAt() MBRAAt2() MBRAAt3()	Return the position of the last occurrence of a substring within a string--both the substring and the string can contain double-byte characters.
MBRright()	Return a substring beginning with the last character of a string containing double-byte characters.
MBRTrim()	Remove trailing spaces--including double-byte spaces--from a string.
MBSLen()	Return the length of a strongly typed string containing double-byte characters.
MBSstuff()	Insert a string into another string, optionally deleting a specified number of characters from the original string--both strings can contain double-byte characters.
MBSsubstr() MBSsubstr2() MBSsubstr3()	Extract a substring from a string--both the substring and the string can contain double-byte characters.
MBTrim()	Remove trailing spaces--including double-byte spaces--from a string.
SBToDB()	Convert single-byte kana characters in a string to their double-byte equivalents.
ToHira()	Convert single-byte and double-byte katakana characters in a string to their double-byte hiragana equivalents.
ToJNum()	Convert the single-byte and double-byte numbers in a string to double-byte kanji numbers.

Left to right languages.

Some languages, like Arabic or Hebrew, are languages where the writing is done from left to right. This means that the controls are or should be reversed on your window to fit the language. This presents more problems to the programmer writing multi-lingual applications that will take care of the two types of languages, right-to-left and left-to-right. You will have to store the controls and label coordinates with your strings. This is a real challenge for programmers, and even more if you are not a native of the language.

Windows Codepages

The list below provides graphical representations and textual listings for each of the Windows codepages.

SBCS (Single-Byte Character Set) Codepages

Microsoft Windows OEM Codepage : 437 (US)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	<u>DEL</u> 007F
80	Ç	ü	é	â	ä	à	å	ç	ë	è	è	ï	î	ì	Ä	Å
90	É	æ	Æ	ö	ö	ò	û	ù	ý	Ö	Ü	¢	£	¥	€	f
A0	á	í	ó	ú	ñ	Ñ	ª	º	¿	¬	¼	½	¾	¿	«	»
B0	▯	▯	▯		┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘
C0	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘
D0	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘	┘
E0	α	β	Γ	Π	Σ	σ	μ	τ	Φ	Θ	Ω	δ	∞	φ	ε	π
F0	≡	±	≥	≤	∫	∫	÷	≈	°	·	·	√	²	²	■	<u>NBSP</u> 00A0

Microsoft Windows Codepage: 1250 (Central Europe)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	! 0021	" 0022	# 0023	\$ 0024	% 0025	& 0026	' 0027	(0028) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	}	~ 007E	<u>DEL</u> 007F
80	€ 20AC	• 2018	/ 201A	• 201C	• 201E	• 2026	• 2020	• 2021	• 2030	Š 0160	< 2039	Š 015A	Ť 0164	Ž 017D	Ž 0179	Ž 017A
90	• 2018	• 2019	• 201C	• 201D	• 2022	• 2013	• 2014	• 2122	Š 0161	> 203A	Š 015B	Ť 0165	Ž 017E	Ž 017A		
A0	<u>NBSP</u> 00A0	ˆ 02C7	ˆ 02D8	Ł 0141	* 00A4	Ą 0104	! 00A6	Ś 00A7	• 00A8	© 00A9	§ 015E	« 00AB	¬ 00AC	- 00AD	@ 00AE	Ž 017B
B0	° 00B0	± 00B1	ˆ 02DB	ł 0142	ˆ 00B4	μ 00B5	¶ 00B6	• 00B7	ˆ 00B8	ą 0105	§ 015F	» 00BB	ł 013D	ˆ 02DD	ł 013E	ż 017C
C0	Ŕ 0154	Á 00C1	Ā 00C2	Ǻ 0102	Ǻ 00C4	Ǻ 0139	Ć 0106	Ç 00C7	Č 010C	É 00C9	Ě 0118	Ě 00CB	Ě 011A	Í 00CD	Ī 00CE	Ď 010E
D0	Đ 0110	Ń 0143	Ň 0147	Ó 00D3	Ö 00D4	Ö 0150	Ö 00D6	× 00D7	Ř 0158	Ů 016E	Ú 00DA	Ů 0170	Ů 00DC	Ů 00DD	Ý 0162	ß 00DF
E0	ŕ 0155	á 00E1	ā 00E2	ǻ 0103	ǻ 00E4	ǻ 013A	ć 0107	ç 00E7	č 010D	é 00E9	ě 0119	ě 00EB	ě 011B	í 00ED	ī 00EE	ď 010F
F0	đ 0111	ń 0144	ň 0148	ó 00F3	ö 00F4	ö 0151	ö 00F6	÷ 00F7	ř 0159	ů 016F	ú 00FA	ů 0171	ů 00FC	ů 00FD	ý 0163	• 02D9

Microsoft Windows Codepage: 1251 (Cyrillic)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	<u>DEL</u> 007F
80	Ѡ	ѡ	Ѣ	ѣ	Ѥ	ѥ	Ѧ	ѧ	Ѩ	ѩ	Ѫ	ѫ	Ѭ	ѭ	Ѯ	ѯ
90	Ѱ	ѱ	Ѳ	ѳ	Ѵ	ѵ	Ѷ	ѷ	Ѹ	ѹ	Ѻ	ѻ	Ѽ	ѽ	Ѿ	ѿ
A0	<u>NBSP</u> 00A0	ѿ	ѿ	Ј	*	Г	!	§	Ё	©	€	«	¬	-	@	Ї
B0	°	±	І	і	Ҁ	μ	¶	·	ё	№	е	»	ј	ѕ	ѕ	ї
C0	А	В	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
D0	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
E0	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
F0	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я

Microsoft Windows Codepage: 1252 (Latin I)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	! 0021	" 0022	# 0023	\$ 0024	% 0025	& 0026	' 0027	(0028) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	}	~ 007E	<u>DEL</u> 007F
80	€ 20AC	• 2018	‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	% 2030	Š 0160	< 2039	€ 0152	• 017D	Ž 017D	• 0178
90	• 2018	‚ 2019	ƒ 201C	„ 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	š 0161	> 203A	œ 0153	• 017E	ž 017E	ÿ 0178	• 0178
A0	<u>NBSP</u> 00A0	ı 00A1	ı̇ 00A2	£ 00A3	* 00A4	¥ 00A5	ı̇ 00A6	§ 00A7	• 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	– 00AD	® 00AE	— 00AF
B0	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	¹ 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C0	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D0	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E0	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F0	ø 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

Microsoft Windows Codepage: 1253 (Greek)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	! 0021	" 0022	# 0023	\$ 0024	% 0025	& 0026	' 0027	(0028) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	}	~ 007E	<u>DEL</u> 007F
80	€ 20AC	• 2018	‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	• 2030	◁ 2039	◁ 2039	◁ 2039	◁ 2039	◁ 2039	◁ 2039	◁ 2039
90	◁ 2039	◁ 2039	◁ 2039	◁ 2039	◁ 2039	◁ 2039	◁ 2039	◁ 2039	◁ 2122	◁ 203A	◁ 203A	◁ 203A	◁ 203A	◁ 203A	◁ 203A	◁ 203A
A0	<u>NBSP</u> 00A0	“ 0385	” 0386	£ 00A3	* 00A4	¥ 00A5	¡ 00A6	§ 00A7	¨ 00A8	© 00A9	◁ 00AB	¬ 00AC	– 00AD	® 00AE	– 2015	
B0	° 00B0	± 00B1	² 00B2	³ 00B3	´ 0384	µ 00B5	¶ 00B6	· 00B7	È 0388	É 0389	Ï 038A	» 00BB	Ù 038C	Ú 00BD	Ý 038E	Ω 038F
C0	Í 0390	Α 0391	Β 0392	Γ 0393	Δ 0394	Ε 0395	Ζ 0396	Η 0397	Θ 0398	Ι 0399	Κ 039A	Λ 039B	Μ 039C	Ν 039D	Ξ 039E	Ο 039F
D0	Π 03A0	Ρ 03A1	• 03A2	Σ 03A3	Τ 03A4	Υ 03A5	Φ 03A6	Χ 03A7	Ψ 03A8	Ω 03A9	Ï 03AA	ÿ 03AB	ά 03AC	έ 03AD	ή 03AE	ί 03AF
E0	ύ 03B0	α 03B1	β 03B2	γ 03B3	δ 03B4	ε 03B5	ζ 03B6	η 03B7	θ 03B8	ι 03B9	κ 03BA	λ 03BB	μ 03BC	ν 03BD	ξ 03BE	ο 03BF
F0	π 03C0	ρ 03C1	ς 03C2	σ 03C3	τ 03C4	υ 03C5	φ 03C6	χ 03C7	ψ 03C8	ω 03C9	ϊ 03CA	ϋ 03CB	ό 03CC	ύ 03CD	ώ 03CE	• 03CF

Microsoft Windows Codepage: 1254 (Turkish)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	! 0021	" 0022	# 0023	\$ 0024	% 0025	& 0026	' 0027	(0028) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	}	~ 007E	<u>DEL</u> 007F
80	€ 20AC	• 2018	‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	% 2030	Š 0160	< 2039	€ 0152	• 0178	• 0178	• 0178
90	• 2018	‚ 2019	ƒ 201C	„ 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	Š 0161	> 203A	œ 0153	• 0178	• 0178	• 0178	ÿ 0178
A0	<u>NBSP</u> 00A0	ı 00A1	ç 00A2	£ 00A3	* 00A4	¥ 00A5	ı 00A6	§ 00A7	• 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	– 00AD	® 00AE	— 00AF
B0	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	• 00B7	¸ 00B8	¹ 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C0	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D0	Ğ 011E	Ń 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	ı 0130	Ş 015E	ß 00DF
E0	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F0	ğ 011F	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ı 0131	ş 015F	ÿ 00FF

Microsoft Windows Codepage: 1255 (Hebrew)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	<u>DEL</u> 007F
80	€ 20AC		,	f 0192	" 201E	... 2026	† 2020	‡ 2021	ˆ 02C6	% 2030		< 2039				
90		\ 2018	/ 2019	" 201C	" 201D	• 2022	— 2013	— 2014	˜ 02DC	™ 2122		> 203A				
A0	<u>NBSP</u> 00A0	ı	ı̇	£	₪	¥	ı̇	₪	ˆ	©	×	«	¬	—	®	—
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	05B0	05B1	05B2	05B3	05B4	05B5	05B6	05B7	05B8	05B9		05BB	05BC	05BD	05BE	05BF
D0	05C0	05C1	05C2	05C3	05F0	05F1	05F2	05F3	05F4							
E0	א	ב	ג	ד	ה	ו	ז	ח	ט	י	כ	ל	מ	נ	ס	ע
F0	פ	צ	ק	ר	ש	ת	ך	ץ	ן	וּ	וּ			<u>LTR</u> 200E	<u>RTL</u> 200F	

Microsoft Windows Codepage: 1256 (Arabic)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	<u>DEL</u> 007F
80	€ 20AC	پ 067E	، 201A	ف 0192	” 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	% 2030	ث 0679	< 2039	Ⓒ 0152	ع 0686	ز 0698	س 0688
90	گ 06AF	، 2018	، 2019	” 201C	” 201D	• 2022	— 2013	— 2014	ك 06A9	™ 2122	، 0691	> 203A	œ 0153	<u>ZWNJ</u> 200C	<u>ZWJ</u> 200D	و 06BA
A0	<u>NBSP</u> 00A0	، 060C	، 00A2	£ 00A3	* 00A4	¥ 00A5	! 00A6	§ 00A7	” 00A8	© 00A9	، 06BE	« 00AB	¬ 00AC	— 00AD	® 00AE	— 00AF
B0	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	د 00B8	١ 00B9	؛ 061B	» 00BB	٤ 00BC	٥ 00BD	٦ 00BE	؟ 061F
C0	^ 06C1	ء 0621	آ 0622	أ 0623	ؤ 0624	إ 0625	ؤ 0626	ا 0627	ب 0628	ة 0629	ت 062A	ث 062B	ج 062C	ح 062D	خ 062E	د 062F
D0	ذ 0630	ر 0631	ز 0632	س 0633	ش 0634	ص 0635	ض 0636	× 00D7	ط 0637	ظ 0638	ع 0639	غ 063A	— 0640	ف 0641	ق 0642	ك 0643
E0	à 00E0	ل 0644	â 00E2	م 0645	ن 0646	ه 0647	و 0648	Ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ى 0649	ي 064A	î 00EE	ï 00EF
F0	، 064B	، 064C	، 064D	، 064E	ö 00F4	، 064F	، 0650	÷ 00F7	، 0651	ù 00F9	، 0652	û 00FB	ü 00FC	<u>LTR</u> 200E	<u>RTL</u> 200F	ء 06D2

Microsoft Windows Codepage: 1257 (Baltic)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL 007F
80	€ 20AC	• 2018	/	“	”	… 2026	†	‡	• 2122	% 2030	<	•	•	•	•	•
90	• 2018	/	“	”	• 2022	—	—	•	•	•	>	•	•	•	•	•
A0	NBSP 00A0	•	¢	£	•	•	•	•	•	•	•	•	•	•	•	•
B0	°	±	²	³	´	µ	¶	•	•	•	•	•	•	•	•	•
C0	À	Ā	Ā	Ć	Ĉ	Č	Ď	Ě	Ě	Ě	Ě	Ě	Ě	Ě	Ě	Ě
D0	Š	Ń	Ń	Ó	Ō	Ŏ	×	Ū	Ł	Ś	Ū	Ū	Ź	Ź	Ź	Ź
E0	ą	ı	ā	ć	ĉ	č	ď	ě	ě	ě	ě	ě	ě	ě	ě	ě
F0	š	ń	ń	ó	ō	ö	÷	ų	ł	ś	ū	ū	ź	ź	ź	•

Microsoft Windows Codepage: 1258 (Viet Nam)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL 007F
80	€ 20AC		,	f	"	...	†	‡	~	%		<	€			
90		\	/	"	"	•	—	—	~	™		>	œ			ÿ 0178
A0	NBSP 00A0	ı	ı	£	*	¥	ı	§	ı	©	ª	«	¬	–	®	—
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	0300	Ì	Í	Î
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	U	~	ß
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	0301	ì	í	î
F0	đ	ñ	.	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ur	đ	ÿ 00FF

Microsoft Windows Codepage: 874 (Thai)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL 0000	STX 0001	SOT 0002	ETX 0003	EOT 0004	ENQ 0005	ACK 0006	BEL 0007	BS 0008	HT 0009	LF 000A	VT 000B	FF 000C	CR 000D	SO 000E	SI 000F
10	DLE 0010	DC1 0011	DC2 0012	DC3 0013	DC4 0014	NAK 0015	SYN 0016	ETB 0017	CAN 0018	EM 0019	SUB 001A	ESC 001B	FS 001C	GS 001D	RS 001E	US 001F
20	SP 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL 007F
80	€ 20AC					… 2026										
90		̀	´	¨	”	•	—	—								
		2018	2019	201C	201D	2022	2013	2014								
A0	NBSP 00A0	ก	ข	ฃ	ค	ฅ	ฉ	ช	จ	ฉ	ซ	ฌ	ญ	ญ	ฎ	ฏ
B0	ฐ	ฑ	ฒ	ณ	ด	ต	ถ	ท	ธ	น	บ	ป	ผ	ฝ	พ	ฟ
C0	ภ	ม	ย	ร	ล	ว	ศ	ษ	ส	ห	ฬ	อ	ฮ	ย	ร	๑
D0	๒	๓	๔	๕	๖	๗	๘	๙	๐	๑	๒					฿ 0E3F
E0	๑	๒	๓	๔	๕	๖	๗	๘	๙	๐	๑	๒	๓	๔	๕	๖
F0	๗	๘	๙	๐	๑	๒	๓	๔	๕	๖	๗	๘	๙	๐	๑	๒

DBCS (Double-Byte Character Set) Codepages

In the following graphical representations, light gray background shading indicates lead bytes. Each of these lead bytes has a hyperlink to a new page showing the 256-character block associated with that lead byte. A darker gray background identifies unused lead bytes.

Microsoft Windows Codepage: 932 (Japanese Shift-JIS)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[¥]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	<u>DEL</u> 007F
80		<u>81</u>	<u>82</u>	<u>83</u>	<u>84</u>	<u>85</u>	<u>86</u>	<u>87</u>	<u>88</u>	<u>89</u>	<u>8A</u>	<u>8B</u>	<u>8C</u>	<u>8D</u>	<u>8E</u>	<u>8F</u>
90	<u>90</u>	<u>91</u>	<u>92</u>	<u>93</u>	<u>94</u>	<u>95</u>	<u>96</u>	<u>97</u>	<u>98</u>	<u>99</u>	<u>9A</u>	<u>9B</u>	<u>9C</u>	<u>9D</u>	<u>9E</u>	<u>9F</u>
A0		。	「	」	、	・	ヲ	ア	イ	ウ	エ	オ	ヤ	ユ	ヨ	ツ
B0	ー	ア	イ	ウ	エ	オ	カ	キ	ク	ケ	コ	サ	シ	ス	セ	ソ
C0	タ	チ	ツ	テ	ト	ナ	ニ	ヌ	ネ	ノ	ハ	ヒ	フ	ヘ	ホ	マ
D0	ミ	ム	メ	モ	ヤ	ユ	ヨ	ラ	リ	ル	レ	ロ	ワ	ン	ニ	。
E0	<u>E0</u>	<u>E1</u>	<u>E2</u>	<u>E3</u>	<u>E4</u>	<u>E5</u>	<u>E6</u>	<u>E7</u>	<u>E8</u>	<u>E9</u>	<u>EA</u>	<u>EB</u>	<u>EC</u>	<u>ED</u>	<u>EE</u>	<u>EF</u>
F0	<u>FO</u>	<u>F1</u>	<u>F2</u>	<u>F3</u>	<u>F4</u>	<u>F5</u>	<u>F6</u>	<u>F7</u>	<u>F8</u>	<u>F9</u>	<u>FA</u>	<u>FB</u>	<u>FC</u>			

The following is an example of a sub-codepage (Japanese) using the Lead Byte 83.

Microsoft Windows Codepage: 932 (Japanese Shift-JIS)



Lead Byte = 0x83

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	ア 30A1	ア 30A2	イ 30A3	イ 30A4	ウ 30A5	ウ 30A6	エ 30A7	エ 30A8	オ 30A9	オ 30AA	カ 30AB	ガ 30AC	キ 30AD	ギ 30AE	ク 30AF	グ 30B0
50	ケ 30B1	ゲ 30B2	コ 30B3	ゴ 30B4	サ 30B5	ザ 30B6	シ 30B7	ジ 30B8	ス 30B9	ズ 30BA	セ 30BB	ゼ 30BC	ソ 30BD	ゾ 30BE	タ 30BF	ダ 30C0
60	チ 30C1	チ 30C2	ツ 30C3	ツ 30C4	ヅ 30C5	デ 30C6	デ 30C7	ト 30C8	ド 30C9	ナ 30CA	ニ 30CB	ヌ 30CC	ネ 30CD	ノ 30CE	ハ 30CF	バ 30D0
70	パ 30D1	ヒ 30D2	ビ 30D3	ピ 30D4	フ 30D5	ブ 30D6	プ 30D7	ヘ 30D8	ベ 30D9	ベ 30DA	ホ 30DB	ボ 30DC	ポ 30DD	マ 30DE	ミ 30DF	
80	ム 30E0	メ 30E1	モ 30E2	ヤ 30E3	ヤ 30E4	ユ 30E5	ユ 30E6	ヨ 30E7	ヨ 30E8	ラ 30E9	リ 30EA	ル 30EB	レ 30EC	ロ 30ED	ワ 30EE	ワ 30EF
90	ヰ 30F0	ヱ 30F1	ヲ 30F2	ン 30F3	ヴ 30F4	カ 30F5	ケ 30F6									Α 0391
A0	Β 0392	Γ 0393	Δ 0394	Ε 0395	Ζ 0396	Η 0397	Θ 0398	Ι 0399	Κ 039A	Λ 039B	Μ 039C	Ν 039D	Ξ 039E	Ο 039F	Π 03A0	Ρ 03A1
B0	Σ 03A3	Τ 03A4	Υ 03A5	Φ 03A6	Χ 03A7	Ψ 03A8	Ω 03A9									α 03B1
C0	β 03B2	γ 03B3	δ 03B4	ε 03B5	ζ 03B6	η 03B7	θ 03B8	ι 03B9	κ 03BA	λ 03BB	μ 03BC	ν 03BD	ξ 03BE	ο 03BF	π 03C0	ρ 03C1
D0	σ 03C3	τ 03C4	υ 03C5	φ 03C6	χ 03C7	ψ 03C8	ω 03C9									
E0																
F0																

Microsoft Windows Codepage: 936 (Simplified Chinese GBK)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	:	;	<	=	>	?
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	} 007D	~ 007E	<u>DEL</u> 007F
80	€ 20AC	<u>81</u>	<u>82</u>	<u>83</u>	<u>84</u>	<u>85</u>	<u>86</u>	<u>87</u>	<u>88</u>	<u>89</u>	<u>8A</u>	<u>8B</u>	<u>8C</u>	<u>8D</u>	<u>8E</u>	<u>8F</u>
90	<u>90</u>	<u>91</u>	<u>92</u>	<u>93</u>	<u>94</u>	<u>95</u>	<u>96</u>	<u>97</u>	<u>98</u>	<u>99</u>	<u>9A</u>	<u>9B</u>	<u>9C</u>	<u>9D</u>	<u>9E</u>	<u>9F</u>
A0	<u>A0</u>	<u>A1</u>	<u>A2</u>	<u>A3</u>	<u>A4</u>	<u>A5</u>	<u>A6</u>	<u>A7</u>	<u>A8</u>	<u>A9</u>	<u>AA</u>	<u>AB</u>	<u>AC</u>	<u>AD</u>	<u>AE</u>	<u>AF</u>
B0	<u>B0</u>	<u>B1</u>	<u>B2</u>	<u>B3</u>	<u>B4</u>	<u>B5</u>	<u>B6</u>	<u>B7</u>	<u>B8</u>	<u>B9</u>	<u>BA</u>	<u>BB</u>	<u>BC</u>	<u>BD</u>	<u>BE</u>	<u>BF</u>
C0	<u>C0</u>	<u>C1</u>	<u>C2</u>	<u>C3</u>	<u>C4</u>	<u>C5</u>	<u>C6</u>	<u>C7</u>	<u>C8</u>	<u>C9</u>	<u>CA</u>	<u>CB</u>	<u>CC</u>	<u>CD</u>	<u>CE</u>	<u>CF</u>
D0	<u>D0</u>	<u>D1</u>	<u>D2</u>	<u>D3</u>	<u>D4</u>	<u>D5</u>	<u>D6</u>	<u>D7</u>	<u>D8</u>	<u>D9</u>	<u>DA</u>	<u>DB</u>	<u>DC</u>	<u>DD</u>	<u>DE</u>	<u>DF</u>
E0	<u>E0</u>	<u>E1</u>	<u>E2</u>	<u>E3</u>	<u>E4</u>	<u>E5</u>	<u>E6</u>	<u>E7</u>	<u>E8</u>	<u>E9</u>	<u>EA</u>	<u>EB</u>	<u>EC</u>	<u>ED</u>	<u>EE</u>	<u>EF</u>
F0	<u>FO</u>	<u>F1</u>	<u>F2</u>	<u>F3</u>	<u>F4</u>	<u>F5</u>	<u>F6</u>	<u>F7</u>	<u>F8</u>	<u>F9</u>	<u>FA</u>	<u>FB</u>	<u>FC</u>	<u>FD</u>	<u>FE</u>	

Microsoft Windows Codepage: 949 (Korean)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	:	;	<	=	>	?
40	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
50	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	₩ 005C] 005D	^ 005E	_ 005F
60	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
70	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	 007C	}	~ 007D	<u>DEL</u> 007F
80		<u>81</u>	<u>82</u>	<u>83</u>	<u>84</u>	<u>85</u>	<u>86</u>	<u>87</u>	<u>88</u>	<u>89</u>	<u>8A</u>	<u>8B</u>	<u>8C</u>	<u>8D</u>	<u>8E</u>	<u>8F</u>
90	<u>90</u>	<u>91</u>	<u>92</u>	<u>93</u>	<u>94</u>	<u>95</u>	<u>96</u>	<u>97</u>	<u>98</u>	<u>99</u>	<u>9A</u>	<u>9B</u>	<u>9C</u>	<u>9D</u>	<u>9E</u>	<u>9F</u>
A0	<u>A0</u>	<u>A1</u>	<u>A2</u>	<u>A3</u>	<u>A4</u>	<u>A5</u>	<u>A6</u>	<u>A7</u>	<u>A8</u>	<u>A9</u>	<u>AA</u>	<u>AB</u>	<u>AC</u>	<u>AD</u>	<u>AE</u>	<u>AF</u>
B0	<u>B0</u>	<u>B1</u>	<u>B2</u>	<u>B3</u>	<u>B4</u>	<u>B5</u>	<u>B6</u>	<u>B7</u>	<u>B8</u>	<u>B9</u>	<u>BA</u>	<u>BB</u>	<u>BC</u>	<u>BD</u>	<u>BE</u>	<u>BF</u>
C0	<u>C0</u>	<u>C1</u>	<u>C2</u>	<u>C3</u>	<u>C4</u>	<u>C5</u>	<u>C6</u>	<u>C7</u>	<u>C8</u>	<u>C9</u>	<u>CA</u>	<u>CB</u>	<u>CC</u>	<u>CD</u>	<u>CE</u>	<u>CF</u>
D0	<u>D0</u>	<u>D1</u>	<u>D2</u>	<u>D3</u>	<u>D4</u>	<u>D5</u>	<u>D6</u>	<u>D7</u>	<u>D8</u>	<u>D9</u>	<u>DA</u>	<u>DB</u>	<u>DC</u>	<u>DD</u>	<u>DE</u>	<u>DF</u>
E0	<u>E0</u>	<u>E1</u>	<u>E2</u>	<u>E3</u>	<u>E4</u>	<u>E5</u>	<u>E6</u>	<u>E7</u>	<u>E8</u>	<u>E9</u>	<u>EA</u>	<u>EB</u>	<u>EC</u>	<u>ED</u>	<u>EE</u>	<u>EF</u>
F0	<u>FO</u>	<u>F1</u>	<u>F2</u>	<u>F3</u>	<u>F4</u>	<u>F5</u>	<u>F6</u>	<u>F7</u>	<u>F8</u>	<u>F9</u>	<u>FA</u>	<u>FB</u>	<u>FC</u>	<u>FD</u>	<u>FE</u>	

Microsoft Windows Codepage: 950 (Traditional Chinese Big5)

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	<u>NUL</u> 0000	<u>STX</u> 0001	<u>SOT</u> 0002	<u>ETX</u> 0003	<u>EOT</u> 0004	<u>ENQ</u> 0005	<u>ACK</u> 0006	<u>BEL</u> 0007	<u>BS</u> 0008	<u>HT</u> 0009	<u>LF</u> 000A	<u>VT</u> 000B	<u>FF</u> 000C	<u>CR</u> 000D	<u>SO</u> 000E	<u>SI</u> 000F
10	<u>DLE</u> 0010	<u>DC1</u> 0011	<u>DC2</u> 0012	<u>DC3</u> 0013	<u>DC4</u> 0014	<u>NAK</u> 0015	<u>SYN</u> 0016	<u>ETB</u> 0017	<u>CAN</u> 0018	<u>EM</u> 0019	<u>SUB</u> 001A	<u>ESC</u> 001B	<u>FS</u> 001C	<u>GS</u> 001D	<u>RS</u> 001E	<u>US</u> 001F
20	<u>SP</u> 0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	<u>DEL</u> 007F
80		<u>81</u>	<u>82</u>	<u>83</u>	<u>84</u>	<u>85</u>	<u>86</u>	<u>87</u>	<u>88</u>	<u>89</u>	<u>8A</u>	<u>8B</u>	<u>8C</u>	<u>8D</u>	<u>8E</u>	<u>8F</u>
90	<u>90</u>	<u>91</u>	<u>92</u>	<u>93</u>	<u>94</u>	<u>95</u>	<u>96</u>	<u>97</u>	<u>98</u>	<u>99</u>	<u>9A</u>	<u>9B</u>	<u>9C</u>	<u>9D</u>	<u>9E</u>	<u>9F</u>
A0	<u>A0</u>	<u>A1</u>	<u>A2</u>	<u>A3</u>	<u>A4</u>	<u>A5</u>	<u>A6</u>	<u>A7</u>	<u>A8</u>	<u>A9</u>	<u>AA</u>	<u>AB</u>	<u>AC</u>	<u>AD</u>	<u>AE</u>	<u>AF</u>
B0	<u>B0</u>	<u>B1</u>	<u>B2</u>	<u>B3</u>	<u>B4</u>	<u>B5</u>	<u>B6</u>	<u>B7</u>	<u>B8</u>	<u>B9</u>	<u>BA</u>	<u>BB</u>	<u>BC</u>	<u>BD</u>	<u>BE</u>	<u>BF</u>
C0	<u>C0</u>	<u>C1</u>	<u>C2</u>	<u>C3</u>	<u>C4</u>	<u>C5</u>	<u>C6</u>	<u>C7</u>	<u>C8</u>	<u>C9</u>	<u>CA</u>	<u>CB</u>	<u>CC</u>	<u>CD</u>	<u>CE</u>	<u>CF</u>
D0	<u>DO</u>	<u>D1</u>	<u>D2</u>	<u>D3</u>	<u>D4</u>	<u>D5</u>	<u>D6</u>	<u>D7</u>	<u>D8</u>	<u>D9</u>	<u>DA</u>	<u>DB</u>	<u>DC</u>	<u>DD</u>	<u>DE</u>	<u>DF</u>
E0	<u>EO</u>	<u>E1</u>	<u>E2</u>	<u>E3</u>	<u>E4</u>	<u>E5</u>	<u>E6</u>	<u>E7</u>	<u>E8</u>	<u>E9</u>	<u>EA</u>	<u>EB</u>	<u>EC</u>	<u>ED</u>	<u>EE</u>	<u>EF</u>
F0	<u>FO</u>	<u>F1</u>	<u>F2</u>	<u>F3</u>	<u>F4</u>	<u>F5</u>	<u>F6</u>	<u>F7</u>	<u>F8</u>	<u>F9</u>	<u>FA</u>	<u>FB</u>	<u>FC</u>	<u>FD</u>	<u>FE</u>	

The following is an example of a sub-codepage (Traditional Chinese Big5) using the Lead Byte A4.

Microsoft Windows Codepage: 950 (Traditional Chinese Big5)

Lead Byte = 0xA4

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	一 4E00	乙 4E59	丁 4E01	七 4E03	乃 4E43	九 4E5D	了 4E86	二 4E8C	人 4EBA	儿 513F	入 5165	八 516B	几 51E0	刀 5200	刁 5201	力 529B
50	匕 5315	十 5341	卜 535C	又 53C8	三 4E09	下 4E0B	丈 4E08	上 4E0A	丫 4E2B	丸 4E38	凡 51E1	欠 4E45	么 4E48	也 4E5F	乞 4E5E	于 4E9E
60	亡 4EA1	兀 5140	刃 5203	匀 52FA	千 5343	叉 53C9	口 53E3	土 571F	士 58EB	夕 5915	犬 5927	女 5973	子 5B50	孑 5B51	孓 5B53	寸 5BF8
70	小 5C0F	尢 5C22	尸 5C38	山 5C71	川 5D0D	工 5DE5	己 5DF1	巳 5DF2	巳 5DF3	巾 5DFE	干 5E72	井 5EFE	弋 5F0B	弓 5F13	才 624D	
80																
90																
A0		丑 4E11	丐 4E10	不 4E0D	中 4E2D	丰 4E30	丹 4E39	之 4E4B	尹 5C39	予 4E88	云 4E91	井 4E95	互 4E92	五 4E94	亢 4EA2	仁 4EC1
B0	什 4EC0	竹 4EC3	仆 4EC6	仇 4EC7	仍 4ECD	今 4ECA	介 4ECB	仄 4EC4	元 5143	允 5141	内 5167	六 516D	夸 516E	公 516C	冗 5197	凶 51F6
C0	分 5206	切 5207	刈 5208	匀 52FB	勾 52FE	勿 52FF	化 5316	匹 5339	午 5348	升 5347	卅 5345	卞 535E	卮 5384	友 53CB	及 53CA	反 53CD
D0	壬 58EC	天 5929	夫 592B	太 592A	夭 592D	孔 5B54	少 5C11	尢 5C24	尺 5C3A	屯 5C6F	巴 5DF4	幻 5E7B	廿 5EFF	弔 5F14	引 5F15	心 5FC3
E0	戈 6208	戶 6236	手 624B	扎 624E	支 652F	文 6587	斗 6597	斤 65A4	方 65B9	日 65E5	曰 66F0	月 6708	木 6728	欠 6B20	止 6B62	歹 6B79
F0	毋 6BCB	比 6BD4	毛 6BDB	氏 6C0F	水 6C34	火 706B	爪 722A	父 7236	爻 723B	片 7247	牙 7259	牛 725B	犬 72AC	王 738B	丙 4E19	

Glossary

Portions of this glossary are reproduced with permission from *Developing International Software*, published by Microsoft Press.

Accent: See Diacritic.

ACP: Acronym for the ANSI Codepage in use. Windows NT uses this codepage to convert to/from Unicode automatically whenever an application calls one of the 'A' entry points.

ANSI: 1) Acronym for the American National Standards Institute. 2) The Microsoft Windows ANSI character set, essentially ISO 8859/x plus additional characters, which was originally based on an ANSI draft standard. See Windows codepages for information about the characters contained in the ANSI character set.

ASCII: Acronym for American Standard Code for Information Interchange. ASCII, the US national variant of ISO 646, is a 7-bit character set encoding that contains characters for upper and lowercase English, American English punctuation, base 10 numbers, and a few control characters. Although very primitive, it's important to note that ASCII's set of 128 characters is the one common denominator contained in all the other common character sets.

Base character: One or more of:
A character that has meaning independent of other characters.
Any graphical character that is not a diacritic.
Any graphical character that is not, itself, a diacritic.

Character: The simplest element used to represent written languages. Note that the appearance of a character is not constant: a character's appearance depends on the font used as well as the context of surrounding text.

A code element. See Glyph.

Character Encoding: A one-to-one mapping from a set of characters into a set of numbers, used to represent text in software.

Codepage (also code-page, code page): An ordered set of characters in which a numeric index (codepoint value) is associated with each character. This term is generally used in the context of codepages defined by Windows 3.1 or MS-DOS, and may also be called a "character set" or charset. See Windows codepages for information about the characters contained in each codepage.

Complex Script: Scripts that require special processing to display, print, and edit. For more information, see the Complex Scripts FAQ.

Composite character (also composed character): A text element consisting of a base character AND a diacritic or accent mark. Although most common in the Latin script, other scripts (including Greek, Devanagari and Tamil) have composite characters too.

Diacritic: Any mark placed over, under, or through a character, usually to indicate a change in phonetic value from the unmarked state. A character that is attached to or overlays a preceding base character. Most diacritics are non-spacing characters that do not increase the width of the base character.

Double-byte character set (DBCS): Any 2-byte form of character encoding. Note that DBCS character sets often contain single byte characters as well. See Multi-byte character set.

Enabling: See Language Enabling.

End-User Defined Character (EUDC): A special character, such as a rare ideograph, that the user creates with an editor and assigns to a code point within a reserved range. Note that such characters are *not* automatically recognized by the system for functions such as sorting.

Floating accent: See Diacritic.

Font: A collection of glyphs for displaying text in a particular typeface.

Formatted text: Text displayed with multiple attributes, such as typeface, slant, weight, and color, and special effects such as shading, underlining, and blinking.

Globalization: Designing and implementing software so that it can support all targeted locales and user interface languages without modification to the software source itself. This processing includes enabling for all target languages, and adding NLS support for target locales.

Glyph: A graphical representation of a character.

IME: See Input Method Editor.

IMM: See Input Method Manager.

Internationalization: See Globalization.

Input language: A language to be inputted into the system, be it through the keyboard, a speech-to-text converter or an IME.

Input locale: An ordered pair consisting of an input language (LangID) and a method of inputting characters in the language. The method can be a keyboard layout, an IME, or other device provided by a vendor, such as a speech recognition engine. See the Locales and Language Groups FAQ for more information.

Input Method Editor (IME): A software program used to enter text with large character sets, such as Chinese, Japanese, and Korean.

Input Method Manager (IMM): The module in Windows that handles communication between Input Method Editors (IMEs) and applications.

Keyboard layout: A standard arrangement of characters on a keyboard that defines which keys produce particular characters or scan codes.

Language ID (LangID): A 16-bit value that identifies a language. A LangID consists of a primary language, such as Arabic, and a sub-language, such as Arabic for Saudi Arabia.

Language Enabling: Adding support to software for document content in a particular language. In this sense, to enable an application for Japanese means to modify the software so that the user can enter, display, edit, and print text containing Japanese. Modifying software so that it can be localized to a particular language. In this sense, enabling for Japanese means to modify software so that it can display Japanese text correctly in menus, dialog boxes, and other user interface elements. Note that in either sense, an enabled product may still have the user interface in English, i.e., it may not be localized. Contrast Localization.

Lead Byte: The byte value that is the first half of a double-byte character. See Double-byte character set.

Ligature: Two or more characters combined to represent a single typographical character. The modern Latin script uses only a few. Other scripts use many ligatures that depend on font and style. Some languages, such as Arabic, have mandatory ligatures; other languages have characters that were derived from ligatures, such as the German ligature of long and short "s" (*ß*) and the ampersand (&), which is the contracted form of the Latin word *et*.

Locale: A generic term indicating a set of attributes related to language and other regional/ethnic preferences. Examples include currency symbol, date and time format, calendar type, number formats, default character encoding, and keyboard layouts. Microsoft uses this term in combination with others to specify a subclass of these preferences. See the Locales and Language Groups FAQ for more information. See also Input locale, System locale, and User locale.

Locale ID (LCID): A 32-bit value that identifies a locale. An LCID consists of a LangID and a sort key ID.

Localization (to a language): Translating the user interface elements from the original language, usually English, to the target language. Contrast Language Enabling.

Logical order: The ordering of characters in text corresponding to the order in which they are inputted (such as when writing by hand, or keying in text using a keyboard). Contrast Visual order.

Multi-byte character set (MBCS): A mixed-width character set, in which some characters consist of more than one byte. A double-byte character set, which is a specific type of multi-byte character set, includes some characters that consist of two bytes.

National Language Support (NLS): The set of system functions in 32-bit Windows that contain national language support (information that is based on language and cultural convention).

Non-spacing character: a character, such as a diacritic, that has no meaning by itself, but overlaps a base character to modify it. Sometimes referred to as a "combining character".

OEMCP: Default OEM codepage of the system. The OEM codepage is the one used for conversions of MS-DOS based text mode applications.

OpenType: A font format jointly developed by Microsoft and Adobe Systems, Inc. to allow for the addition of Type 1 (PostScript) font data to a TrueType font. For more information, see Microsoft Typography's OpenType FAQ.

OpenType Layout: An extension to the OpenType format designed to provide support for international and high-end typography. See the OpenType Layout specification on Microsoft Typography's site for more information.

Plain text: A string of text to be displayed with one value for each text attribute, i.e., one typeface, one slant, and one weight. Contrast Formatted text.

Precomposed character: A single character that represents a sequence of characters, usually a combination of a base character and one or more diacritics.

Private Use Area (PUA): The area in Unicode from U+E000 through U+F8EE that is set aside for vendor-specific or user-designated characters.

Reading order: The overall direction of a sequence of text. Whereas words in a given script always flow in the direction associated with that script (e.g., LTR for Latin, RTL for Hebrew), the flow sentence itself depends on the reading order. For example, a mixture of Arabic and English text may be regarded as French embedded in an overall Arabic sentence, implying RTL reading order, or as Arabic embedded in French, implying LTR reading order.

Release delta: The time between the release of the domestic product and the release of the localized edition. See also Simultaneous ship.

Rich Text: See Formatted text.

Script: A collection of characters for displaying written text, all of which have a common characteristic that justifies their consideration as a distinct set. One script may be used for several different languages (e.g., Latin script, which covers all of Western Europe), and some written languages require multiple scripts (e.g., Japanese, which requires at least three scripts: the hiragana and katakana syllabaries and the Kanji ideographs imported from China). Note that this sense of the word script has nothing to do with programming scripts such as Perl or VB Script.

Separators: Symbols used to separate items in a list, mark the thousands place in numbers, or represent the decimal point. Different locales follow different conventions for separators.

Simultaneous ship (sim-ship): The release of localized editions of a product at the same time or soon after the domestic edition is released, usually within 30 days. See also Release delta.

Single byte character set: A character encoding in which each character is represented by 1 byte. Single-byte character sets are mathematically limited to 256 characters.

Slant: The obliqueness or tilt of the glyphs in a font. The most common slants are "regular" and "italic".

Spacing character: A character with a non-zero width. Contrast with non-spacing character.

Syllabary: A set of written characters in which each character represents a syllable -- for example, a consonant sound followed by a vowel sound. Examples of syllabaries include Japanese Katakana/Hiragana and the Indic scripts.

System locale: The locale emulated by the system, as seen by applications. For example, if the system locale for US Windows 2000 is set to Hebrew, then ANSI applications will see it as Hebrew localized Windows 2000, although the user interface of the system will still be in English. The system locale is system wide, in that it applies to all users. Changing the system locale requires a reboot. See the [Locales and Language Groups FAQ](#) for more information.

Trail byte: The byte value that is the second half of a double-byte character.

Typeface: Name given to a particular style of text. In contrast, a font is an implementation of a typeface.

Unicode: a 16-bit character encoding that contains all of the characters in common use in the world's major languages. Unicode provides an unambiguous representation of text across a range of scripts, languages and platforms.

Uniscribe: The Unicode Script Processor built into Windows 2000. Uniscribe supports line measurement, display, caret movement, character selection, justification, and linebreaking, of Unicode plaintext and rich text. Uniscribe covers most characters in Unicode. For more information, see the Uniscribe presentation given at the 13th International Unicode conference.

User locale: The user default preferences for calendar type, date format, currency, and number format. The user locale is a per-user setting, and does not require a reboot or logoff/logon. See the Locales and Language Groups FAQ for more information.

Visual order: The ordering used to display glyphs on a screen, printed page, or other medium. Usually used with bi-directional text, because reordering is required to go from logical order to visual order. Contrast Logical order.

Weight: The thickness or darkness of glyphs in a font. The most common weights are "regular" and "bold", but in some font families can include such various weights as "light", "demi", "heavy" or "extra bold", and so on.

Writing system: The collection of scripts and orthography required to represent a given human language in visual media.

Conclusion

Writing your application for international distribution can be a big challenge, but the rewards will be worth the effort. Most users will be very happy to work in their native language, and they will promote your product for you. With a good organization, a good architecture, a good tool like CA-Visual Objects[®] and teamwork between the end users and the programmers, you will make your product the best on the market.

Never forget the cultural side of translations from one language to another; the quality of the details will ensure that you will be successful.

Bibliography

CA-Visual Objects[®] 2.0, 2.5, printed and online documentation. Computer Associates.

Microsoft Knowledge Base. Microsoft Corporation[®].

Microsoft MSDN[®] CD. Microsoft Corporation[®].

Microsoft Windows NT 4.0 Resource Kit[®]. Microsoft Corporation[®].

Microsoft Win 32 API documentation. Microsoft Corporation[®].

Unicode Inc.
1965 Charleston Road
Mountain View, CA 94043
Phone (415) 961-4189

Jean-Paul Bleau is a software developer and consultant. He has been working with computers since 1978, with dBase II when it was on CP/M, with CA-Clipper® since 1986 and CA-Visual Objects® since the pre-release in 1994. He built specialized software for Fertility Centers to take complete care of electronic patients' charts in a health care environment, and he created CGPlus, a complete accounting system for CA-Visual Objects® 2.5 and CA-Clipper®. He is also the author of several utilities that share data with accounting packages like Point-of-Sales written in CA-Clipper®. Jean-Paul has worked in the US, Belgium, France, Canada and Africa, where he has had to deal with different types of taxes and accounting cultures. He has been attending CA-World since 1995. He was a speaker at many CA-World Technicon sessions and CA-World's eBusiness Solutions in Internet Time. He is also available to speak at other conferences or for corporate training. Jean-Paul can be reached by email at JPBleau@qc.aira.com

Copyright Notice

No part of this paper may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the author.

Trademark Acknowledgements

CA-Visual Objects 2.5® is a registered trademark of Computer Associates. All other product names and services identified throughout these notes are trademarks or registered trademarks of their respective companies. They are used throughout these notes for education only and for the benefit of such companies. No such uses, or the use of any trade name, are intended to convey endorsement or other affiliation with the notes.

COPYRIGHT NOTICE FOR MICROSOFT CONTENTS. Copyright © 2000 Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 U.S.A. All rights reserved.

TRADEMARKS. Microsoft, Windows, Windows NT, MSN, The Microsoft Network, Home Essentials, HomeAdvisor, Sidewalk, Expedia, Encarta, Bookshelf, PowerPoint, BackOffice, Outlook, FrontPage, Computing Central, MapPoint, CarPoint, Hotmail, WebTV, Advisor FYI, ZoneMatch, ZoneMessage, and/or other Microsoft products referenced herein are either trademarks or registered trademarks of Microsoft. The names of actual companies and products mentioned herein may be the trademarks of their respective owners. The example companies, organizations, products, people and events depicted herein are fictitious. No association with any real company, organization, product, person, or event is intended or should be inferred.