

LabVIEW User Manual

LABVIEW MANAGER DATA TYPES

Table of Contents

LabVIEW Manager Data Types

...

Updated 2025-08-15 | ⏳ 4 minute(s) read # LabVIEW # User Manual

LabVIEW Manager data types include the following data types:

- Scalar
- Char
- Dynamic
- Memory-related
- Constants

Scalar

Scalar data types include Boolean and numeric.

Boolean

External code modules work with two kinds of Boolean scalars—those existing in LabVIEW block diagrams and those passing to and from manager routines. The manager routines use a conventional Boolean form where 0 is FALSE and 1 is TRUE. This conventional Boolean form is called a Bool32 and is stored as a 32-bit value.

LabVIEW block diagrams store Boolean scalars as 8-bit values. The value is 0 if FALSE and 1 if TRUE. This Boolean form is called an LVBoolean.

The following table describes the two forms of Boolean scalars.

Name	Description
Bool32	32-bit integer, 0 if FALSE, 1 if TRUE
LVBoolean	8-bit integer, 0 if FALSE, 1 if TRUE

Numeric

The manager data types support 8-, 16-, 32-, and 64-bit signed and unsigned integers.

LabVIEW supports the following single, double, and extended floating-point data types.

Type	Description
Single	32-bit
Double	64-bit
Extended	Up to 80-bit

LabVIEW supports complex numbers containing two floating-point numbers, with different complex numeric types for each of the floating-point data types.

LabVIEW supports the following basic data types for numbers:

- Signed integers
 - int8 8-bit integer
 - int16 16-bit integer
 - int32 32-bit integer
 - int64 64-bit integer
- Unsigned integers
 - UInt8 8-bit integer
 - UInt16 16-bit integer

- `uInt32` 32-bit integer
- `uInt64` 64-bit integer
- `size_t` 32-bit integer
- Floating-point numbers
 - `float32` 32-bit floating-point number
 - `float64` 64-bit floating-point number
 - `floatExt` extended-precision floating-point number

The following table explains how various platforms store extended-precision numbers.

Platform	Storage Format
Windows and Linux	80-bit structure with two <code>int32</code> components, <code>mhi</code> and <code>ml0</code> , and an <code>int16</code> component, <code>e</code>
Mac Intel	64-bit double-precision floating-point number

Complex Numbers

The complex data types are structures with two floating-point components, `re` and `im`. As with floating-point numbers, complex numbers can have 32-bit, 64-bit, and extended-precision components. The following table contains the code for the type definitions for each of these complex data types.

Complex Number Type	Code
32-bit	<pre>typedef struct { float32 re, im; } cmplx64;</pre>
64-bit	<pre>typedef struct { float64 re, im; } cmplx128;</pre>
Extended-precision	<pre>typedef struct { floatExtre, im; } cmplxExt;</pre>

char

The `char` data type is defined by C to be an 8-bit signed integer. LabVIEW defines an unsigned `char` data type, with the following type definition:

```
typedef uInt8 uchar;
```

Dynamic

LabVIEW defines a number of data types you must allocate and deallocate dynamically. Arrays, strings, and paths have data types you must allocate using [memory manager](#) and [file manager](#) routines.

Arrays

LabVIEW supports arrays of any of the basic data types described in this section. You can construct more complicated data types using clusters, which can in turn contain scalars, arrays, and other clusters.

The first four bytes of a LabVIEW array indicate the number of elements in the array. The elements of the array follow the length field.

Strings

LabVIEW supports C- and Pascal-style strings, lists of strings, and `LStr`, a special string data type you use for string parameters to external code modules. The [support manager](#) contains routines for manipulating strings and converting them among the different types of strings.

C-Style Strings (CStr)

A C-style string `CStr` is a series of zero or more unsigned characters, terminated by a zero. C strings have no effective length limit. Most manager routines use C strings, unless you specify otherwise. The following code is the type definition for a C string:

```
typedef uchar *CStr;
```

Pascal-Style Strings (PStr)

A Pascal-style string PStr is a series of unsigned characters. The value of the first character indicates the length of the string. A PStr can have a range of 0 to 255 characters. The following code is the type definition for a Pascal string:

typedef uchar	Str255[256], Str31[32],
*StringPtr,	
**StringHandle;	
typedef uchar	*PStr;

LabVIEW Strings (LStr)

The first four bytes of a LabVIEW string LStr indicate the length of the string. The specified number of characters follow the length of the string. LStr is the string data type used by LabVIEW block diagrams. The following code is the type definition for an LStr string:

```
typedef struct {
    int32 cnt;
    /* number of bytes that follow */
    uchar str[1];
    /* cnt bytes */
} LStr, *LStrPtr, **LStrHandle;
```

Concatenated Pascal String (CPStr)

Many algorithms require manipulation of lists of strings. Arrays of strings are usually the most convenient representation for lists. However, arrays of strings can place a burden on the memory manager because of the large number of dynamic objects it must manage. To make working with lists more efficient, LabVIEW supports the concatenated Pascal string CPStr data type, which is a list of Pascal-style strings concatenated into a single block of memory. Using the CPStr data structure, you can use support manager routines to create and manipulate lists.

The following code is the type definition for a CPStr string:

```
typedef struct {
    int32 cnt;
    /* number of pascal strings that follow */
    uchar str[1];
    /* cnt concatenated pascal strings */
} CPStr, *CPStrPtr, **CPStrHandle;
```

Paths

A path (pathname) indicates the location of a file or directory in a file system. LabVIEW has a separate data type for a path, represented as Path, which the file manager defines in a platform-independent manner. The actual data type for a path is private to the file manager and subject to change. You can create and manipulate Path data types using file.

Memory-Related

LabVIEW uses pointers and handles to reference dynamically allocated memory. The memory-related data types have the following type definitions:

```
typedef uchar *UPtr;
typedef uchar **UHandle;
```

Constants

The manager data types define the following constant for use with external code modules:

```
NULL 0(uint32)
```

The following constants define the possible values of the Bool32 data type:

FALSE 0 (int32)

TRUE 1 (int32)

The following constants define the possible values of the LVBoolean data type:

LVFALSE 0 (uInt8)

LVTRUE 1 (uInt8)

Parent topic: [LabVIEW Shared Libraries](#)

Previous

[← Using LabVIEW Manager Functions in Shared Libraries](#)

Next

[LabVIEW Manager Function Errors →](#)

