774-Netlinx Studio Unicode How-To

NetLinx Studio 2.4 and Unicode

Starting with NetLinx Studio 2.4, NetLinx now supports 16-bit Unicode characters. You can type Unicode character literals strings into you program, assigned them to variables, manipulate them using string operations, read and write Unicode characters to the file system and send Unicode strings to user interfaces for display.

Configuring NetLinx Studio

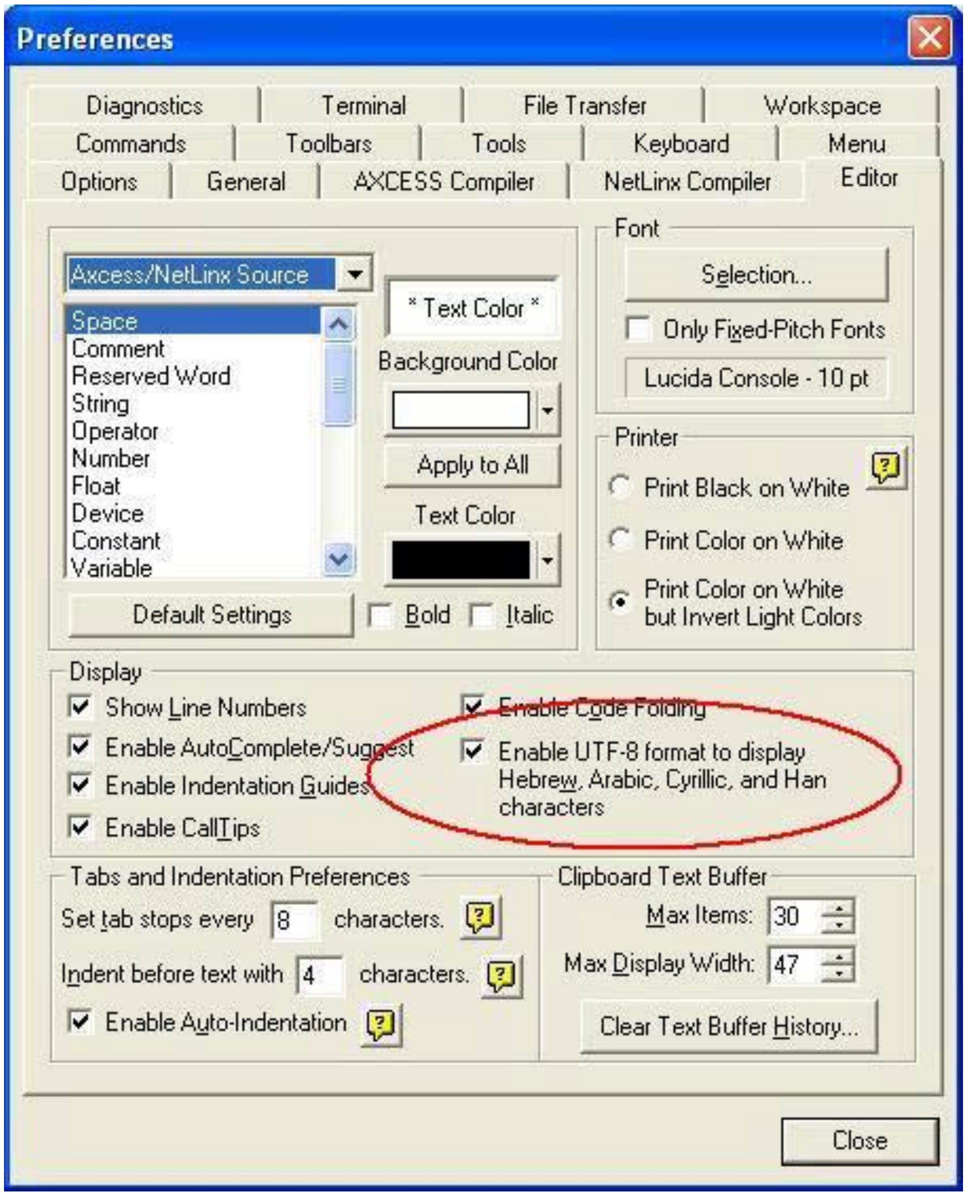
Before you begin to work with Unicode, you must change two settings in the NetLinx Studio Preferences:

1. In the Settings menu/Preferences/Editor tab/Display section/ you must check the "Enable UTF-8 Format to display Hebrew, Arabic, Cyrillic, and Han characters" checkbox. This will tell Studio to store your file as UTF-8, which will support Unicode characters.
2. In the Settings menu/Preferences/NetLinx Compiler tab/Options section/ you must check the "Enable _WC Preprocessor (Unicode) checkbox. This will tell Studio to process the _WC pre-processor statements to properly handle Unicode embedded in your source files at compile time.
3. Finally, you must include the UnicodeLib.axi in the module or source file where you want to use the Unicode functions.

We will step you through this, with screenshots, below.

To enable UTF-8, in the NetLinx Studio Editor:

- Choose Settings->Preferences from the menu bar
- Select the Editor tab
- Under Display, check the Enable UTF-8 format checkbox.
- Close the Preferences dialog.



Axcess/NetLinx Source

Space
Comment
Reserved Word
String
Operator
Number
Float
Device
Constant
Variable

* Text Color *

Background Color

Text Color

[ ] Bold [ ] Italic

Font
[ ] Only Fixed-Pitch Fonts
Lucida Console - 10 pt

Printer
( ) Print Black on White
( ) Print Color on White
(●) Print Color on White but Invert Light Colors

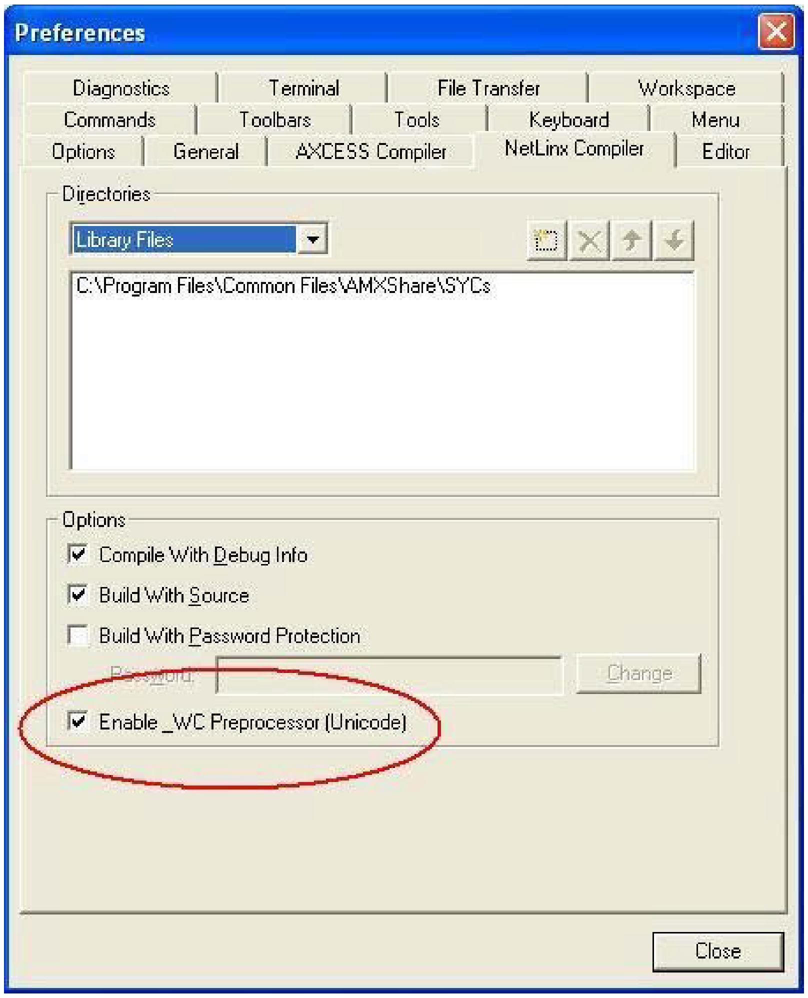
Display
[✓] Show Line Numbers
[✓] Enable AutoComplete/Suggest
[✓] Enable Indentation Guides
[✓] Enable CallTips
[✓] Enable Code Folding
[✓] Enable UTF-8 format to display Hebrew, Arabic, Cyrillic, and Han characters

Tabs and Indentation Preferences
Set tab stops every 8 characters.
Indent before text with 4 characters.
[✓] Enable Auto-Indentation

Clipboard Text Buffer
Max Items: 30
Max Display Width: 47

To enable Unicode Compiling in NetLinx Studio:

- Choose Settings->Preferences from the menu bar
- Select the NetLinx Compiler tab
- Under Options, check the Enable _WC Preprocessor checkbox.
- Close the Preferences dialog.



C:\Program Files\Common Files\AMXShare\SYCs

☑ Compile With Debug Info
☑ Build With Source
☐ Build With Password Protection
☑ Enable _WC Preprocessor (Unicode)

Including the Unicode Library

The Unicode Library is implemented in a NetLinx Include file, UnicodeLib.axi, that must be included in your program in order to access the Unicode functions. The Unicode Library is located in an Include file located in the C:\Program Files\Common Files\AMXShare\AXIs directory. Because this location is the default Include search path, you do not need to specify the directory in the include statement.

To include the Unicode Library to your program add these lines to your program:

```
(***********************************************************)
(* INCLUDE FILES GO BELOW *)
(***********************************************************)
#INCLUDE 'UnicodeLib.axi'
```

Defining a Unicode String Literal

To enter Unicode characters into your program, enclose the characters in single quotes, like you would any other string, and wrap the string literal in the Unicode macro _WC. Here is an example:

_WC('Your string goes here')

All Unicode string literals must be wrapped in the _WC macro. Failing to wrap a Unicode string in the _WC macro will result in a compiler error.

Storing a Unicode String

Unicode strings are stored in WIDECHAR arrays, similar to the way ASCII strings are stored in CHAR arrays. To define a WIDECHAR constant or variable and initialize it using a Unicode string literal, use the following syntax:

WIDECHAR wcMyString[] = _WC('My String')

Note: The "wc" prefix is Hungarian notation for widechar. This is simply a programming convention and is completely optional. Hungarian notation helps you better identify your variables while you are programming and is a general recommended standard. For more information, see https://en.wikipedia.org/wiki/Hungarian_notation

Working with WIDECHAR arrays and Unicode Strings

Working with WIDECHAR arrays and Unicode strings is very similar to working with CHAR arrays and ASCII strings. Most operation that can be performed on a CHAR array can be performed on a WIDECHAR array. For instance, to assign a string to a variable use this syntax:

wcMyString = _WC('My String')

The string functions defined for CHAR arrays have been defined for WIDECHAR array for use in Unicode programming. These functions allow you to operate on strings similar to the way you would with CHAR array. For instance, to remove the first 3 characters from a WIDECHAR array and return those characters as a WIDECHAR array, use WC_GET_BUFFER_STRING:

wcRemoved = WC_GET_BUFFER_STRING(wcMyString,3)

You will find that most other function work exactly as their CHAR counterpart do except they work on and return WIDECHAR arrays. The list of Unicode compatible functions is:

- WC_COMPARE_STRING
- WC_GET_BUFFER_CHAR
- WC_GET_BUFFER_STRING
- WC_LEFT_STRING
- WC_FIND_STRING
- WC_LENGTH_STRING
- WC_LOWER_STRING
- WC_MAX_LENGTH_STRING
- WC_MID_STRING
- WC_REMOVE_STRING
- WC_RIGHT_STRING
- WC_SET_LENGTH_STRING
- WC_UPPER_STRING

Character Case Mappings Converting between upper and lower case is accomplished by using the Unicode.org character database to determine the mapping between upper case and lower case characters. Not all Unicode characters have an upper or lower case equivalent; these characters will not be affected by WC_UPPER_STRING and WC_LOWER_STRING. Only the characters defined by Unicode.org as having an upper or lower case mapping are affected by these functions. For more information on Unicode character conversion, see the http://www.unicode.org/faq/conversion_mapping.html

Concatenating String Unicode strings and WIDECHAR array cannot be concatenated using the same syntax that ASCII strings use. In NetLinx, string expressions are enclosed in double quotes and can only contain 8-bit strings. To concatenate Unicode strings and WIDECHAR arrays, you must use the WC_CONCAT_STRING function:

wcMyString = WC_CONCAT_STRING(_WC('First name'),_WC(' SurName'))

If you attempt to concatenate Unicode strings or WIDECHAR arrays using NetLinx string expressions, expect data loss.

Converting between WIDECHAR and CHAR

On occasion, you may need to convert a CHAR array to a WIDECHAR array or a WIDECHAR array to a CHAR array. The CH_TO_WC and WC_TO_CH functions can be used to accomplish these conversions. For example:

wcMyString = CH_TO_WC('Any ASCII string')

wcMyString = CH_TO_WC(cMyString)

cMyString = WC_TO_CH(_WC('Any Unicode string'))

cMyString = WC_TO_CH (wcMyString)

When converting from WIDECHAR to CHAR, Unicode characters are converted to '?'. Any ASCII or extended ASCII characters, i.e. 8-bit characters, contained in the WIDECHAR array will appear in the CHAR array. Converting from CHAR to WIDECHAR never results in loss of data.

Using FORMAT

The NetLinx Unicode library does not include a Unicode compatible FORMAT function. In NetLinx, the format function is used to convert numbers to text. To use FORMAT with Unicode string, use FORMAT to convert the number to a CHAR array and then use CH_TO_WC and WC_CONCAT_STRING to combine the result with an existing WIDECHAR array. The following two syntaxes are functionality equivalent:

fTemperature = 98.652

cMyString = FORMAT('The current temperature is %3.2f',fTemperature)

fTemperature = 98.652

cTempString = FORMAT('%3.2f',fTemperature)

wcMyString = _WC('The current temperature is ')

wcMyString = WC_CONCAT_STRING(wcMyString,CH_TO_WC(cTempString))

Reading and Writing to Files

The NetLinx Unicode library supports reading and writing of WIDECHAR arrays. The WC_FILE routines operate the same as the FILE routines with the exception of FILE_OPEN. WC_FILE_OPEN takes an additional parameter; the file format.

The WC_FILE_OPEN returns a special file handle so it is important to only use the file handle returned by WC_FILE_OPEN with other WC_FILE functions and the file handle used with WC_FILE functions must have been obtained by calling WC_FILE_OPEN.

The NetLinx Unicode library supports three different file formats for compatibility with files created on a computer. Windows Notepad supports the same three file formats so files created in Notepad can be read using the WC_FILE routines and files created using the WC_FILE routines can be read with Notepad.

When reading or appending to file, the file format is automatically determined when the file is opened. You can pass in a variable to WC_FILE_OPEN and the function will set the variable to the file format that was detected. When writing files, the file format parameter will determine how data is written to the file. The following constants can be used for specifying or checking the file format: WC_FORMAT_UNICODE, WC_FORMAT_UNICODE_BE, WC_FORMAT_UTF8. The Unicode file format, specified by the constant WC_FORMAT_UNICODE , is the fastest to encode and decode. You should use this format unless you have a particular application that requires either UTF-8 or Unicode BE encoding.

The WC_FILE_READ/WRITE functions take the number of **characters** that will be read or written to the file. However, the functions return the number of **bytes** read or written to the file, not the number of characters. For Unicode and Unicode BE encoding, there are 2 bytes for every character. For UTF-8 encoding, the number of bytes for every character varies depending on the character.

-

Unicode filenames are not supported. The parameter for the file name is a CHAR array. Always use a non-Unicode name for the file.

The following file functions support WIDECHAR arrays:

- WC_FILE_OPEN
- WC_FILE_CLOSE
- WC_FILE_READ
- WC_FILE_READ_LINE
- WC_FILE_WRITE
- WC_FILE_WRITE_LINE

Send strings to a User Interface

Sending a WIDECHAR array to a user interface is accomplished using WC_TP_ENCODE. WC_TP_ENCODE takes a WIDECHAR array and returns a CHAR array formatted for a user interface ^UNI or ^BAU command.

cMyString = WC_TP_ENCODE(wcMyString)

SEND_COMMAND dvTP,"'^UNI-1,0,',cMyString "

Right-to-Left Unicode Strings

Right-to-Left Unicode languages are stored in memory the same way left-to-right language are. The first memory position of an array contains the first logical character. You can access the right-most character of a Right-to-Left Unicode string using this notation:

wchChar = wcString[1]

Right-to-left languages are not stored differently than left-to-right languages, they are simply rendered differently than right to left languages.

However, note that the functions WC_LEFT_STRING and WC_RIGHT_STRING remove a number of characters from the start and end of a string respectively. Using WC_LEFT_STRING on a right-to-left language will return the number of right-most, i.e. first, characters you requested, not the left-most, i.e. end, characters.

WC_LEFT_STRING returns the number of characters request from the front of the string and WC_RIGHT_STRING return the number of characters requested from the end of the string, regardless of the language's orientation.

Compiler Errors

The most common type of compiler errors you will encounter while programming for Unicode are caused by not wrapping Unicode string literals in _WC, passing a WIDECHAR to a function that take a CHAR array or passing a CHAR array to a function that takes a WIDECHAR array.

If you forget to wrap a Unicode string in _WC, expect to see the following compiler error:

On the line where the string is defined:

C10571: Converting type [string] to [WIDECHAR]

On the line where the constant or variable is used:

C10585: Dimension mismatch: [1] vs. [0] and C10533: Illegal assignment statement

If you try to pass a CHAR array to a function that expects a WIDECHAR array, expect to see the following compiler error:

On the line where the function call is made

C10585: Dimension mismatch: [1] vs. [0] and Type mismatch in call for parameter [WCDATA]

If you try to pass a WIDECHAR array to a function that expects a CHAR array, expect to see the following compiler:

On the line where the function call is made

C10585: Dimension mismatch: [1] vs. [0] and Type mismatch in call for parameter [A]

Note: Parameter names might not match those listed above.