UNIT-3
Virtual Instrumentation

- **Graphical programming**
  - ✓ What is Virtual Instrumentation
  - ✓ Why is Virtual Instrumentation necessary
  - ✓ What is a Virtual Instrument and how is it different from a traditional instrument?
  - ✓ Advantage of Virtual Instrumentation techniques
  - ✓ Advantages of Virtual Instruments versus Traditional Instruments
  - ✓ Layers of Virtual Instrumentation
- **Data types**
- **Concept of WHILE Loop & FOR Loop**
- **Array & Cluster**
  - ✓ Array Vs Cluster
- **Chart & Graph**
  - ✓ Chart Vs Graph
- **Case structure and Sequence structure**
- **Formula nodes**
- **Need of software based instruments for industrial automation.**

Graphical programing:

LabVIEW graphical programs are called **Virtual Instruments** (VIs). VIs run based on the concept of data flow **programming**. This means that execution of a block or a **graphical** component is dependent on the flow of data, or more specifically a block executes when data are made available at all of its inputs.

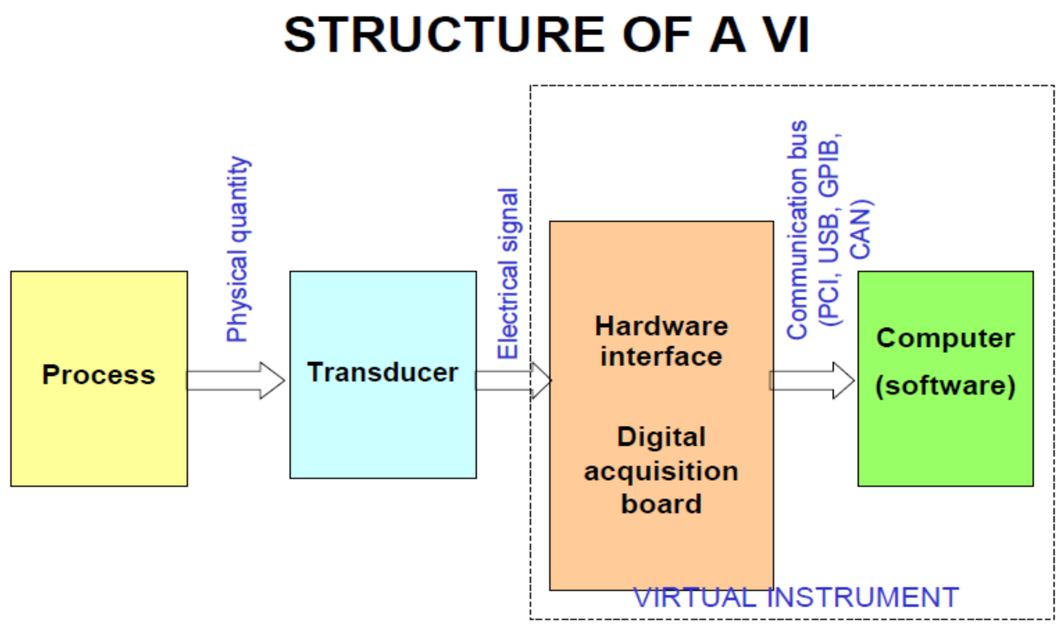
What is Virtual Instrumentation?

Virtual instrumentation combines mainstream commercial technologies, such as C, with flexible software and a wide variety of measurement and control hardware, so engineers and scientists can create user defined systems that meet their exact application needs. With virtual instrumentation, engineers and scientists reduce development time, design higher quality products, and lower their design costs.

Why is Virtual Instrumentation necessary?

Virtual instrumentation is necessary because it delivers instrumentation with the rapid adaptability required for today's concept, product, and process design, development, and delivery. Only with virtual instrumentation can engineers and scientists create the user-defined instruments required to keep up with the world's demands. The only way to meet these demands is to use test and control architectures that are also software centric. Because virtual instrumentation uses highly productive software, modular I/O, and commercial platforms, it is uniquely positioned to keep pace with the required new idea and product development rate

Virtual Instrumentation is the use of customizable software and modular measurement hardware to create user-defined measurement systems, called virtual instruments.



Advantage of VI

- Possibility of measuring on a large number of points and places
- Complex processing of data and measurement information
- Local or remote data storing
- Remote transmission of data through wired or wireless communication
- Statics and forecasts accomplishment
- Flexibility: possibility of extension or adding of new function to the instrument by simple modification of software

- Improving measurement accuracy by statistical processing and compensation of influence factors
- Possibility of adding of new functions for process testing, monitoring and control

What is a virtual instrument and how is it different from a traditional instrument?

Every virtual instrument consists of two parts – software and hardware. A virtual instrument typically has a sticker price comparable to and many times less than a similar traditional instrument for the current measurement task. A traditional instrument provides them with all software and measurement circuitry packaged into a product with a finite list of fixed-functionality using the instrument front panel. A virtual instrument provides all the software and hardware needed to accomplish the measurement or control task. In addition, with a virtual instrument, engineers and scientists can customize the acquisition, analysis, storage, sharing, and presentation functionality using productive, powerful software.

Advantages of Virtual Instruments versus Traditional Instruments

Flexibility: You can easily add additional functions such as a filter routine or a new data view to a virtual instrument.

Storage: Today's personal computers have hard disks that can store dozens of gigabytes which is an absolute plus if you want to process mass data like audio or video.

Display: Computer monitors usually have better color depth and pixel resolution than traditional instruments. Also you can switch easily between different views of the data (graphical, numerical).

Costs: PC add-in boards for signal acquisition and software mostly cost a fraction of the traditional hardware they emulate.

Virtual Instrumentation for Industrial I/O and Control

- PCs and PLCs both play an important role in control and industrial applications. PCs bring greater software flexibility and capability, while PLCs deliver outstanding ruggedness and reliability. But as control needs become more complex, there is a recognized need to accelerate the capabilities while retaining the ruggedness and reliabilities.
- Multi domain functionality (logic, motion, drives, and process) – the concept supports multiple I/O types. Logic, motion, and other function integration is a requirement for increasingly complex control approaches.
- Software tools for designing applications across several machines or process units – the software tools must scale to distributed operation.

Layers of Virtual Instrumentation

- **Application Software:** Most people think immediately of the application software layer. This is the primary development environment for building an application.
- **Test and Data Management Software:** Above the application software layer the test executive and data management software layer. This layer of software incorporates all of the functionality developed by the application layer and provides system-wide data management.
- **Measurement and Control Services Software:** The last layer is often overlooked, yet critical to maintaining software development productivity.

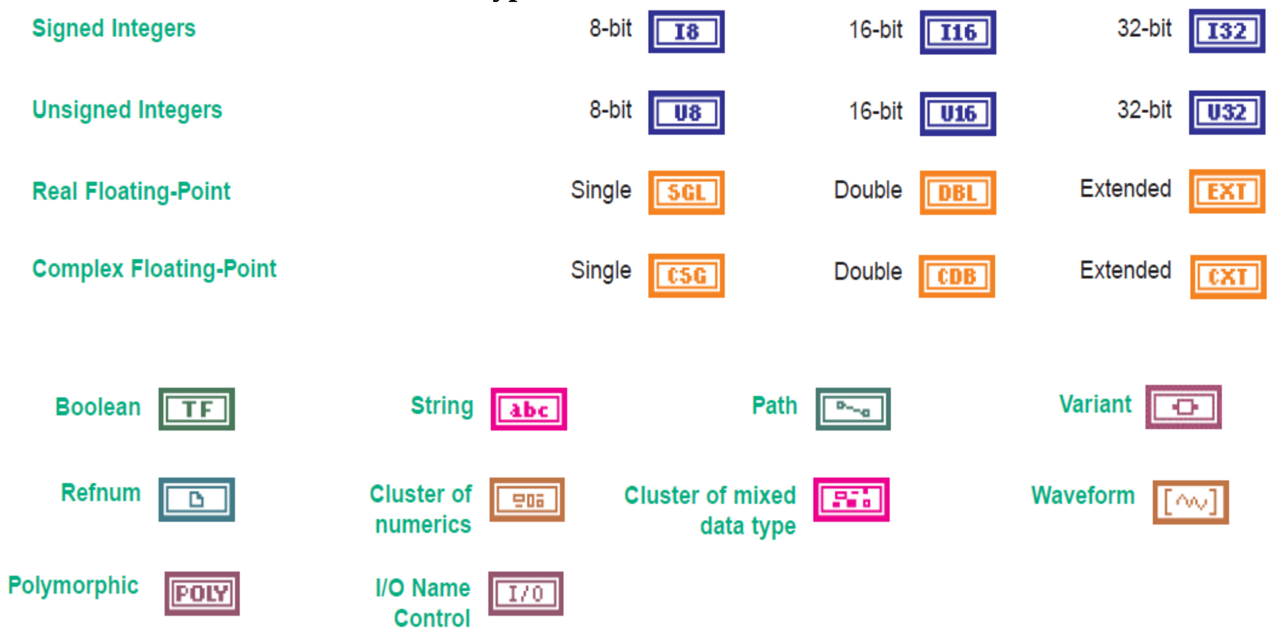
Data types

There are four basic data types in LabVIEW

- ✓ **Floating Point & Integer:** Floating point and integer are numeric types and simply hold numbers
- ✓ **Boolean:** Boolean stores true and false values
- ✓ **String:** Strings store text and characters

Example

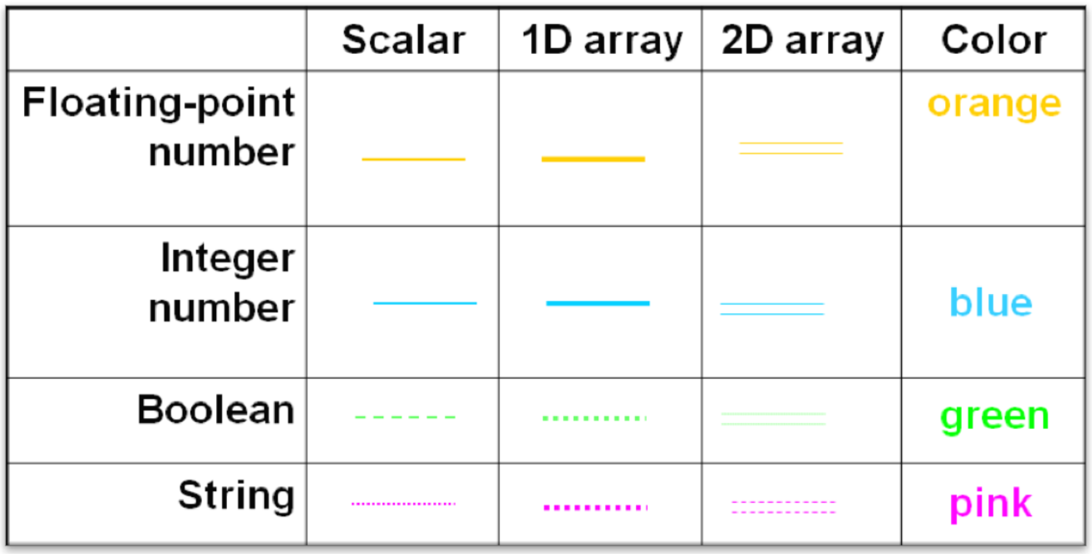
Control & Indicator Terminal Data Type:



Signed Integers	8-bit I8	16-bit I16	32-bit I32
Unsigned Integers	8-bit U8	16-bit U16	32-bit U32
Real Floating-Point	Single SGL	Double DBL	Extended EXT
Complex Floating-Point	Single CSG	Double CDB	Extended CXT

Boolean TF	String abc	Path	Variant
Refnum	Cluster of numerics	Cluster of mixed data type	Waveform
Polymorphic POLY	I/O Name Control I/O		

Wire Data Type:

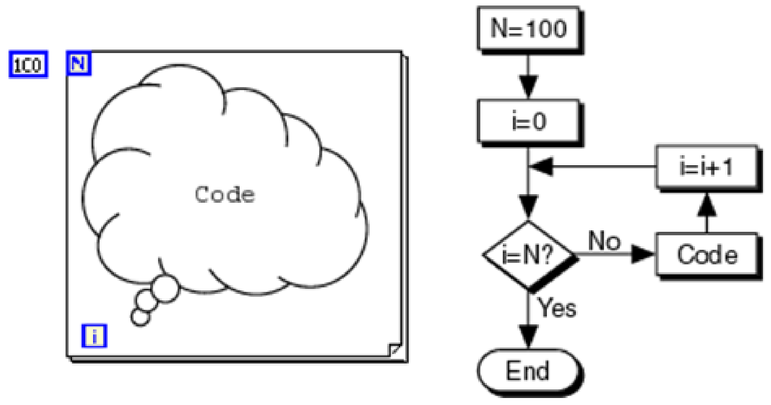


	Scalar	1D array	2D array	Color
Floating-point number	—	━	═	orange
Integer number	—	━	═	blue
Boolean	- - - - - -	··········	═	green
String	··········	··········	═	pink

Concept of Loops

For Loop

A For Loop executes a sub diagram a set number of times figure below shows a For Loop in LabVIEW, FOR Loop – Repeats enclosed instructions for a predetermined amount of iterations (N). Where, Index i, holds current iteration number

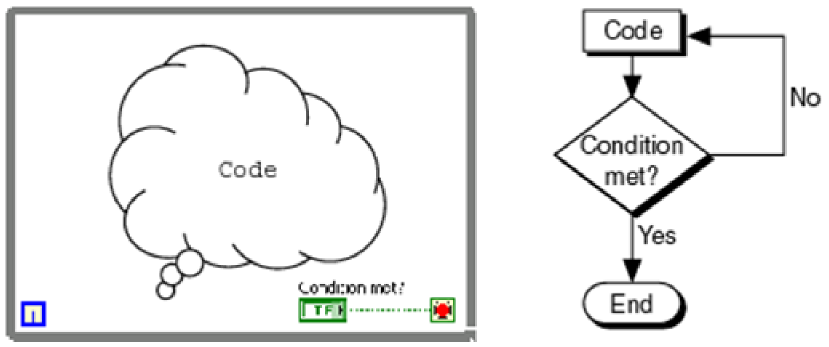


```
N=100;
i=0;
Until i=N:
Repeat (code;i=i+1);
End;
```

Fig1: For loop in LabVIEW

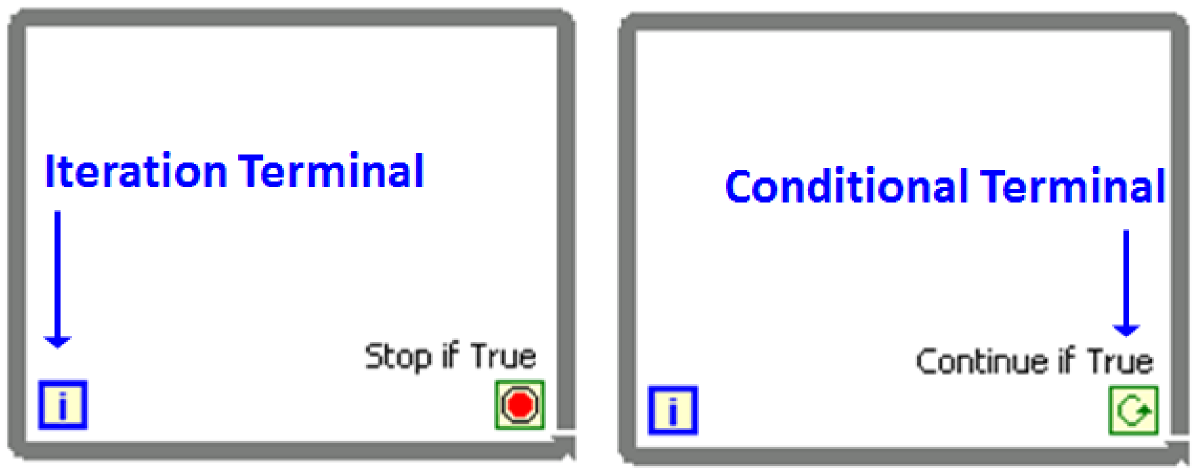
While Loop

A While Loop executes a sub diagram a set number of times figure below shows a While Loop in LabVIEW. WHILE Loop – Repeats enclosed instructions until stop condition is met. Where, Index i, holds current iteration number



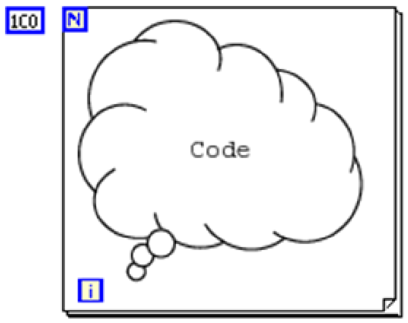
```
Repeat (code);
Until Condition met;
End;
```

Fig1: While loop in LabVIEW



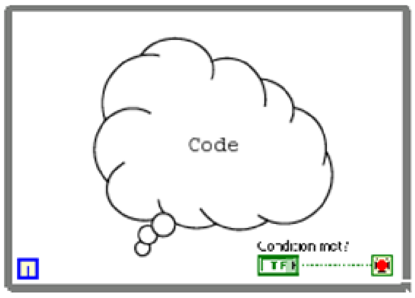
Difference between For Loop & While Loop:

For Loop



- Executes a set number of times unless a conditional terminal is added
- Can execute zero times
- Tunnels automatically output an array of data

While Loop



- Stops executing only if the value at the conditional terminal meets the condition
- Must execute at least once
- Tunnels automatically output the last value

Arrays & Cluster

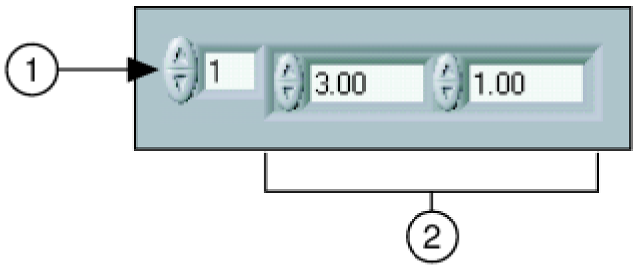
Array:

Arrays group data elements of the same type. An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as $2^{31} - 1$ elements per dimension, memory permitting.

You can build arrays of numeric, Boolean, path, string, waveform, and cluster data types. Consider using arrays when you work with a collection of similar data and when you perform repetitive computations. Arrays are ideal for storing data you collect from waveforms or data generated in loops, where each iteration of a loop produces one element of the array.

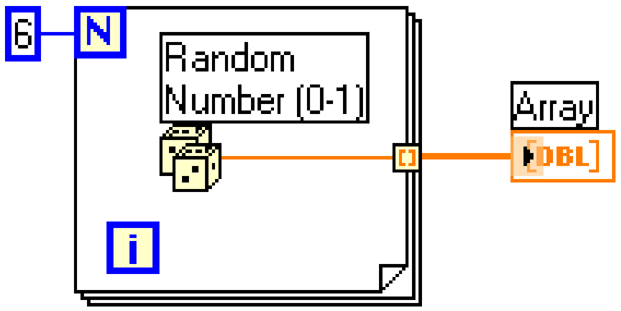
You cannot create an array of arrays. However, you can create an array of clusters, where each cluster contains one or more arrays.

Array elements are ordered. An array uses an index so you can readily access any particular element. The index is zero-based, which means it is in the range 0 to N−1, where N is the number of elements in the array. For example, if you create an array of the planets in the solar system, N= 9 for the nine planets, so the index ranges from 0 to 8. Earth is the third planet, so it has an index of 2.

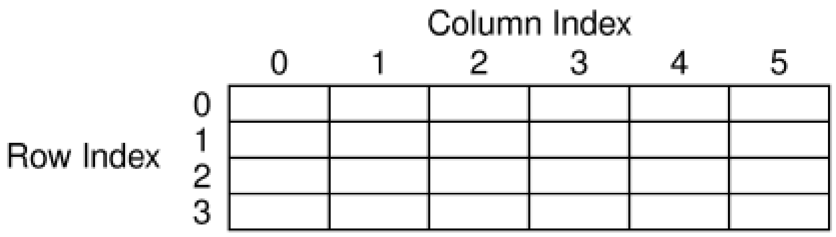


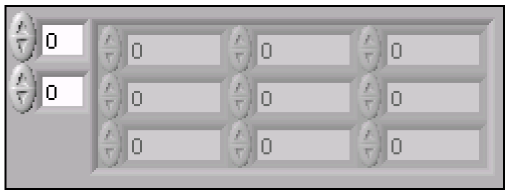
1- Index display, 2- Element display

Creating Array Constants

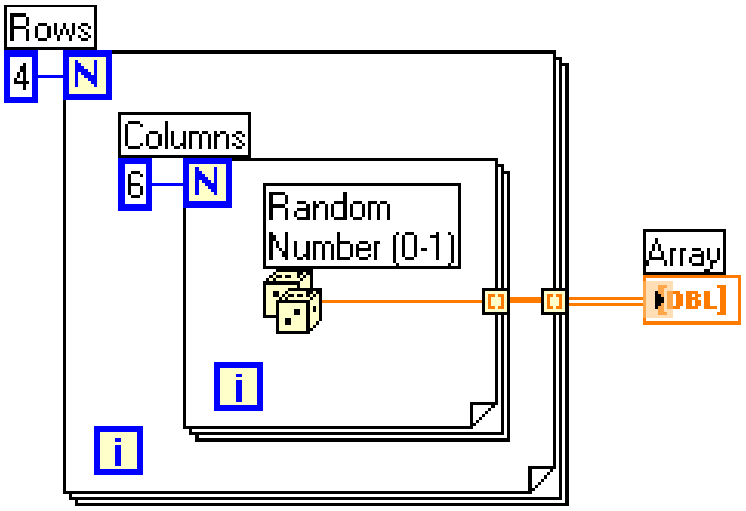


2D Array:





Creating Two-Dimensional Arrays



Summary, Tips, and Tricks on Arrays

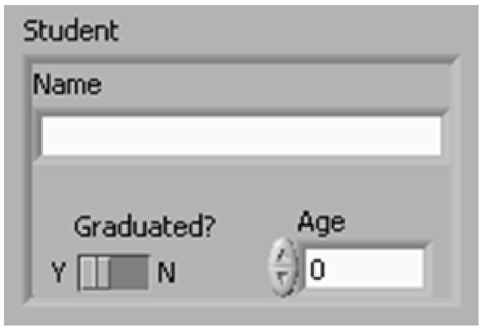
- ✓ Arrays group data elements of the same type. You can build arrays of numeric, Boolean, path, string, waveform, and cluster data types.
- ✓ The array index is zero-based, which means it is in the range 0 to n−1, where n is the number of elements in the array.

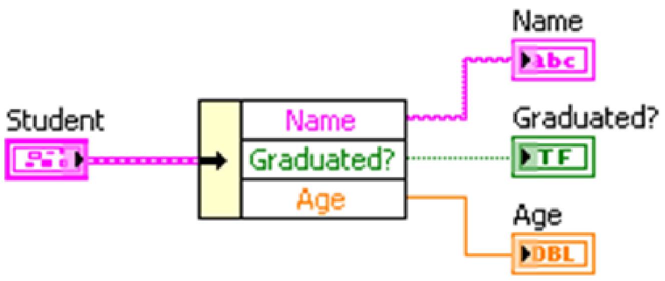
- ✓ You must insert an object in the array shell before you use the array on the block diagram. Otherwise, the array terminal appears black with an empty bracket.
- ✓ To create an array control or indicator, select an array on the Controls >> Array & Cluster palette, place it on the front panel, and drag a control or indicator into the array shell.
- ✓ If you wire an array to a For Loop or While Loop input tunnel, you can read and process every element in that array by enabling auto-indexing.
- ✓ Use the Array functions located on the Function >> All Functions >> Array palette to create and manipulate arrays.
- ✓ By default, LabVIEW enables auto-indexing in For Loops and disables auto-indexing in While Loops.
- ✓ Polymorphism is the ability of a function to adjust to input data of different data structures.

Cluster:

Clusters group data elements of mixed types, such as a bundle of wires, as in a telephone cable, where each wire in the cable represents a different element of the cluster.

Like an array, a cluster is either a control or an indicator. A cluster cannot contain a mixture of controls and indicators.





✓ Array Vs Cluster

- Clusters differ from arrays in that they are a fixed size
- Clusters can contain mixed data types; arrays contain only one data type
- Like an array, a cluster is either a control or an indicator and cannot contain a mixture of controls and indicators

Chart & Graph

Chart:

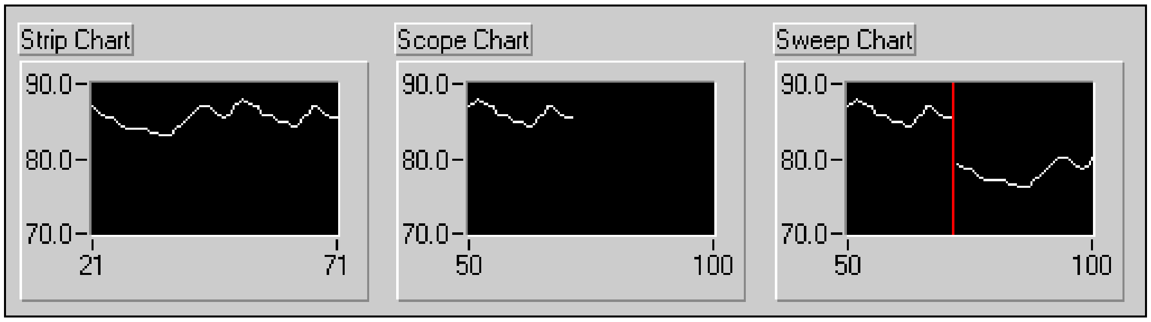
The waveform chart is a numeric indicator that displays one or more plots. The waveform chart is located on the Controls >> Graph Indicators palette. Waveform charts can display single or multiple plots. Fig. below shows the elements of a multiple plot waveform chart.

Charts use three different modes to scroll data. Right-click the chart and select Advanced >> Update Mode from the shortcut menu. Select Strip Chart, Scope Chart, or Sweep Chart. The default mode is Strip Chart.

A strip chart shows running data continuously scrolling from left to right across the chart. A scope chart shows one item of data, such as a pulse or wave, scrolling partway across the chart from left to the right.

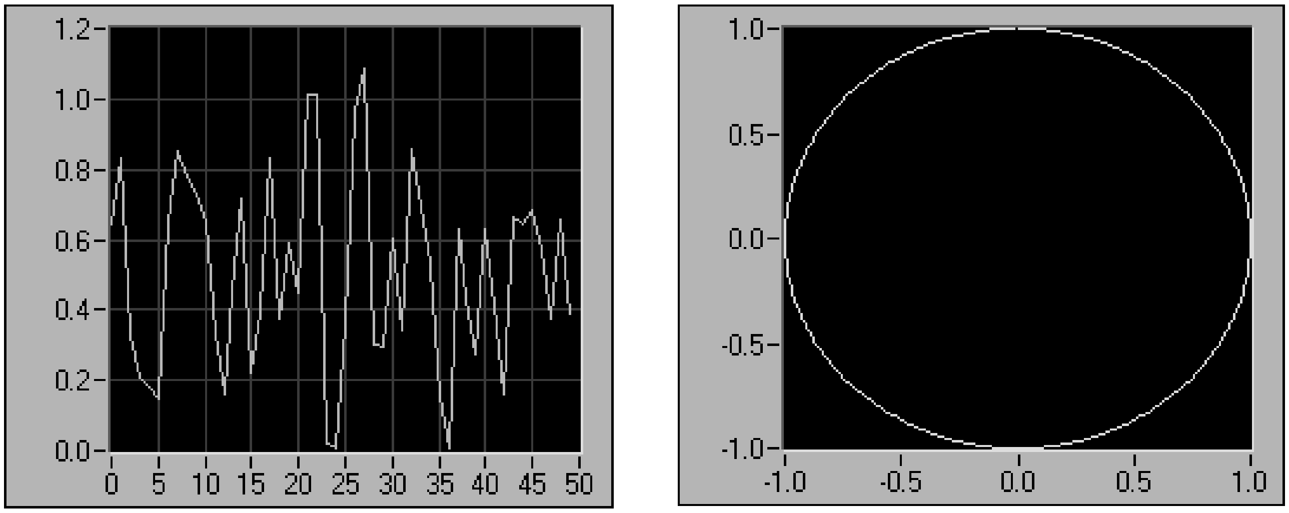
A sweep chart is similar to an EKG display. A sweep chart works similarly to a scope except it shows the older data on the right and the newer data on the left separated by a vertical line.

The scope chart and sweep chart have retracing displays similar to an oscilloscope. Because there is less overhead in retracing a plot, the scope chart and the sweep chart display plots significantly faster than the strip chart.



Waveform and XY Graphs

Waveform graphs display evenly sampled measurements. XY graphs display any set of points, evenly sampled or not.

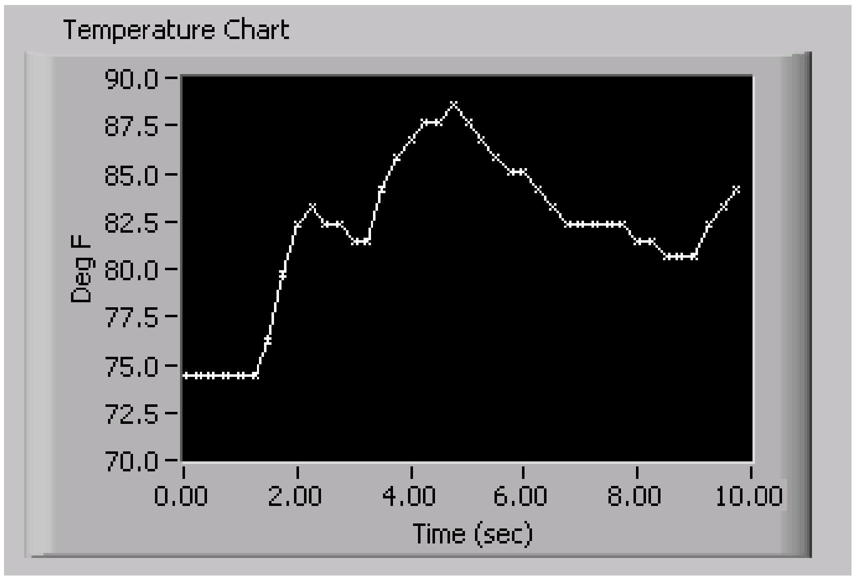


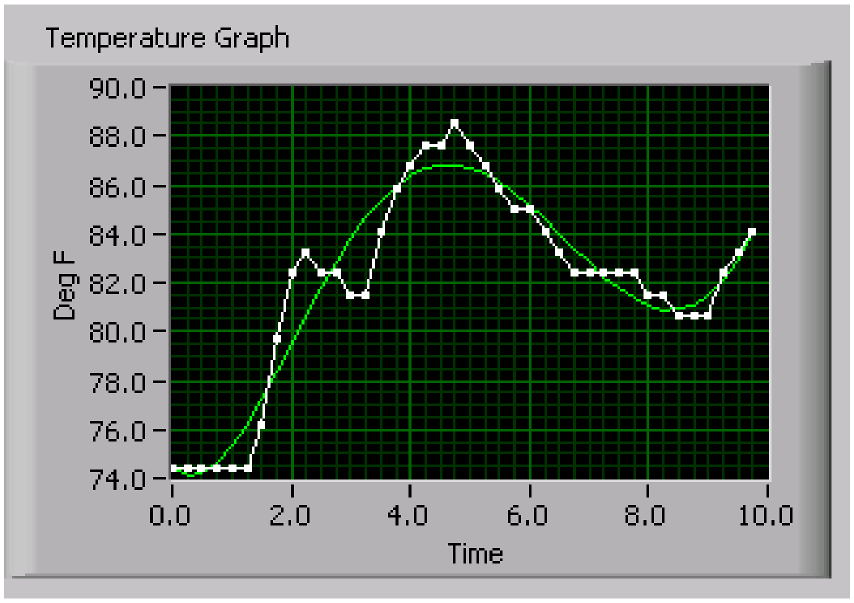
Waveform & X-Y graph

The waveform graph plots only single-valued functions, as in y = f(x), with points evenly distributed along the x-axis, such as acquired time-varying waveforms.

The XY graph is a general-purpose, Cartesian graphing object that plots multivalued functions, such as circular shapes or waveforms with a varying time base. Both graphs can display plots containing any number of points. Both types of graphs accept several data types, which minimizes the extent to which you must manipulate data before you display it.

- ✓ Single-Plot Waveform Graphs
- ✓ Multi-Plot Waveform Graphs
- ✓ Single-Plot XY Graphs
- ✓ Multi-Plot XY Graphs





✓ Charts vs Graph:

Graphs and charts differ in the way they display and update data. VIs with graphs usually collect the data in an array and then plot the data to the graph, which is similar to a spreadsheet that first stores the data then generates a plot of it.

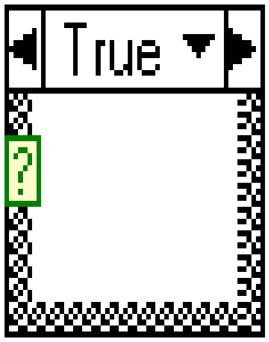
In contrast, a chart appends new data points to those already in the display to create a history. On a chart, you can see the current reading or measurement in context with data previously acquired.

Case structure and Sequence structure

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.

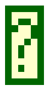
✓ Case Structure:

A Case structure 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 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.

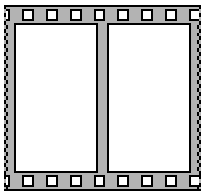


Specify a default case for the Case structure to handle out-of-range values. Otherwise, you must explicitly list every possible input value. For example, if the selector is an integer and you specify cases for 1, 2, and 3, you must specify a default case to execute if the input value is 4 or any other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

✓ Sequence Structures:

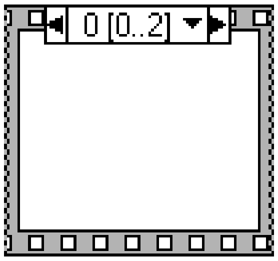
Flat Sequence Structure

The Flat Sequence structure 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

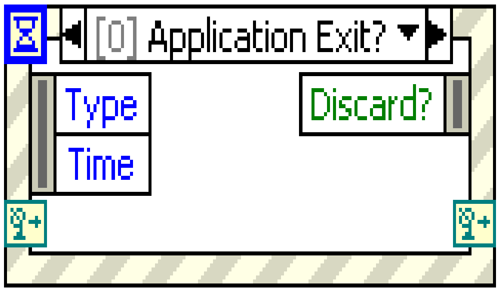
The Stacked Sequence structure stacks each frame so you see only one frame at a time and executes frame 0, then frame 1, and so on until the last frame executes. The Stacked Sequence structure returns data only after the last frame executes. Use the Stacked Sequence structure if you want to conserve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

✓ Event Structure Components

Use the Event structure to handle events in a VI. The Event structure works like a Case structure with a built-in Wait on Notification function. The Event structure can have multiple cases, each of which is a separate event-handling routine. You can configure each case to handle one or more events, but only one of these events can occur at a time. When the Event structure executes, it waits until one of the configured events occur, then executes the corresponding case for that event. The Event structure completes execution after handling exactly one event. It does not implicitly loop to handle multiple events. Like a Wait on Notification function, the Event structures can time out while waiting for notification of an event. When this occurs, a specific Timeout case executes.



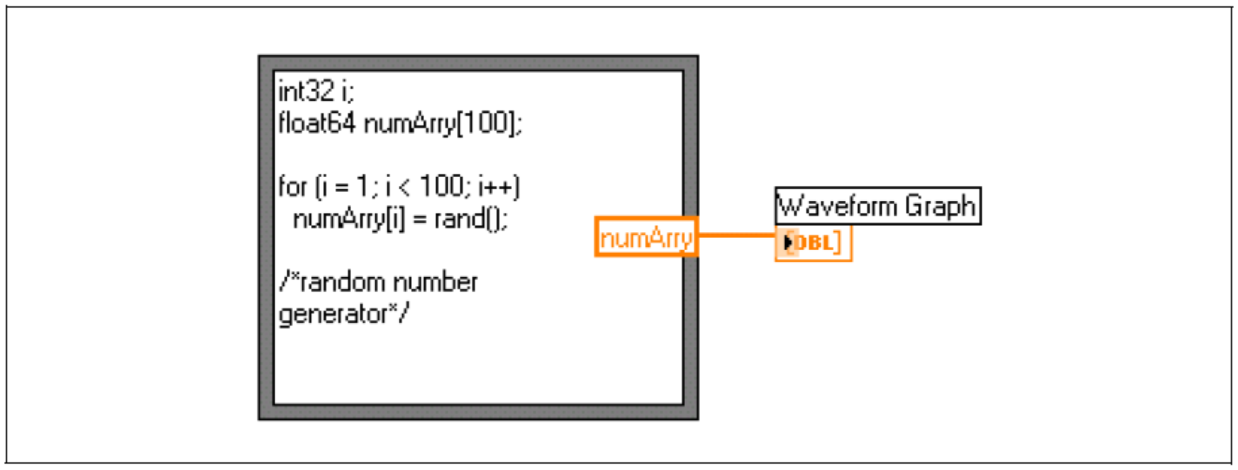
The event selector label at the top of the Event structure indicates which events cause the currently displayed case to execute. View other event cases by clicking the down arrow next to the case name and selecting another case from the shortcut menu.

Formula nodes

The Formula Node is a convenient text-based node you can use to perform mathematical operations on the block diagram. You do not have to access any external code or applications, and you do not have to wire low-level arithmetic functions to create equations. In addition to text-based equation expressions, the Formula Node can accept text-based versions of if statements, while loops, for loops, and do loops, which are familiar to C programmers. These programming elements are similar to what you find in C programming but are not identical. Formula Nodes are useful for equations that have many variables or are otherwise complicated and for using existing text-based code. You can copy and paste the existing text-based code into a Formula Node rather than recreating it graphically.

Formula Nodes use type checking to make sure that array indexes are numeric data and that operands to the bit operations are integer data. Formula Nodes also check to make sure array indexes are in range. For arrays, an out-of-range value defaults to zero, and an out-of-range assignment defaults to *nop* to indicate no operation occurs. Formula Nodes also perform automatic type conversion

The Formula Node is a resizable box similar to the For Loop, While Loop, Case structure, Stacked Sequence structure, and Flat Sequence structure. However, instead of containing a subdiagram, the Formula Node contains one or more C-like statements delimited by semicolons, as in the following example. As with C, add comments by enclosing them inside a slash/asterisk pair (/*comment*/).

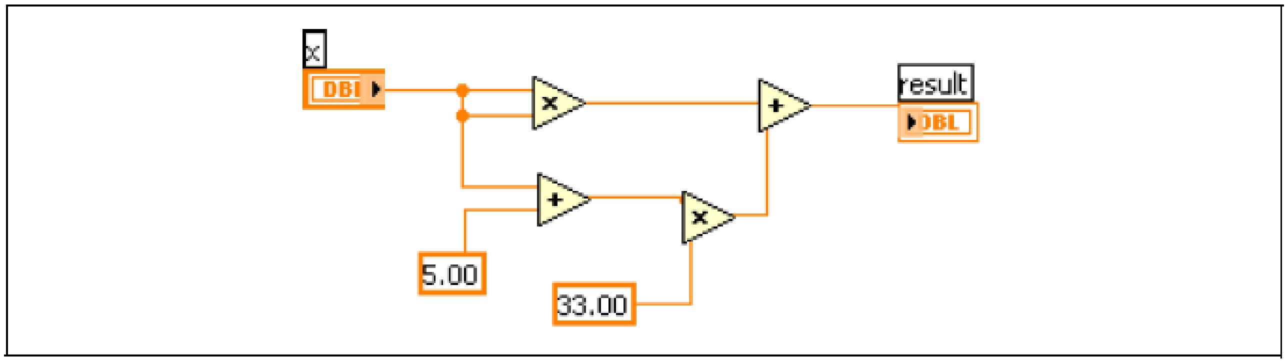


```
int32 i;
float64 numArry[100];

for (i = 1; i < 100; i++)
  numArry[i] = rand();

/*random number
generator*/
```

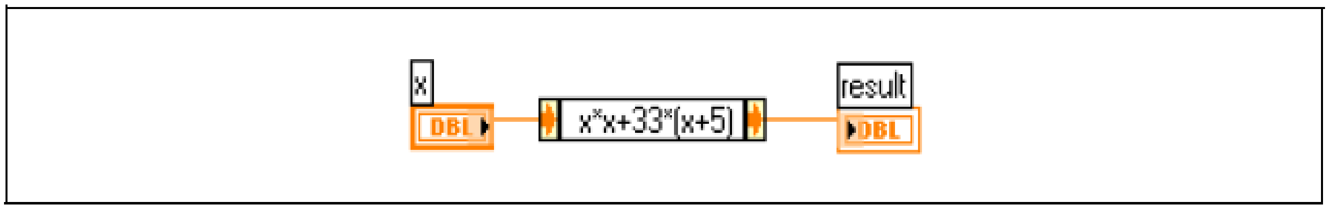
Formula Node

Example, consider this simple equation: $\boldsymbol{x \times x + 33 \times (x + 5)}$

The block diagram in the following figure uses Numeric functions to represent this equation.



Use an Expression Node, as shown in the following figure, to create a much simpler block diagram.



Math Script Node

Need of software based instruments for industrial automation.

- PCs and PLCs both play an important role in control and industrial applications. PCs bring greater software flexibility and capability, while PLCs deliver outstanding ruggedness and reliability. But as control needs become more complex, there is a recognized need to accelerate the capabilities while retaining the ruggedness and reliabilities.
- Multi domain functionality (logic, motion, drives, and process) – the concept supports multiple I/O types. Logic, motion, and other function integration is a requirement for increasingly complex control approaches.
- Software tools for designing applications across several machines or process units – the software tools must scale to distributed operation.