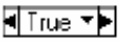
Case and Sequence Structures

Case, Stacked Sequence, Flat Sequence, and Event structures contain multiple subdiagrams. A Case structure executes one subdiagram depending on the input value passed to the structure. A Stacked Sequence and a Flat Sequence structure execute all their subdiagrams in sequential order. An Event structure executes its subdiagrams depending on how the user interacts with the VI.

Case Structures

A Case structure, shown at left, has two or more subdiagrams, or cases. Only one subdiagram is visible at a time, and the structure executes only one case at a time. An input value determines which subdiagram executes. The Case structure is similar to case statements or `if...then...else` statements in text-based programming languages.



The case selector label at the top of the Case structure, shown at left, contains the name of the selector value that corresponds to the case in the center and decrement and increment arrows on each side. Click the decrement and increment arrows to scroll through the available cases. You also can click the down arrow next to the case name and select a case from the pull-down menu.

Wire an input value, or selector, to the selector terminal, shown at left, to determine which case executes. You must wire an integer, Boolean value, string, or enumerated type value to the selector terminal. You can position the selector terminal anywhere on the left border of the Case structure. If the data type of the selector terminal is Boolean, the structure has a `TRUE` case and a `FALSE` case. If the selector terminal is an integer, string, or enumerated type value, the structure can have any number of cases.

Specify a default case for the Case structure to handle out-of-range values. Otherwise, you must explicitly list every possible input value. For example, if the selector is an integer and you specify cases for `1`, `2`, and `3`, you must specify a default case to execute if the input value is `4` or any other valid integer value.

Case Selector Values and Data Types

You can enter a single value or lists and ranges of values in the case selector label. For lists, use commas to separate values. For numeric ranges, specify a range as `10..20`, meaning all numbers from 10 to 20 inclusively. You also can use open-ended ranges. For example, `..100` represents all

numbers less than or equal to 100, and `100..` represents all numbers greater than or equal to 100. You also can combine lists and ranges, for example `..5, 6, 7..10, 12, 13, 14`. When you enter values that contain overlapping ranges in the same case selector label, the Case structure redisplays the label in a more compact form. The previous example redisplays as **`..10, 12..14`**. For string ranges, a range of `a..c` includes all of `a` and `b`, but not `c`. A range of `a..c,c` includes the ending value of `c`.

When you enter string and enumerated values in a case selector label, the values display in quotation marks, for example `"red"`, `"green"`, and `"blue"`. However, you do not need to type the quotation marks when you enter the values unless the string or enumerated value contains a comma or range symbol (`","` or `".."`). In a string value, use special backslash codes for non-alphanumeric characters, such as `\r` for a carriage return, `\n` for a line feed, and `\t` for a tab. Refer to the *LabVIEW Help* for a list of these backslash codes.

If you change the data type of the wire connected to the selector terminal of a Case structure, the Case structure automatically converts the case selector values to the new data type when possible. If you convert a numeric value, for example 19, to a string, the string value is `"19"`. If you convert a string to a numeric value, LabVIEW converts only those string values that represent a number. The other values remain strings. If you convert a number to a Boolean value, LabVIEW converts `0` to FALSE and `1` to TRUE, and all other numeric values become strings.

If you enter a selector value that is not the same type as the object wired to the selector terminal, the value appears red to indicate that you must delete or edit the value before the structure can execute, and the VI will not run. Also, because of the possible round-off error inherent in floating-point arithmetic, you cannot use floating-point numerics as case selector values. If you wire a floating-point value to the case, LabVIEW rounds the value to the nearest even integer. If you type a floating-point value in the case selector label, the value appears red to indicate that you must delete or edit the value before the structure can execute.

Input and Output Tunnels

You can create multiple input and output tunnels for a Case structure. Inputs are available to all cases, but cases do not need to use each input. However, you must define each output tunnel for each case. When you create an output tunnel in one case, tunnels appear at the same position on the border in all the other cases. If at least one output tunnel is not wired, all output tunnels on the structure appear as white squares. You can define

a different data source for the same output tunnel in each case, but the data types must be compatible for each case. You also can right-click the output tunnel and select **Use Default If Unwired** from the shortcut menu to use the default value for the tunnel data type for all unwired tunnels.

Using Case Structures for Error Handling

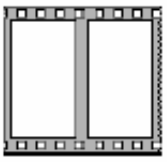
When you wire an error cluster to the selector terminal of a Case structure, the case selector label displays two cases, `Error` and `No Error`, and the border of the Case structure changes color—red for `Error` and green for `No Error`. The Case structure executes the appropriate case subdiagram based on the error state. Refer to the *Error Handling* section of Chapter 6, *Running and Debugging VIs*, for more information about handling errors.

Sequence Structures

A sequence structure contains one or more subdiagrams, or frames, that execute in sequential order. Sequence structures are not used commonly in LabVIEW. Refer the *Using Sequence Structures* and the *Avoiding Overusing Sequence Structures* sections of this chapter for more information on when to use sequence structures.

There are two types of sequence structures, the Flat Sequence structure and the Stacked Sequence structure.

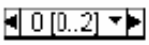
Flat Sequence Structure



The Flat Sequence structure, shown at left, displays all the frames at once and executes the frames from left to right until the last frame executes. Use the Flat Sequence structure to avoid using sequence locals and to better document the block diagram. When you add or delete frames in a Flat Sequence structure, the structure resizes automatically. To rearrange the frames in a Flat Sequence structure, cut and paste from one frame to another.

Stacked Sequence Structure

The Stacked Sequence structure, shown at left, stacks each frame so you see only one frame at a time and executes frame 0, then frame 1, and so on until the last frame executes. The Stacked Sequence structure returns data only after the last frame executes. Use the Stacked Sequence structure if you want to conserve space on the block diagram.



The sequence selector identifier, shown at left, at the top of the Stacked Sequence structure contains the current frame number and range of frames.

Use the sequence selector identifier to navigate through the available frames and rearrange frames. The frame label in a Stacked Sequence structure is similar to the case selector label of the Case structure. The frame label contains the frame number in the center and decrement and increment arrows on each side. Click the decrement and increment arrows to scroll through the available frames. You also can click the down arrow next to the frame number and select a frame from the pull-down menu. Right-click the border of a frame, select **Make This Frame**, and select a frame number from the shortcut menu to rearrange the order of a Stacked Sequence structure.

Unlike the case selector label, you cannot enter values in the frame label. When you add, delete, or rearrange frames in a Stacked Sequence structure, LabVIEW automatically adjusts the numbers in the frame labels.

Using Sequence Structures

Use the sequence structures to control the execution order when natural data dependency does not exist. A node that receives data from another node depends on the other node for data and always executes after the other node completes execution.

Within each frame of a sequence structure, as in the rest of the block diagram, data dependency determines the execution order of nodes. Refer to the *Data Dependency and Artificial Data Dependency* section of Chapter 5, *Building the Block Diagram*, for more information about data dependency.

The tunnels of Stacked Sequence structures can have only one data source, unlike Case structures. The output can emit from any frame, but data leave the Stacked Sequence structure only when all frames complete execution, not when the individual frames complete execution. As with Case structures, data at input tunnels are available to all frames.

To pass data from one frame to any subsequent frame of a Stacked Sequence structure, use a sequence local terminal, shown at left. An outward-pointing arrow appears in the sequence local terminal of the frame that contains the data source. The terminal in subsequent frames contains an inward-pointing arrow, indicating that the terminal is a data source for that frame. You cannot use the sequence local terminal in frames that precede the first frame where you wired the sequence local.

Refer to the `examples\general\structs.llb` for examples of using sequence structures.

Avoiding Overusing Sequence Structures

To take advantage of the inherent parallelism in LabVIEW, avoid overusing sequence structures. Sequence structures guarantee the order of execution and prohibit parallel operations. For example, asynchronous tasks that use I/O devices, such as PXI, GPIB, serial ports, and DAQ devices, can run concurrently with other operations if sequence structures do not prevent them from doing so. Sequence structures also hide sections of the block diagram and interrupt the natural left-to-right flow of data.

When you need to control the execution order, consider establishing data dependency between the nodes. For example, you can use error I/O to control the execution order of I/O. Refer to the *Error Handling* section of Chapter 6, *Running and Debugging VIs*, for more information about error I/O.

Also, do not use sequence structures to update an indicator from multiple frames of the sequence structure. For example, a VI used in a test application might have a **Status** indicator that displays the name of the current test in progress. If each test is a subVI called from a different frame, you cannot update the indicator from each frame, as shown by the broken wire in the Stacked Sequence structure in Figure 8-6.

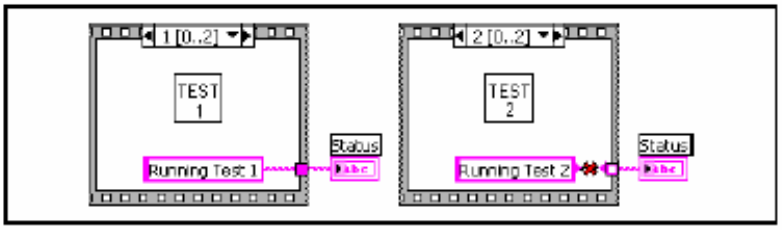


Figure 8-6. Updating an Indicator from Different Frames in a Stacked Sequence Structure

Because all frames of a Stacked Sequence structure execute before any data pass out of the structure, only one frame can assign a value to the **Status** indicator.

Instead, use a Case structure and a While Loop, as shown in Figure 8-7.

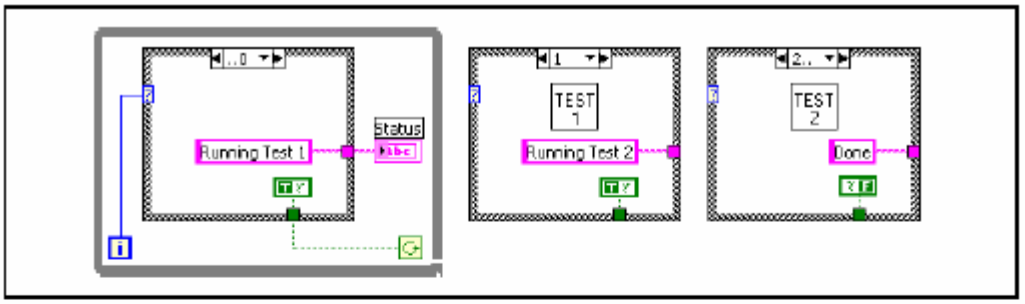


Figure 8-7. Updating an Indicator from Different Cases in a Case Structure

Each case in the Case structure is equivalent to a sequence structure frame. Each iteration of the While Loop executes the next case. The **Status** indicator displays the status of the VI for each case. The **Status** indicator is updated in the case before the one that calls the corresponding subVI because data pass out of the structure after each case executes.

Unlike a sequence structure, a Case structure can pass data to end the While Loop during any case. For example, if an error occurs while running the first test, the Case structure can pass FALSE to the conditional terminal to end the loop. However, a sequence structure must execute all its frames even if an error occurs.

Replacing Sequence Structures

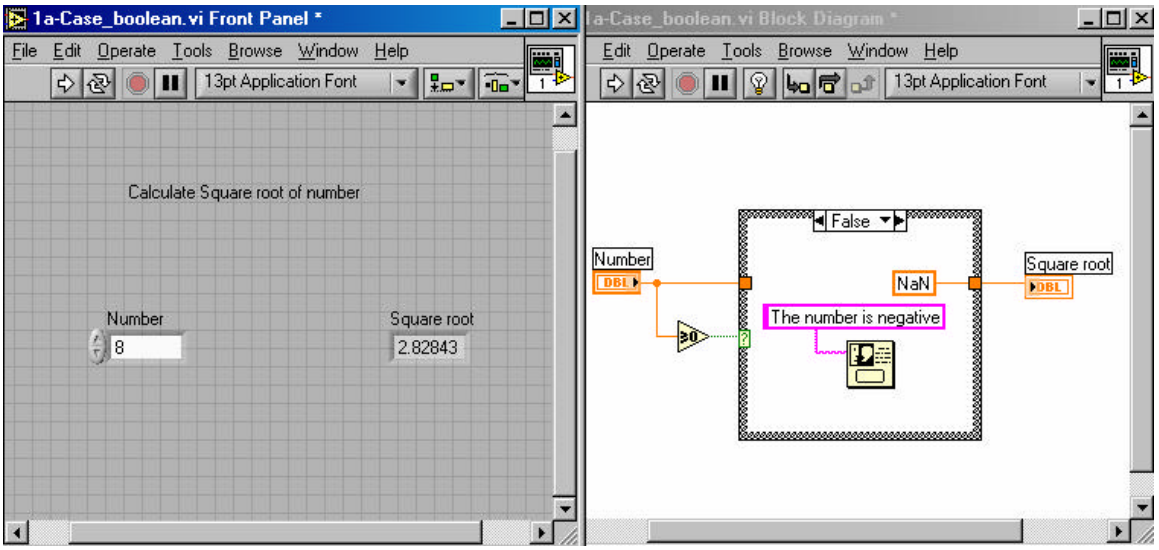
Right-click a Flat Sequence structure and select **Replace with Stacked Sequence Structure** from the shortcut menu to convert a Flat Sequence structure to a Stacked Sequence structure. Right-click a Stacked Sequence structure and select **Replace»Replace with Flat Sequence** from the shortcut menu to convert a Stacked Sequence structure to a Flat Sequence structure.

Case & Sequence Structure Summary

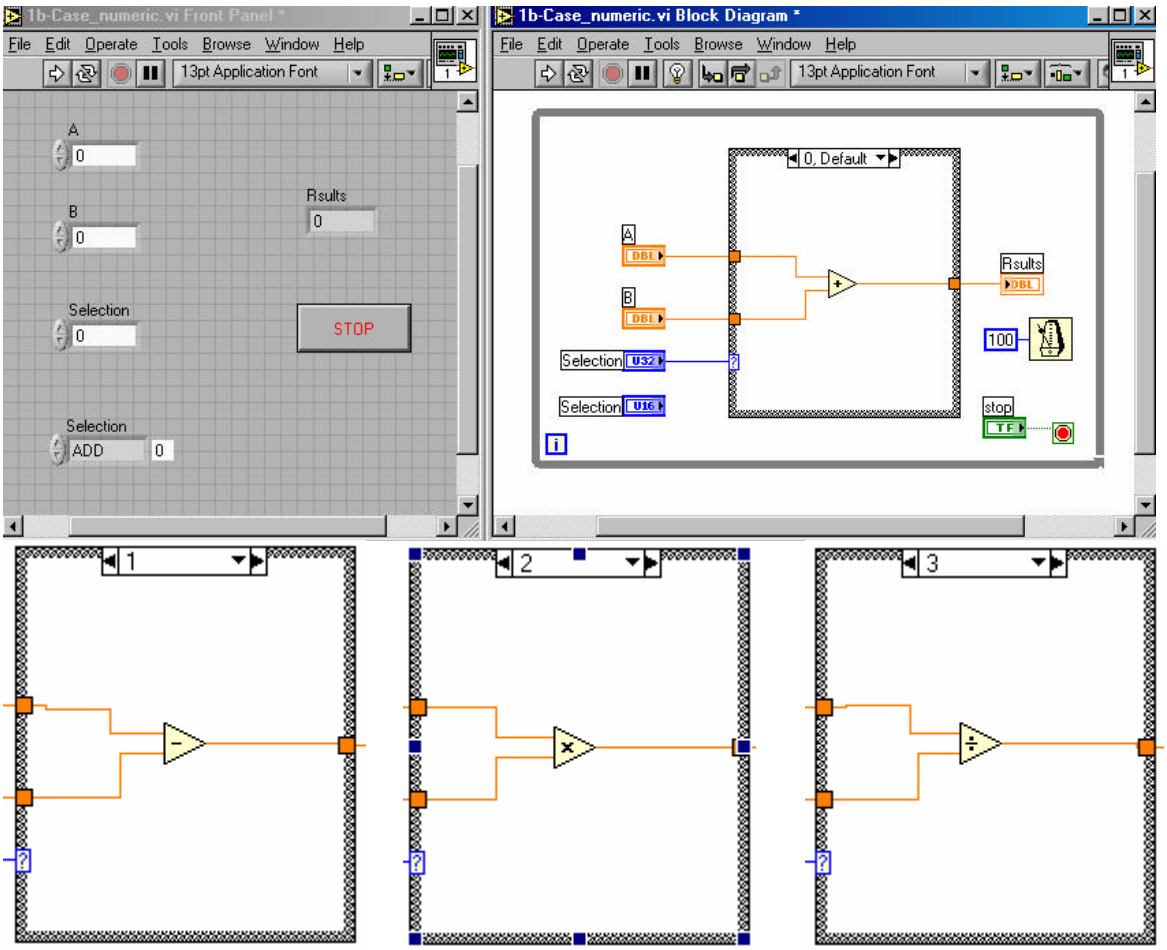
- LabVIEW has two structures to control data flow—the Case structure and the Sequence structure. LabVIEW depicts both structures like a deck of cards; only one case or one frame is visible at a time.
- You use the Case structure to branch to different diagrams depending on the input to the selection terminal of the Case structure. You place the subdiagrams inside the border of each case of the Case structure. The case selection can be Boolean (2 cases), string, or numeric (2^{31}-1 cases). LabVIEW automatically determines the selection terminal type when you wire a Boolean, string, or integer control to it.
- If you wire a value out of one case, you must wire something to that tunnel in every case.
- You use the Sequence structure to execute the diagram in a specific order. The diagram portion to be executed first is placed in the first frame of the structure, the diagram to be executed second is placed in the second frame, and so on.
- You use sequence locals to pass values between Sequence structure frames. The data passed in a sequence local is available only in frames subsequent to the frame in which you created the sequence local, and not in frames that precede the frame.

Case and Sequence examples

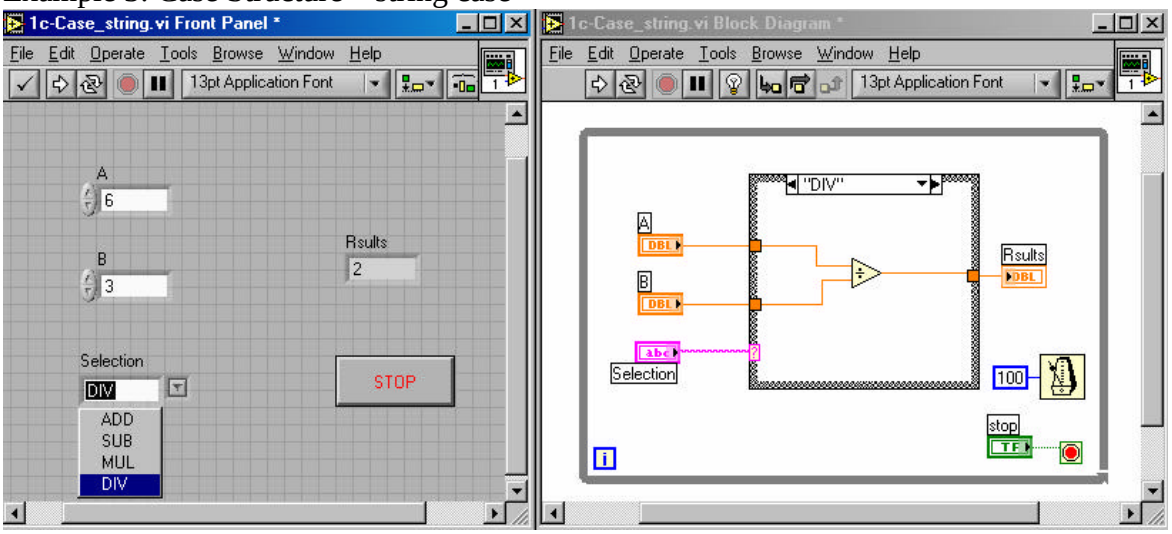
Example 1: Case structure – True and False case



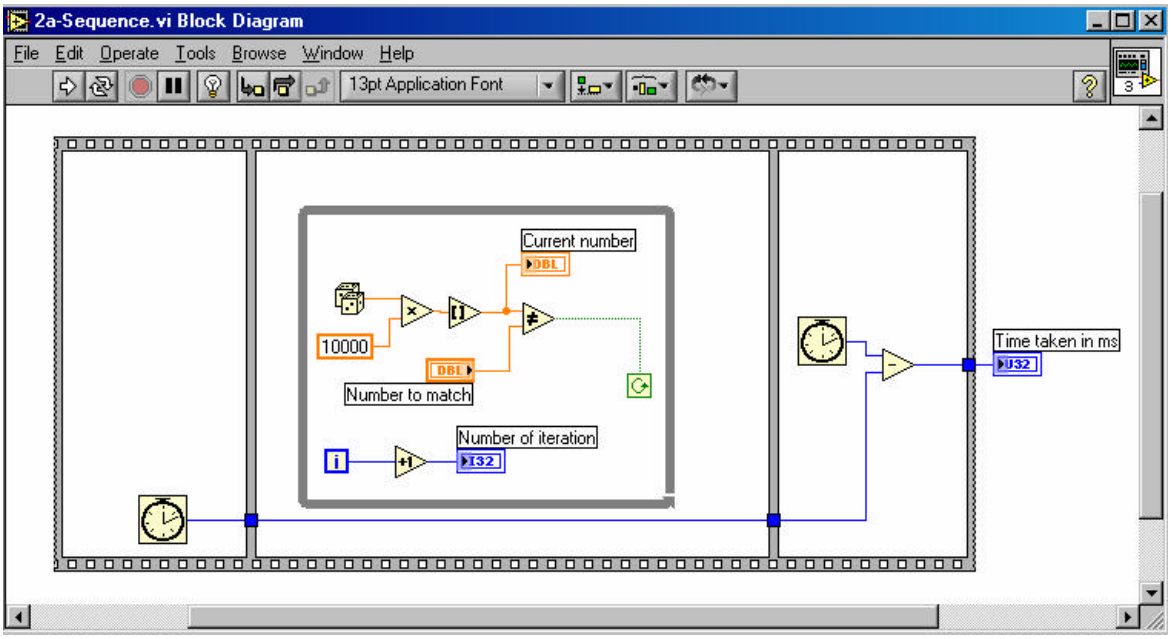
Example 2: Case structure – numeric case



Example 3: Case Structure – string case



Example 4: Case structure – number to match (Flat sequence structure)



Example 5: Case structure – number to match (Stacked sequence structure)