A Study and Comparison of Data Transfer Methods in LabVIEW

Shehryar Humayun, Maria Mehmood, Faran Mahmood and Qamar UlIslam

Department of Electrical Engineering, Institute of Space Technology, Islamabad, Pakistan

Abstract: For large applications it is often necessary to pass data among different pieces of codes. Passing data between different routines of a same piece of code or between different codes of separate applications is often a difficult task. It generally involves measures to determine whether the data is continuous or non-continuous. This paper discusses techniques of passing data between different nodes with focus on LabVIEW. Different methods are discussed, each with its advantages, disadvantages and memory considerations. This involves analyzing of mechanisms like buffering for of the methods. Implementation in a typical LabVIEW environment is also given. The paper also debates when to use which method and its effectiveness. Finally benchmark results of number of operations per iteration for Read and Write of two common methods are given.

Key words: Data transfer methods • LabVIEW • Global Variable • Queue • Notifier • Shared Variable

INTRODUCTION

Large applications are developed by combining many different subroutines that work independently but, however, interlinked due to the same data that is being processed by all of them.

For Example: Consider a system where data is acquired from some source like oscilloscope on one computer and then passed to another computer to be processed and converted in a waveform on the LabVIEW interface. Now to make this system efficient and compatible we need to develop several small routines that initializes the oscilloscope interface, trigger data acquisition, handle the errors, convert to array etc. All these routines will basically work on the same data that we take from the oscilloscope. So to make them work together smoothly the user is faced with the problem of passing data among them.

Data transfer is not only used to allow the user to exchange data but also gives the opportunity to determine data flow that which VI will execute before which one. This is only possible because necessary data transfer makes the VIs dependent on each other.

Now once it is clear that data transfer is necessary the user is faced with the dilemma of which method to choose for transferring the data. This paper will discuss those methods turn by turn and present the pros and cons of each one to present a clear picture that which method should be adopted in which scenario.

MATERIALS AND MEHTODS

Global Variables: Global Variables are one of the options for inter-VI data transmission. It can be inferred from the name, ‘global’, that these variables can be accessed from any virtual instrument which currently resides in the memory space. Whenever a global variable is created, LabVIEW application generates a VI (global). This global VI contains no block diagram, but a front panel. We call this VI as Global Variable Holder. User can put indicators and controls to this VI in order to set whether it would be read from or written to, from another VI. User also describes the type of data to be transferred using this global VI. Several objects can be placed simultaneously on a single Global Variable Holder or separate global VIs can be created for each object.

Notifiers: Another possibility for transferring data (Boolean, string, num) between two VIs in a LabVIEW based program is use of Notifiers. However, they can only be used for communication between VIs executing on the same computer. Notifiers do not themselves transfer data between VIs but they aid in making this transmission
Queue System: Queue functions or objects can form yet another system for data communication in LabVIEW between two different VIs.

Types: Two important types of queues used in LabVIEW environment are:

- Last-In First-Out (LIFO)
- First-In First-Out (FIFO)

Both forms of queues work by forming lines of data. Whenever there is a queue involved in a system, a buffer must also be the part of the system. However the way in which data is queued determines whether it will be Last-In First-Out or First-In First-Out.

Last-In First-Out is actually stacking of data. It can be compared to a real world example of multiple saucers stacked together. The saucer to come out first would be the last saucer stacked. It actually works on a push and pop mechanism.

First-In First-Out depicts a queuing system, where data gets in a buffer from one end and retrieved from the other.

Shared Variables: Shared Variable is a recent feature added in later versions of LabVIEW. It can solve problems related to data transfer between VIs. Majority of the features of a shared variable can be compared to that of a global variable except that it can only be implemented within a single program. However, as opposed to global variable, a shared variable can be accessed over a network.

RESULTS AND DISCUSSION

Global variables

Pros:

- For data transfer between two VIs, it is the easiest method.
- From data continuity point of view, it is best to be used for variables which contain non-continuous data.

Cons:

- Global Variables collect and store only one value at a time, so they cannot be used for variables which carry continuously changing data.
- User needs to create a system for inspecting (preferably by using loops) if the values are changing or not.

Memory Consideration: Special attention is needed while using global variables, especially with large arrays. Whenever receiving node accepts data from a global variable, copy of that data is produced by LabVIEW. For small data operations global variables will suffice,
but when dealing bigger arrays the load on memory increases terribly. Even if a single element of an array of N elements is accessed, whole array is copied into the memory. Eventually it results in slow and inefficient performance.

**Notifiers**

**Pros:**

- Notifiers enable a programmer to transfer data between VIs without any polling. There is no need for a loop that continuously checks the value of a variable. So part of the code which needs data to be received stops working so computer power is saved.
- No buffers are needed.

**Cons:**

- Synchronization is necessary between rate of transmission and reception of notifications to prevent data loss.
- As no buffers are included with notifiers, so to prevent data loss a feedback system can be put in place.
- Notifiers can be used for continuous data transfer as well, however it is not recommended.
- Notifiers do not work across a network. Consequently notifiers cannot be used with applications running a VI server.

**Memory Consideration:** While using notifiers inside a loop, a new reference is created by LabVIEW at the start of each loop iteration. This gradually increases memory usage as each reference needs four additional bytes. However when VI is stopped, these memory bytes are released. But if your program needs to run for longer times (e.g. a continuous monitoring system) then memory requirements keeps increasing. A remedy to this problem can be use of Release Notifier object inside the loop, so that after every iteration, the used memory space is released.

**Queue System**

**Pros:**

- One important advantage in a queued system is that it does not require synchronization. For instance if the receiving node has not dequeued the buffer and it becomes full, in that case transmitting node automatically waits for de-queuing process.
- This insures no loss of data.
- Most suitable system for continuous data transmission.

**Cons:**

- Data elements in a queue cannot be retrieved randomly. Only method is to de-queue data elements on sample by sample basis.
- It needs a buffer to be implemented.
- It requires a loop to continuously poll and check whether data has arrived.

**Memory Consideration:** Whenever a queue is implemented, the memory reserved for the queue is never dislocated till the queue is finished (mostly when the VI is stopped). If data comes in discontinuous bursts, then for some period of time, the queue will go unused and so the memory. Performance can be increased by dynamically allocating the queue memory at run time rather than pre-allocating memory.

**Shared Variables:**

**Pros:**

- Shared Variables are enabled to be published over a network and thus create Published Shared Variable. This variable can then be accessed over a network and VI server.
- No buffer created, lesser memory space required.

**Cons:**

- This feature only available in version 8 and above.
- As there is no buffer, so it can only stock a single value at a given time. So loss of data might occur for continuous transmissions.

**Memory Consideration:** Shared Variables can put additional load on the memory. One reason for this load is SVE (Shared Variable Engine) which has to be loaded for using Shared Variables. Also while using SV for network publishing creates additional references and thus puts load on memory.

**CONCLUSION**

To conclude the LabVIEW package provides the user with many different methods to transfer data not only among VIs on the same computer but also among VIs
Table 1: Comparison of different data Transfer Methods

<table>
<thead>
<tr>
<th>Feature</th>
<th>Global Variable</th>
<th>Notifier</th>
<th>Queues</th>
<th>Shared Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmitting Mode</td>
<td>Non-continuous</td>
<td>Both</td>
<td>Continuous</td>
<td>Non-Continuous</td>
</tr>
<tr>
<td>Availability</td>
<td>Base Package</td>
<td>Base Package</td>
<td>Base Package</td>
<td>Only V 8.0 &amp; above</td>
</tr>
<tr>
<td>Poling</td>
<td>Required</td>
<td>Not required</td>
<td>Required</td>
<td>Required</td>
</tr>
<tr>
<td>Buffer</td>
<td>None</td>
<td>None</td>
<td>Yes</td>
<td>None</td>
</tr>
</tbody>
</table>

![Fig. 4: Global Variable Vs Shared Variable](image)

Running on different computers. The choice of which method to use depends on many factors e.g. continuity of data, availability of buffer, number of variables to be passed, type of data to be shared etc. However, eventually it all comes down to the type of application being developed. Table 1 shows the comparison of different methods that the researchers inferred after implementation.

For comparing the performance of shared and global variables, a benchmarking VI was used as described by [8]. For a global variable, the write performance is marginally higher than shared variable. However, there is a significant increase in performance for Read in global variable. While Read/Write combined test showed almost the same results.

REFERENCES