Porting and Localizing LabVIEW VIs

This application note describes porting VIs among platforms in LabVIEW. It also describes localizing VIs in LabVIEW.

Portable and Nonportable VIs

VIs are portable among all platforms on which LabVIEW runs as long as the versions of LabVIEW are the same. VIs that contain Code Interface Nodes (CIN) or platform-specific features, such as ActiveX, are not portable. Porting in these cases breaks the VIs.

LabVIEW uses the same file format on all platforms.

When you open the VI on the new platform, LabVIEW detects the VI is from another platform and recompiles the VI to use the correct instructions for the current processor. If you transfer VIs on a disk formatted for a different platform, you may require a utility program to read the disk, such as Apple File Exchange on Macintosh.

You cannot port the following VIs:

- VIs distributed in the `vi.lib` directory – Each distribution of LabVIEW contains its own `vi.lib`. Do not move VIs in `vi.lib` across platforms.
- Compatibility VIs – LabVIEW uses compatibility VIs if you created VIs in earlier versions of LabVIEW and open them in a newer version of LabVIEW. Compatibility VIs emulate functionality in previous versions of LabVIEW. Porting breaks these VIs because they use CINs that are available only on one platform.
- VIs that contain CINs – LabVIEW displays an **object code not found** error when the CIN is from a different platform. If you write your CIN source code in a platform-independent manner, you can recompile it on another platform and relink it to the ported VI.
- VIs that contain the Call Library function – Porting breaks these VIs unless they can find a library of the same name as the Call Library Function references.
- Platform-specific communication VIs, such as ActiveX VIs on Windows and AppleEvents VIs on Mac OS.
- The LabVIEW for Windows In Port and Out Port Utility VIs and the LabVIEW for Macintosh Peek and Poke Utility VIs.

Porting Among Platforms

Portability issues include differences in filenames, separator characters, resolutions and fonts, possible overlapping of labels, and differences in picture formats.

Filename Differences

To ease porting among platforms, consider the filename. Windows and UNIX filenames can have 255 characters, including the `.vi` extension. Mac OS filenames can have 31 characters, including the `.vi` extension.

To avoid complications, save VIs with names valid for all platforms or save them in a VI library. A library name must conform to platform limits, but VIs in libraries can have names up to 255 characters, regardless of platform. Thus, VI libraries are the most convenient format for transferring VIs because libraries eliminate most file system dependencies.

Filenames are case sensitive on UNIX but not elsewhere. If you refer to a VI by name, capitalize the name consistently wherever you use it.

Separator Character Differences

If you develop a VI for use on other platforms, do not use platform-specific path separator characters \ , / , : in filenames. Avoid any special characters in filenames because different file systems interpret them differently. For example, hidden files in UNIX begin with a period.

Resolution and Font Differences

Fonts can vary from platform to platform. After you port a VI, select new fonts to obtain an appealing display. When you design VIs, keep in mind the three fonts that map best among platforms: the Application font, the System font, and the Dialog font.

The Application font is the LabVIEW default font used in the **Controls** palette, the **Functions** palette, and new controls.

- **(Windows)** LabVIEW uses the font that Windows uses for filenames in Explorer. For the U.S. version of Windows, this is usually MS Sans Serif or Tahoma. The size of the font depends on the settings of the video driver because you can set higher resolution video drivers to use Large Fonts or Small Fonts. We recommend Small Fonts and Windows standard settings.
- **(Mac OS)** LabVIEW uses the same font the Finder uses for filenames. For the U.S. Mac OS, LabVIEW uses Geneva. For the Japanese Mac OS, LabVIEW uses Osaka.
- **(UNIX)** LabVIEW uses Helvetica by default.

The System font is the font LabVIEW uses for menus.

- **(Windows)** This font is the same as the Application font, but the font size may differ.
- **(Mac OS)** For the U.S. Mac OS, LabVIEW uses Chicago. For the Japanese Mac OS, LabVIEW uses Osaka.
- **(UNIX)** LabVIEW uses Helvetica by default.

The Dialog font is the font LabVIEW uses for text in dialog boxes.

- **(Windows)** This font is the same as the System font.
- **(Mac OS)** This font is the same as the System font.
- **(UNIX)** LabVIEW uses Helvetica by default.

When you port a VI that contains one of these fonts to another platform, LabVIEW ensures the font maps to something similar on that platform.

If you do not use the defined fonts, the font can change size on the new platform because of the differences in the fonts available and differences in the resolution of the display on the other platform. For example, if you select Geneva or New York on the Macintosh and port the VI to UNIX, LabVIEW cannot find that font name on Windows and substitutes it with Arial. The substituted font may not match the original, and objects may overlap. If you port a VI to Windows and Windows does not recognize the fonts, the new font may display differently.

If you use a defined font on a section of text, avoid changing the size of that text. If you change the font to something other than the default size and then port the VI to a different platform, LabVIEW tries to match the font with the new size, which might be inappropriate depending on the resolution of the screen.

Character Code Issues

A computer represents data as bytes. It stores letters and other characters by assigning each a character code.

In the simplest case, one byte corresponds to one character according to a mapping table, or encoding. U.S. computer systems use single-byte encoding, which represents up to 256 characters. Other systems, such as Japanese or Chinese systems, use multi-byte encoding to represent several thousand characters.

Among the various encoding, the most common are ASCII and ISO Latin I, which is an extension of ASCII.

ASCII encoding is a 7-bit code (from 0 to 127). It contains upper and lowercase English, American English punctuation, base 10 numbers, and a few control codes. Control codes are a set of nondisplayable characters used for text formatting. You can print most of the ASCII characters in the \0x21 to \0x7F range.

To represent international characters, single-byte operating systems use an 8-bit extension of ASCII. The lower 128 character codes are identical to ASCII, and the upper 128 are different for each code page. The various international characters reside in the upper 128 character codes. Windows, UNIX, and Mac OS use 8-bit character sets, but they each map differently, which causes portability problems.

For the United States and Western European countries, Windows and UNIX use the same code page. A code page is a list of selected character codes in a certain order. Mac OS uses a different code page than Windows and UNIX. For Japanese operating systems, all platforms use the same code page, so portability is less of an issue for a Japanese application.

When you port a VI from one platform to another, such as from Macintosh to PC or from PC to Macintosh, you might encounter problems because of the differences between platform code pages, especially if you use non-ASCII characters in the upper 128 range. For example, the word café in French uses the extended character é, whose hexadecimal value is \0xE9 on the PC. This same hexadecimal value represents the character È on the Macintosh.

LabVIEW performs a code page conversion for elements such as captions, VI and parameter tips and descriptions, VI titles, and other private data. Private data is listbox item names, table row and column headers, font used for table cells, and graph plot names and cursor names. However, LabVIEW does not map labels or string content. Labels and string content might not appear as you expect after you port them to a new platform, and you may need to modify them manually. Therefore, if you plan to port your VIs from Windows or UNIX to Mac OS and vice versa, avoid using international characters in labels or string content. However, you can port a Japanese application without restriction from Windows or UNIX to Macintosh because these platforms use the same code page for Japanese.

For example, LabVIEW does not map string content because strings can be interpreted as binary information. If you communicate with an instrument using GPIB or serial communication and if your instrument expects the binary string \0x63\0x61\0x66\0xE9, this string appears as café on the Windows platform, and it appears as cafÈ on the Macintosh platform. In both cases, you should send the same binary string to the instrument. To avoid this communication error, LabVIEW does not perform character mapping on string values.

LabVIEW does not map labels because it can break VIs, such as using the Unbundle by Name function. The VI is broken because when you change the label, the reference to the control becomes invalid.

Overlapping Labels

When you move a VI to a new platform, controls and labels might change size, depending on whether the fonts are smaller or larger on the new platform. LabVIEW tries to keep labels from overlapping their owners by moving them away from the owning control. Also, every label and constant has a default **Size to Text** attribute. When you first create a label or constant, this attribute is set so the bounds of the object resize as necessary to display all the enclosed text.

If you manually resize the label, LabVIEW turns off this attribute, and the item in the shortcut menu no longer shows a checkmark. With **Size to Text** turned off, the bounds of the object stay constant, and LabVIEW clips or crops the enclosed text as necessary. If you do not want LabVIEW to clip text when you move among platforms, leave this attribute on for labels and constants. After you resize the label, you can manually turn on **Size to Text**.

Not all computers have high-resolution monitors. However, if you have a monitor with a high resolution, do not make your front panels very large if you want them to port well. For best portability results, leave extra space between controls and avoid overlapping controls. When you port a label that overlaps another label, it might overlap the whole control when you enlarge the font on the new platform.

Picture Differences

The most basic type of picture contains a bitmap, which is a series of values that specify the color of each pixel in the picture. More complex pictures can contain commands the computer system executes every time it displays the picture.

Use the paint layer of a graphics application to create bitmap-based pictures. Bitmaps are common storage formats for pictures on all platforms. If you use pictures that contain bitmaps on front panels, the pictures usually look the same when you load the VIs on another platform. However, pictures that contain drawing commands such as filling might include commands that other platforms do not support. Check how VIs look on another platform if you expect to use them there.

Use the draw layer of a graphics application to create pictures that contain drawing commands. To make the pictures more portable, paste the final picture into a paint layer of a graphics application before you import the picture into the VI.

(Windows and Mac OS) With some graphic applications on Windows and many graphics applications on Mac OS, you can cut or copy a picture with a nonrectangular shape. For example, in Mac OS, you can use a Lasso tool to select the outline of a circle, triangle, or a more complicated shape such as a musical note.

Other platforms might draw these irregular shapes on a rectangular white background. In Windows, if you want pictures with irregular shapes and pictures that scale well, use applications that support enhanced metafiles.

Porting across LabVIEW Applications

You can convert any LabVIEW VI that still contains its block diagram to BridgeVIEW, provided the version of LabVIEW and BridgeVIEW are compatible. Converting VIs from BridgeVIEW to LabVIEW is also possible, but not all VIs convert.

LabVIEW to LabVIEW

Localized versions of LabVIEW use the same executable as the U.S. version. Therefore, localized versions are compatible with the U.S. version. You can load VIs saved in LabVIEW Datalogging and Supervisory Control (DSC) module only if you select **Save for Previous** to save those VIs for LabVIEW 5.0.

Table 1 shows compatibility between LabVIEW versions. The check marks indicate that the 5.0.x, 5.1.x, and 6.0.x version headings can load that specific version of LabVIEW. The X indicates that the 5.0.x and 5.1.x version headings cannot load that specific version of LabVIEW unless you select **Save for Previous**. The ☐ indicates that you must buy a conversion kit to convert the VI to LabVIEW 4.0, then open the VI in LabVIEW 6.0.

Table 1. LabVIEW Compatibility

Saved in LabVIEW Version	Loaded in LabVIEW Version 5.0.x	Loaded in LabVIEW Version 5.1.x	Loaded in LabVIEW Version 6.0.x
3.x	✔	✔	☐
4.x	✔	✔	✔
5.0.x	✔	✔	✔
5.1.x	✘	✔	✔
6.0.x	✘	✘	✔

LabVIEW to BridgeVIEW

The most important factor to notice when you convert VIs from LabVIEW to BridgeVIEW is the version of LabVIEW you used to create the VI. You can load all LabVIEW VIs saved with LabVIEW 3.*x* through LabVIEW 4.*x* into any version of BridgeVIEW, provided the VIs still have block diagrams. You can load the VIs you saved with LabVIEW 5.1.*x* only in BridgeVIEW 2.1 or later. You can load the VIs you saved with LabVIEW 5.0.*x* only in BridgeVIEW 2.0 or later. If you save VIs in LabVIEW DSC module, you cannot load them in any version of BridgeVIEW. You can load VIs saved in LabVIEW DSC module only if you select **Save for Previous** to save those VIs for LabVIEW 5.0.

Table 2 shows compatibility between LabVIEW and BridgeVIEW versions.

Table 2. LabVIEW and BridgeVIEW Compatibility

LabVIEW Version	BridgeVIEW Version
4.*x* or earlier	All versions
5.0	2.0
5.1	2.1
DSC module	Unable to load VIs from LabVIEW DSC module

BridgeVIEW to LabVIEW

Two factors determine whether LabVIEW can load and run a BridgeVIEW VI. First, the versions of LabVIEW and BridgeVIEW must be compatible. For example, you cannot load VIs you created with BridgeVIEW 1.0 in LabVIEW 4.1.

The second factor that determines if you can convert a VI from BridgeVIEW to LabVIEW is whether the VI contains VIs specific to BridgeVIEW. Any BridgeVIEW VI that uses features specific to BridgeVIEW may not load or operate correctly in LabVIEW. You can load VIs you saved with any version of BridgeVIEW in LabVIEW DSC module. However, you can load VIs you saved with BridgeVIEW 2.0 or earlier only in LabVIEW 5.0 or later. You can load VIs you saved with BridgeVIEW 2.1 only in LabVIEW 5.1 or later.

If you plan to use BridgeVIEW to develop VIs you intend to convert to LabVIEW, select the LabVIEW palette set in BridgeVIEW. This palette set displays the same functions as the LabVIEW palette set. VIs you build using only the functions in these palettes are portable between BridgeVIEW and LabVIEW.

Localizing LabVIEW VIs

You can localize a VI by exporting its strings in a tagged file, translating the file, and importing it back into the VI.

The tagged text file, also known as a VI string file, contains information to localize the VI title, description, control captions, control descriptions, and other private data of controls. Because you cannot localize control labels, a control also has a caption. By exporting the VI strings into a text file, LabVIEW creates a caption for each control and substitutes the label with a caption in the front panel.

Besides translating strings on the front panel, you can use localized decimal separators when converting numbers to strings. Use the Format Date/Time String function to specify how to display the date and time.

You can localize the following VI strings:

- VI window titles and descriptions
- Object caption and descriptions
- Free labels
- Default data (string, table, path, and array default data)
- Private data (listbox item names, table row and column headers, graph plot names, and graph cursor names)
- Block diagram strings (free labels and string constants), if you specify it

Note Localizing block diagram string constants is risky because it can break VIs.

You can localize the strings on the front panel and in the block diagram by exporting those strings to a tagged text file, which is the export and import VI string functionality in LabVIEW. You can localize the text file, and import that file back into the VI.

Complete the following steps to export and import the strings you localize on the front panel:

1. Select **Tools»Advanced»Export Strings** to export the VI strings.
2. Save the VI when the **Name a VI string file to write** dialog box appears. LabVIEW saves the VI as a text file in the same directory as your VI. The default name is the same as the VI name.
3. Click **Yes** in the message box that appears to localize control names.
4. Click **Yes** in the message box that appears to localize the diagram strings.
5. Open the text file to localize the VI strings and save your changes.
6. Select **Tools»Advanced»Import Strings** to import the VI strings.
7. **Select a VI string file to read** dialog box appears, prompting you to open the text file.
8. LabVIEW creates a log file that contains a list of any errors that occurred during the operation and prompts you to save.

LabVIEW does not import a VI string file if the file contains unknown or missing tags. If you incorrectly localize the VI string, it may contain unknown or missing tags. If you modified the VI between the time you export the strings and the time you import your translation, the VI string file may contain unknown or missing tags.

Syntax of the VI String File

The format of the VI string file is much like an HTML file. The system marks every element with a start tag and an end tag. A start tag begins with < and ends with >. An end tag starts with </ and ends with >. LabVIEW ignores whitespace characters except within text. Because the character < indicates the beginning of a tag, LabVIEW uses << for the less than character in text. For the > character, LabVIEW uses >> for the greater than character. LabVIEW replaces the double quotation mark with "". Also, LabVIEW denotes end-of-line characters as <CR>, <CRLF>, or <LF>, which are treated as a carriage return, a carriage return followed by a line feed, and a line feed, respectively. The format of the VI string file is intended to be machine readable. Do not be concerned if you find it difficult to read. If you change or delete tags, LabVIEW issues errors when it imports the file into the VI.

Table 3 lists the VI tag types and the corresponding VI tag syntax.

Table 3. VI Tag Types and Syntax

VI Tag Type	VI Tag Syntax
[VI string file].	<VI [vi attributes] > [vi info] </VI>
[VI attributes].	syntaxVersion=5 LVversion=nnn revision=nnn name="text"
[VI info].	[vi title] [description] [content] [bdcontent]
[VI title].	<TITLE>text</TITLE> | .<TITLE><NO_TITLE></TITLE>.
[description].	<DESC>text</DESC>.
[content].	<CONTENT>[objects]</CONTENT>.
[bd content].	<BDCONTENT>[bd objects] </BDCONTENT>

A space separates VI attributes. No space is permitted between the attribute name and the following equal sign and between the equal sign and the attribute value.

For example:

```
<VI syntaxVersion=5 LVversion=4502007 revision=10 name="AO Generate Waveform.vi">
<TITLE>AO Generate Waveform.vi</TITLE>
<DESC>This VI generates a timed, simple-buffered waveform for the given output channel
at the specified update rate.</DESC>
<CONTENT>
.....
</CONTENT>
</VI>
```

Table 4 lists the tags that describe the content of the front panel (free and object-owned labels, caption labels, attributes) and the corresponding tag syntax.

Table 4. Tags for Front Panel Content

Content Tag Type	Content Tag Syntax
[content]	<CONTENT>[objects]</CONTENT>
[objects]	([control] | [label]) *
[control]	<CONTROL [control attributes]> [control info] </CONTROL>
[label]	<LABEL>[style text] </LABEL>
[style text]	<STEXT>text with font information </STEXT>

Between `<STEXT>` and `</STEXT>`, you can type font specifications. LabVIEW encodes font information using the following format: `<FONT name="font name" size='3' style='BIUSO' color=00ff00>`. You can list the font attributes in any order. Font specification is different from other elements because it has no end tag. For example, you can describe a caption with the text "**Bold** *label*" as follows:

```
<LABEL><STEXT><FONT name="times new roman" size=12 style='B'>Bold <FONT
style='I'>label</STEXT></LABEL>
```

You can define fonts as `predef` to specify one of the predefined fonts – Application font, System font, or Dialog font.

Table 5 lists the tags that describe the objects that contain data and the corresponding tag syntax.

Table 5. Tags for Objects Containing Data

Content Tag Type	Content Tag Syntax
[control]	<CONTROL [control attributes]> [control info] </CONTROL>
[control attributes]	ID=xxx type="Boolean" name="switch"
[control info]	[description] [tip strip] [parts] [privData section] [defData section] [content]
[tip strip].	<TIP>text</TIP>
[parts]	<PARTS> [part]*</PARTS>
[part]	<PART [part attributes]> [part info] </PART>
[part attributes]	partID=nnn partOrder=nnn
[part info]	[control] | [label] | [multiLabel]

The following is an example of a ring control description with a caption of "Ring" and options **Load**, **Unload**, **Open**, and **Close**.

```
<CONTROL ID=87 type="Ring" name="RING control">
<DESC>ring control</DESC>
<PARTS>
<PART ID=12 order=0
type="Ring Text"><MLABEL><STRINGS><STRING>Load</STRING><STRING>Unload</STRING><STRING>
Open</STRING><STRING>Close</STRING></STRINGS></MLABEL></PART>
<PART ID=82 order=0 type="Caption"><LABEL><STEXT><FONT color=FF0033
size=12>Ring</STEXT></LABEL></PART>
</PARTS>
</CONTROL>
```

LabVIEW uses the MLABEL (multilabel) tag above to designate the option string on a ring control or the strings on Boolean buttons, a string for each of the four states. The following is a generic description of the MLABEL tag syntax:

```
[multiLabel]<MLABEL> [mlabel info] </MLABEL>
[mlabel info][font][strings]
```

Table 6 lists the tags that describe the default data for strings, tables, arrays, and paths and the corresponding tag syntax.

Table 6. Tags for Default Data of Strings, Tables, Arrays, and Paths

Content Tag Type	Content Tag Syntax
[defData section]	<DEFAULT> [defData] </DEFAULT>
[defData]	[str def] | [table def] | [arr data] | [path data]
[str def]	[string] | <SAME_AS_TEXT>
[table def]	[strings]
[arr data]	<ARRAY nElems=n> [arr element data] </ARRAY>
[arr element data]	[clust data] | [str data] | [non-str data]
[str data]	[string]
[non-str data]	<NON_STRING>
[clust data]	<CLUSTER nElems=n> [clust element data] </CLUSTER>
[clust element data]	[clust data] | [str data] | [non-str data] | [arr data] | [path data]
[path data]	<PATH type =”absolute”> a<SEP> system </PATH>

For [arr data], n [arr element data] must follow the <ARRAY> tag. Likewise, for [clust data], n [clust element data] must follow the <CLUSTER> tag.

For the string control default data [str def], use a special tag, <SAME_AS_LABEL>, which indicates the string default data is the same as the text label on the string parts list. When you use <SAME_AS_LABEL>, it is unnecessary to type the same text for both the text label and the string default data.

For path control default data [path data], the <PATH> start tag can have an attribute that specifies the path type. The possible attribute values are “absolute”, “relative”, “not-a-path”, and “unc”. A <SEP> tag separates the path segments that come between the <PATH> and </PATH> tags. For example, on the Windows platform, an absolute path c:\windows\temp\temp.txt is written as follows:

```
<PATH type="absolute">c<SEP>windows<SEP>temp<SEP>temp.txt</PATH>
```

Tables 7 and 8 list the tags that describe private data and the corresponding tag syntax.

Table 7. Tags for Private Data and Syntax

Content Tag Type	Content Tag Syntax
[privData section]	<PRIV> [privData] </PRIV>
[privData]	([items] [col header] | [row header] | [cell fonts] | [plots] | [cursors] | [path privData] | [tab control privData])
[items]	<ITEMS> [string]* </ITEMS>
[col header]	<COL_HEADER> [string]* </COL_HEADER>
[row header]	<ROW_HEADER> [string]* </ROW_HEADER>
[cell fonts]	<CELL_FONTS> [cell font]* </CELL_FONTS>
[plots]	<PLOTS> [string]* </PLOTS>
[cursors]	<CURSORS> [string]* </CURSORS>
[cell font]	[row# col#][font]
[font]	<FONT name="font name" size='x' style='BIUSO' color=000000>
[path privData]	<PROMPT>text</PROMPT> <MTCH_PTN>TEXT</MTCH_PTN> <STRT_PTH>[path data]</STRT_PTH>
[tab control privData]	<PAGE_CAPTIONS>[string]*</PAGE_CAPTIONS>
[tab control page]	<PAGE> [description] [tip strip] [objects] </PAGE>

`[strings]` and `[string]` have the following format:

```
[strings]<STRINGS> [string]* </STRINGS>
[string]<STRING> text </STRING>
```

Table 8. Content Tag and Syntax

Content Tag Type	Content Tag Syntax
[bd content]	<BDCONTENT> [bdobjects]</BDCONTENT>
[bd objects]	([control] | [label] | [node])*
[node]	<NODE [node attributes]>[node info]</NODE>
[node attributes]	ID=xxx type="Sequence"
[node info]	[description] [bd content]

Editing VI Window Titles

You can customize the VI window title to make it more descriptive than the VI filename, which is important for localized VIs. You can localize the VI window title, which is not governed by file-system naming constraints and is still recognized by the VIs that might call it because the calling VI references the subVI by its filename, not its title. Use the export/import tools to localize the window title, or use the **Window Appearance** page of the **VI Properties** dialog box to manually change the VI window title while you edit a VI. You can also use VI Server to change the VI window title programmatically.

Period and Comma Decimal Separators

By default, LabVIEW adapts to use the appropriate decimal separator on your operating system preferences, which affects the display of numbers within controls and the operation of functions that convert between numbers and strings. This causes problems when passing numbers as strings to instruments or parsing instrument output strings.

The following functions use the operating system settings by default. To use a period, you must wire a False value to the Boolean input called **Use System Decimal Point (+)**.

- Fract/Exp String To Number
- Number To Fractional String
- Number to Exponential String
- Number To Engineering String

For the functions that take a format string, the %x; format code, where x represents the separator character, can be used to specify the decimal separator for that specific call. These functions include:

- Format Into String
- Scan From String
- Format Value
- Scan Value
- Array To Spreadsheet String
- Spreadsheet String To Array

Select **Tools»Options** and select **Use localize decimal point** in the **Front Panel** pull-down menu to specify whether to use the system settings (locale) for the decimal separator or a period.

Format Date/Time String

You can set the display format of date and time for strings by using the Format Date/Time String Function by default. LabVIEW reads the operating system settings and displays the date/time representation appropriate for the current locale. You also can customize the display by selecting **Format & Precision** on the shortcut menu.

Time and Date Format for Numerics, Data Types, and Graphs

If you want to display numerics with a time and date format, you can select a custom-fixed format or use a system (locale-specific) format by selecting **Format & Precision** on the shortcut menu of numerics, and waveform data types or selecting **Formatting** on the shortcut menu for graphs. By default, waveform data types use the system format.