

LabVIEW™ 高级性能和通讯课程手册

原著：LabVIEW™ Advanced Performance & Communication Course Manual

© National Instruments Corporation (December 2001)

原著来源：SimWe仿真论坛-虚拟仪器版

摘译者：南京大学电子系 Mebusw 2004.12. (第一版)

请免费传播，不得用于商业目的!

谨以此摘译本向NI致敬，同时鄙视那些以抄袭为手段，编写出版高价教科书赚取利润的骗子老师。

摘译本面向的是有一定LabVIEW基础的读者，仅在原版基础上摘录重点内容进行了翻译，但保留了相应原文供参考，建议只看中文和图例即可，不重要的地方我只列出原文或干脆删去。去掉了所有的练习，融入了正文中，请参考原本。个别地方以7.0版本为准作了改动和补充，所有翻译和中文字输入由本人独自完成。

愿与大家交流，并欢迎高手们批评指正：mebusw@163.com

原著目录

Lesson 1

Data Management and Synchronization

A. Event-Driven Programming.....	1-2
B. Local and Global Variables	1-12
C. VI-Based Global Variables	1-18
D. Advanced Synchronization Techniques.....	1-22
E. Synchronization of Tasks.....	1-26

Lesson 2

Improving Performance

A. Multitasking, Multithreading, and Multiprocessing	2-2
B. Defining and Allocating Threads and Priorities in LabVIEW	2-5
C. Monitoring VI Performance Using the Profile Window	2-8
D. Speeding Up Your VIs.....	2-13
E. System Memory Issues	2-28
F. Optimizing VI Memory Use	2-31

Lesson 3

TCP/IP

A. Computer Networks.....	3-2
B. Introduction to TCP/IP.....	3-4
C. Client/Server Model.....	3-8
D. TCP/IP VIs and Functions	3-13
E. LabVIEW Web Server.....	3-24

Lesson 4

DataSocket

A. Components of DataSocket	4-2
B. DataSocket VIs	4-5
C. DataSocket Server Manager	4-10
D. Front Panel DataSocket	4-14
E. Variant Functions.....	4-15
F. Bi-directional Communication	4-22
G. Reading and Writing Data from Other Sources.....	4-27
H. Creating Interactive Web Pages Using DataSocket.....	4-29

Lesson 5

VI Server

A. What is VI Server?.....	5-2
B. VI Server Programming Model	5-5
C. VI Server Functions	5-6
D. Strictly Typed VI Refnums.....	5-17
E. Remote Communication	5-23
F. VI Server Configuration for External Applications.....	5-24

Lesson 6

Calling and Creating Shared Libraries (DLLs)

A. What is a Shared Library/DLL?.....	6-2
B. Accessing Shared Libraries/DLLs	6-4
C. Debugging Call Library Function Errors.....	6-19
D. Building DLLs in LabVIEW	6-20
E. Building DLLs on Other Operating Systems.....	6-24

Lesson 7

ActiveX Automation

A. OLE and ActiveX	7-2
B. ActiveX Features in LabVIEW.....	7-5
C. LabVIEW ActiveX Automation Server.....	7-6

Lesson 8

LabVIEW ActiveX Automation Client and ActiveX Container

A. LabVIEW as an ActiveX Automation Client	8-2
B. ActiveX Automation Client VIs	8-3
C. Programming Model for Automation Functions	8-4
D. ActiveX Variant to LabVIEW Data.....	8-5
E. Automation with LabVIEW and Microsoft Excel	8-6
F. Remote Automation.....	8-10
G. ActiveX Containers.....	8-12

Lesson 9

Error Trapping Techniques

A. Error Trapping Basics.....	9-2
B. Debugging Basics	9-3
C. Memory Leaks	9-6
D. Multithreading Errors	9-11
E. Using Configuration Files.....	9-15

Lesson 1

Data Management and Synchronization

Filter and Notify Events



There are two types of events—notify and filter. **Notify** events tell LabVIEW that a user action has already occurred, such as when a user changes the value of a control. For example, you can write a VI that notifies the Event structure on the block diagram when the user clicks a button on the front panel. Because the value changes when the user clicks the button, the Event structure is notified. The Event structure can handle multiple notify events at one time. If you configure a single case to handle multiple notify events, only the data that is common to all handled event types is available. **Filter** events allow control over how the user interface behaves. With filter events, you can validate or change the event data before the user interface can process it, or you can discard the event data entirely, preventing the change from affecting the VI. For example, you can configure a Menu Selection event that does not allow the user to interactively close the front panel of the VI. You also can configure a Key Down event to modify all characters typed into a string control so they are uppercase, or you can modify the event to filter unwanted characters. The existence of terminals on the right side of the Event structure, such as the Discard terminal, identify a filter event.

Note If you configure the Event structure to handle menu events, and have the Get Menu Selection function configured for the same menu item, the Event structure takes precedence and LabVIEW ignores the Get Menu Selection function. In any given VI you should use either the Event structure or the Get Menu Selection function.

有两类事件。**通知**告诉 LabVIEW 一个用户动作已经发生。事件结构每次可以处理多个通知，这时只有对所有事件都常用的数据才可用。**过滤(事件名带有?)**允许用户控制用户界面行为。你可以在用户界面处理事件之前改变，允许或放弃事件数据。如果配置 Event structure 处理菜单事件，并将 Get Menu Selection function 函数配置给同一菜单项，LabVIEW 仅处理第一个忽略后面的。在 VI 中应当既使用 Event structure 又使用 Get Menu Selection function

Avoid Overusing Local and Global Variables



Local and global variables do not follow the general rules of dataflow programming, which can lead to race conditions and make it difficult to debug your program. In addition, local and global variables copy data during reads and do not reuse data buffers as subVI terminals do, which can lead to inefficient memory use. When you build subVIs, you create a connector pane that describes how data is passed to and from the subVI. In contrast, local and global variables can be written to or read from at any point in a VI—local variables from within the same VI and global variables from within all VIs currently running. In memory, these variables use data buffers that cannot always be reused. New data buffers are generated depending on how you use the local or global variable.

局部和全局变量不能重用内存，每次调用将分配新内存存放数据的副本。 可以从VI中任意点读或写，可能造成Race Condition.

Local or global variable cannot be read in place in memory without making copies, because the other process that is accessing the local or global variable could write to the buffer after you have read the value. This means that reading data from a local variable creates a new buffer for the data from its associated front panel object. If the data buffer for the extra copy is large, you will use much more memory than intended. As a result, you should avoid using local variables to read large data sets.

A global variable is similar to a subVI in the way you create them, with one important difference—the operating system can reuse the data buffers used in a subVI after the subVI finishes executing. Global variables always make a new copy of their data buffers whenever you read from the global. Avoid overusing global variables and consider using data management subVIs that minimize the number of block diagrams that manipulate the entire global variable data set.

You can avoid race conditions by using a subVI instead of a global or local variable. If you call the same subVI from two different places, by default, LabVIEW does not execute that subVI in parallel. Therefore, only one call to a subVI executes to completion, then the other call executes. You can avoid race conditions by placing all the operations that modify a global variable inside a single subVI. All operations to that global variable data are performed to completion before another call to the same subVI modifies the global variable data.

因为LabVIEW不会同时执行某个SubVI，并且SubVI可重用内存，所以用SubVI 代替 global or local variable可以提高效率并且避免Race Condition.

Data Exchange & Synchronization

You can use the Queue and Notifier functions to pass data between parallel block diagrams and pass references to subVIs. The internal routines in LabVIEW have been optimized, so no polling is necessary.

When working with different tasks in parallel, you might need to coordinate tasks to execute at the same time. This coordination is know as synchronization. If one task is dependent on one other task, you can create subVIs for each and wire them in a LabVIEW block diagram. Other techniques are necessary if you want to wait at some point for some other tasks before continuing.

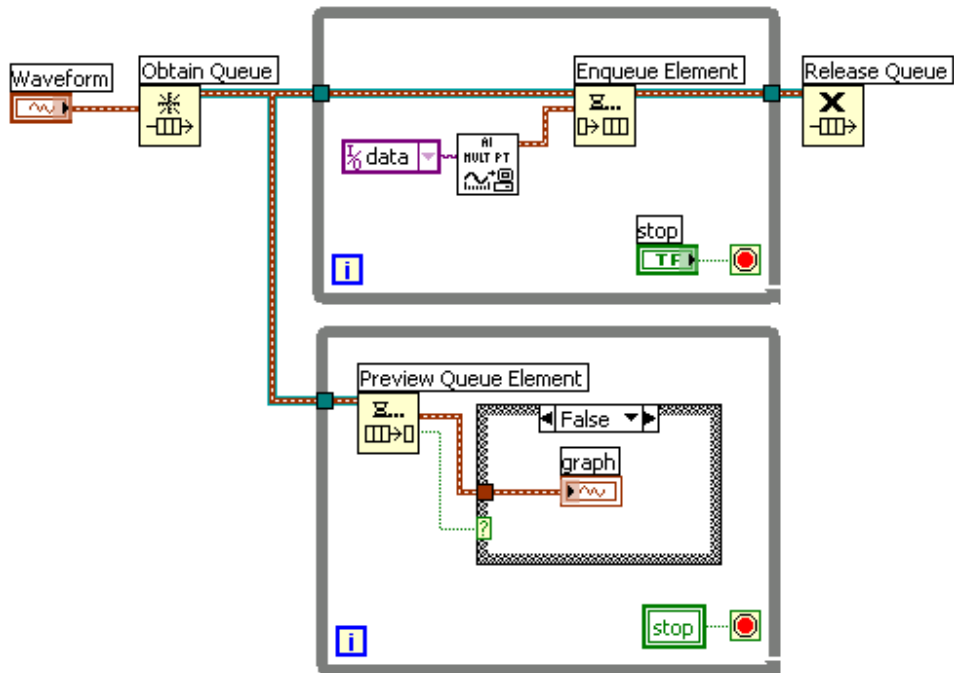
使用Queue and Notifier functions在两个平行框图中传递数据或向SubVIs中传递引用，可以避免轮循(Poll)。

使用global variables, notifiers(通知), queues(队列), semaphores(旗语), rendezvouses(集合点) and occurrences(发生).来同步各项任务。

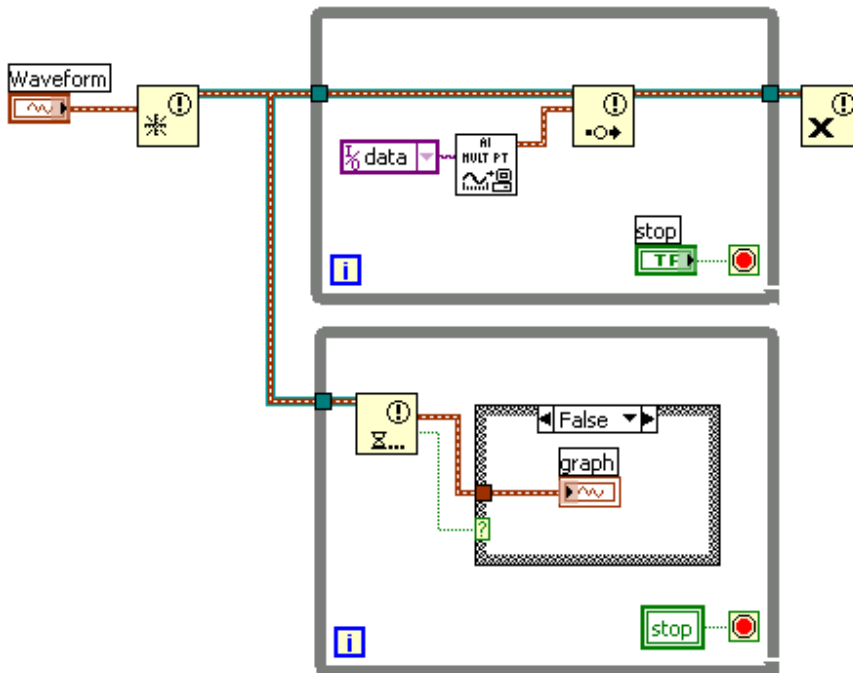
Occurrences较易使用，但不能避免陷入无限循环。Semaphores and rendezvous可以避免无限循环。



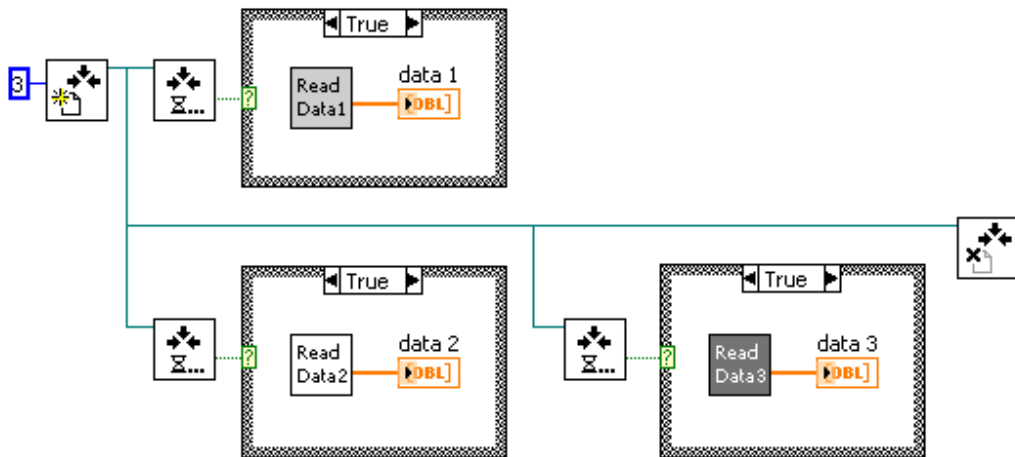
You use the **Queue** functions to store a set of data that can be passed between multiple loops running simultaneously or between VIs. Queues work best when one process reads data from the queue while multiple processes write data to the queue. When you create a queue, you specify whether the size of that queue is bounded and whether to remove data from the beginning or the end of the queue. Data is stored in string format, so you must flatten other data types to a string before being written to the queue. You can use queues to let parallel tasks know when some data has changed.



You use the **Notification** functions to enable one process in an application to tell one or more other processes to start running. The Notification VIs also store and send message data, so you can either send strings or flatten any other data types to strings to transfer data between parallel processes. The notification VIs can replace global or local variables because you can pass data back and forth between them. The notification VIs also remove the possibility of race conditions because each process waits until notification has been received before it executes.



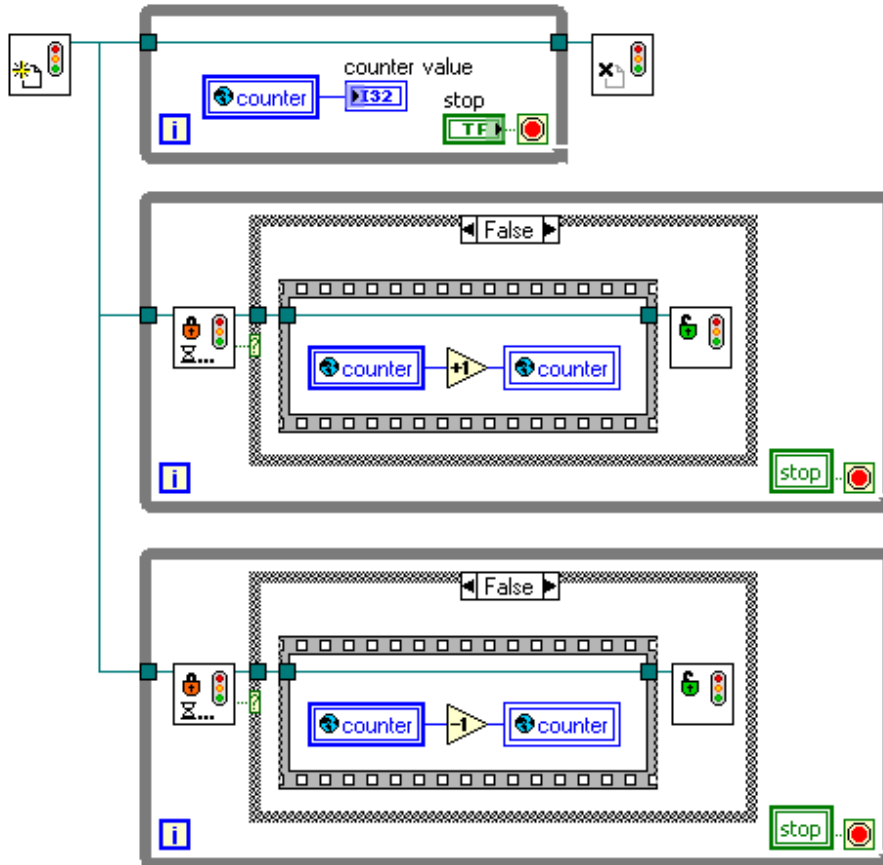
Note If several VIs send a notification to the same notifier or if a VI sends multiple notifications in a quick succession, the readers always receive the last notification sent.



You use **Rendezvous** to allow multiple tasks to synchronize with one another. When you have multiple tasks that run in parallel, you might want them to wait for each other at some point before proceeding. You use the Rendezvous VIs when you need multiple tasks to be synchronized at a certain place in the application. Each task that reaches a rendezvous waits until all other predetermined tasks reach the rendezvous point, at which point all the tasks continue execution. You use the Semaphore VIs to protect access to a global variable resource by allowing only one task at a time to access the global variable resource. The following are several advantages to using rendezvous to synchronize data.

- There is no polling involved.

- You can pass references to subVIs.
- You can abort infinite waits.

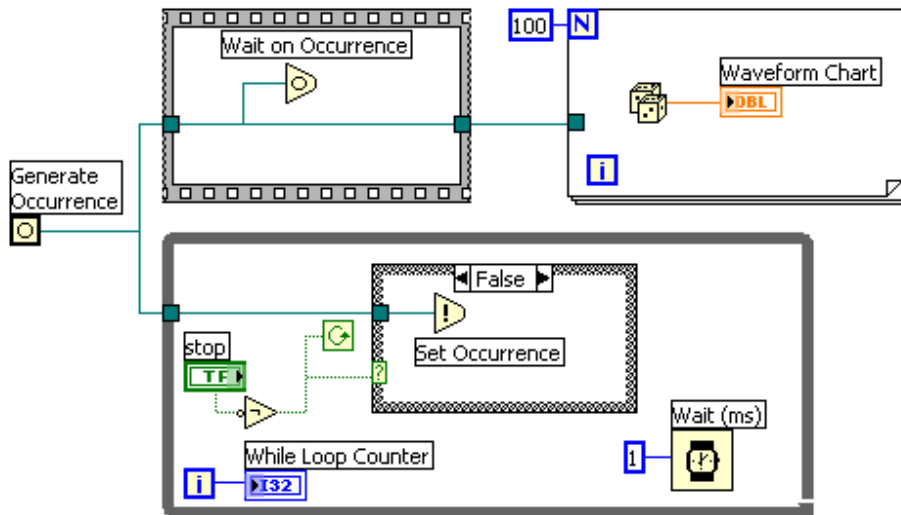


Use **Semaphores** to protect access to shared resources, such as groups of global variables, instruments, or files. In general, you can use semaphores to avoid race conditions associated with multiple parallel VIs modifying global variables or accessing a common instrument or writing to a common file. You use the Create Semaphore VI to create a semaphore, or mutex, and you define the number of tasks that can access the resource at any given time. Each time LabVIEW accesses the semaphore, that number is decremented. When the number is zero, no other task can access the global variable resource until another task has finished. Tasks use the Acquire Semaphore VI and Release Semaphore VI to gain and lose access to the global variable resource. The Destroy Semaphore VI deletes the semaphore when you no longer need to access the global variable resource. The following are several advantages to using semaphores to synchronize data.

- There is no polling involved.
- You can pass references to subVIs.
- You can abort infinite waits.



You use the **Occurrence** functions to put a portion of a block diagram to sleep while other tasks are running. The running tasks can call the Set Occurrence function at any time. Any loop or code that is using the **Wait on Occurrence** function will then execute. Occurrences are very similar to the Notification VIs, except Occurrences functions do not pass any data between themselves.



There are advantages and disadvantages to using occurrences to synchronize data.

- There is no polling involved.
- You can pass references to subVIs.
- It can be difficult to abort infinite waits.
- You cannot unset an occurrence.

Lesson 2 Improving Performance

Defining and Allocating Threads and Priorities in LabVIEW

Multitasking, multithreading, and multiprocessing are different technologies, but the terms for these technologies are often used interchangeably. Although LabVIEW automatically handles the multithreading of VI execution, you can modify the **Priority** and **Preferred Execution System** settings to control which threads are allocated to a VI. You can find these settings on the **Execution** page of the **File»VI Properties** dialog box. LabVIEW uses default settings for execution systems and threads, which eliminates the need for you to create, start, or stop threads and still leaves a large amount of flexibility. The priorities and execution systems shown in the **VI Properties»Execution** dialog box define the different threads available.

Note Subroutine priority VIs always use the execution system of their calling VI. Background priority VIs do not have a thread allocated to them and use the next higher priority threads when nothing else is available to run.

Note In a multiprocessor environment, thread allocation is dynamic and depends on which operating system you are using. Therefore, there is no straightforward way to determine which threads are running at any instant.

Monitoring VI Performance Using the Profile Window

You can use the **Profile** window to analyze how your application uses execution time and memory. The **Profile** window displays the performance information for all VIs in memory in an interactive table format. From the window, you can choose the type of information to gather and sort the information by category. You also can monitor subVI performance within different VIs. To show the **Profile** window, select **Tools»Advanced»Profile VIs**.

Many of the options in the **Profile** window are available only after you begin a profiling session. During a profiling session, you can take a snapshot of the available data and save the data to an ASCII spreadsheet file. The timing measurements accumulate each time you run a VI.

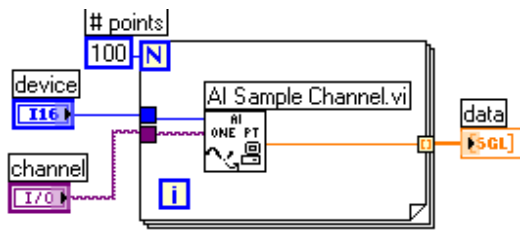
Note All statistics measured in a profiling session are collected for a complete run of a VI, not a partial run of a VI.

简介功能评价的是VI的整个运行情况，而非某一部分或某一阶段。

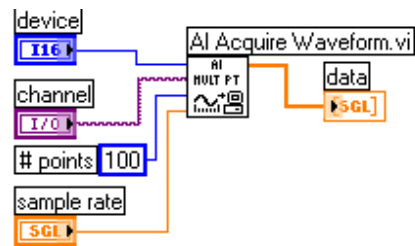
Speeding Up Your VIs----- Input/Output

Input/Output (I/O) calls generally take more time than a computational operation. For example, a simple serial port read operation can have an associated overhead of several milliseconds. This overhead is present not only in LabVIEW but also in other applications because an I/O call involves transferring information through several layers of an operating system. The best method of reducing this I/O overhead is to minimize the number of I/O calls you make in your application. Structure your application so that you transfer larger amounts of data with each call instead of making several I/O calls that transfer a small amount of data.

For example, consider creating a data acquisition (DAQ) application that acquires 100 points of data. For faster execution, use a multi-point data transfer function, such as the AI Acquire Waveform VI, instead of using a single-point data transfer function, such as the AI Sample Channel VI. To acquire 100 points, use the AI Acquire Waveform VI with an input specifying that you want 100 points. This is faster than using the AI Sample Channel VI in a loop with a wait function to establish the timing.



Single-Point Data Transfer (Slower Method)



Multiple-Point Data Transfer (Faster Method)

In the previous block diagram, overhead for the AI AcquireWaveform VI is roughly the same as the overhead for a single call to the AI Sample Channel VI, even though it is transferring much more data. In addition, the data collected by the AI Acquire Waveform VI uses hardware timers to control the sampling. Calling AI Sample Channel VI in a loop does not provide data collected at a constant sample rate.

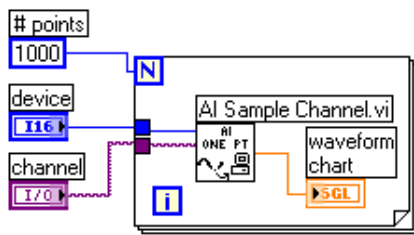
输入输出比运算操作消耗更多的时间。减少额外开支的方法就是每次尽可能传送更多的数据，从而减少I/O次数。

Screen Display

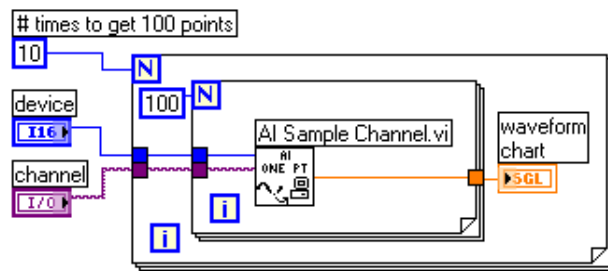
Updating controls on a front panel is another time-consuming task in an application. Although multithreading helps to reduce the effect that display updates have on overall execution time, complicated displays, such as graphs and charts, can adversely affect execution speed. This effect can become significant on Macintosh, which does not support multithreading. Although most indicators do not redraw when they receive data values that are the same as the old data, graphs and charts always redraw. To minimize this overhead, keep front panel displays simple and try to reduce the number of front panel objects. Disabling autoscaling, scale markers, and grids on graphs and charts improves their efficiency.

The design of subVIs also can reduce display overhead. If subVIs have front panels that remain closed during execution, none of the controls on the front panel can affect the overall VI execution speed.

As shown in the following block diagram, you can reduce display overhead by minimizing the number of times your VI updates the display. Drawing data on the screen is an I/O operation, similar to accessing a file or GPIB instrument. For example, you can update a waveform chart one point at a time or several points at a time. You get much higher data display rates if you collect your chart data into arrays so that you can display several points at a time. This way, you reduce the amount of I/O overhead required to update the indicator.



Single-Update Charting (Slower Method)



Multiple-Update Charting (Faster Method)

更新前面板的控件也是消耗时间的工作。每次更新尽量多的数据，减少更新次数，从而减少额外开销。

SubVI Overhead

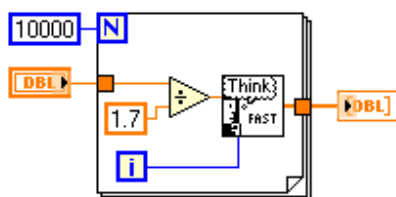
Each call to a subVI involves a certain amount of overhead. This overhead is fairly small—tens of microseconds—especially when compared to I/O overhead and display overhead, which can range from milliseconds to tens of milliseconds. However, if you make 10,000 calls to a subVI in a loop, the overhead could significantly affect your execution speed. In this case, consider embedding the loop in the subVI.

Another way to minimize subVI overhead is to turn your subVIs into subroutines by selecting **Execution** from the top pull-down menu in the **File»VI Properties** dialog box and then selecting **Subroutine** from the **Priority** pull-down menu. However, there are some trade-offs. Subroutines cannot display front panel data, call timing or dialog box functions, or multitask with other VIs. Subroutines are generally most appropriate for VIs that do not require user interaction and are short, frequently executed tasks.

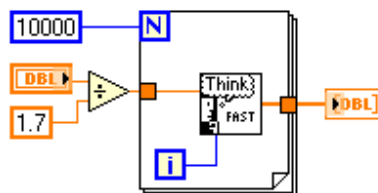
每次调用SubVI包含一定的相当小的开销，但如果大量调用，这些开销不能被忽略，可以将SubVI设置为子程序。这样一来，将不能显示前面板，不能调用时间和对话框函数，以及和其他VI进行多任务。一般适用于不包含用户界面，短小但频繁执行的任务。

Unnecessary Computation in Loops

Avoid placing calculations in loops if the calculation produces the same value for every iteration. Instead, move the calculation out of the loop and pass the result into the loop. For example, consider the following two block diagrams. The result of the division is the same every time through the loop. Therefore, you can increase performance by moving the division out of the loop.

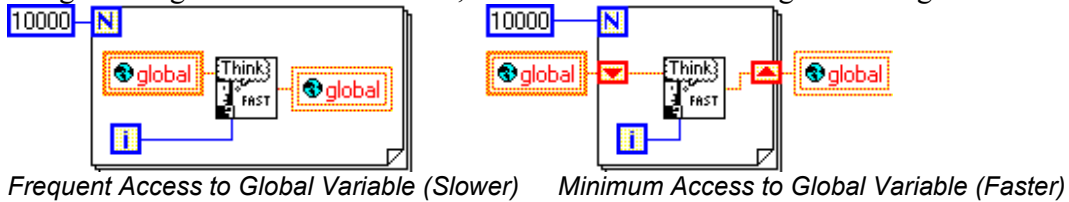


Unnecessary Computation in Loop (Slower)



Only Necessary Computation in Loop (Faster)

Also, avoid accessing local and global variables unnecessarily in loops. For example, the following first block diagram wastes time by reading from the global variable and writing to the global variable during each iteration of the loop. If you know that the global variable is not read from or written to by another block diagram during the loop, consider using shift registers to store the data, as shown in the following block diagrams.



Notice that you need the shift registers to pass the new value from the subVI to the next iteration of the loop. Beginning LabVIEW users commonly omit these shift registers. Without using a shift register, the results from the subVI are never passed back to the subVI as the new input value, as shown in the following block diagram.

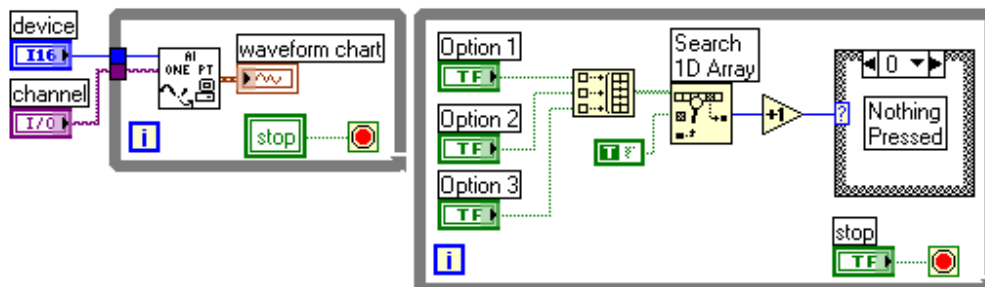


In the previous block diagram, the global variable is read once before the loop begins, and the same value is passed to the function 10,000 times. The result of the loop is the same as if you had written the code as shown in the following block diagram.

如果运算产生同一数值，那么把它放到循环之外。
另外，如果需要在循环内访问local and global variables并不断改变其值，考虑用 shift registers代替。

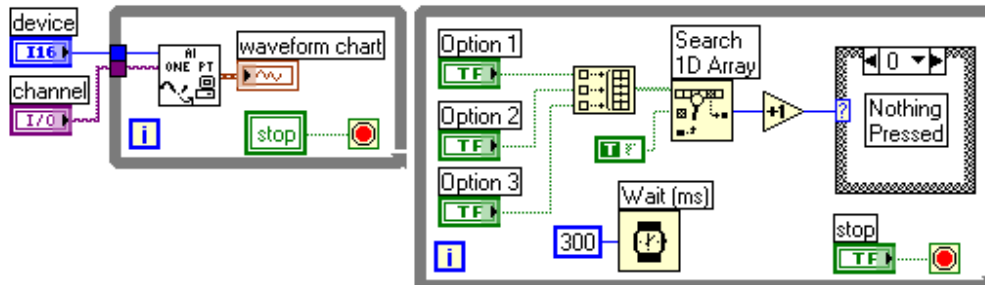
Parallel Diagrams

When several loops run in parallel on a block diagram, LabVIEW changes among the loops periodically. If some of these loops are less important than others, use the Wait (ms) function to ensure that the less important loops use less time. For example, consider the following block diagram.



The two loops run in parallel. One of the loops acquires data and must execute as frequently as possible. The other loop monitors user input. In the previous block diagram,

both loops get equal time. The loop monitoring the user input can execute several hundred times per second. In reality, this loop needs to execute only a few times per second because the user cannot make changes to the interface very quickly. You can call



the Wait (ms) function in the user interface loop to give significantly more execution time to the data acquisition loop, as shown in the following block diagram.

当框图中有多个平行的循环，LabVIEW将周期性的切换。如果某些循环不那么重要，使用

Wait (ms) function来减少它们所占用的时间。

Reentrancy and SubVI Memory Use

Another setting you should remember when you are concerned with memory use in a subVI is the **Reentrant Execution** option in the **VI Properties** window. You can use reentrancy to create a new data space for each instance of the VI. This means that if your data space is already very large, you will get a copy of it if you have reentrancy enabled.

The memory monitoring tools in LabVIEW do not report information on reentrant VIs. So you must keep track which subVIs have this feature enabled. Reentrancy is usually used when you are calling a subVI in different locations at the same time and that subVI is storing data in an uninitialized shift register between each call. Otherwise, you might not need to configure subVIs for reentrant execution.

再进入执行选项，它为SubVI的每个实例建立一个数据空间，用于在不同位置调用同一个SubVI，并且每个SubVI在已初始化的移位寄存器中保存了数据的时候。除此之外，不需要此选项。

Optimizing VI Memory Use

(Windows, Sun, and UNIX) LabVIEW allocates memory dynamically, taking as much as needed.

(Macintosh) LabVIEW allocates a single block of memory at launch time, out of which all subsequent allocations are performed. If LabVIEW runs out of memory, it cannot increase the size of this memory pool. Therefore, you should set this parameter as large as possible.

(Windows and Macintosh) You can use virtual memory to increase the amount of memory available for your applications. Virtual memory uses available disk space for

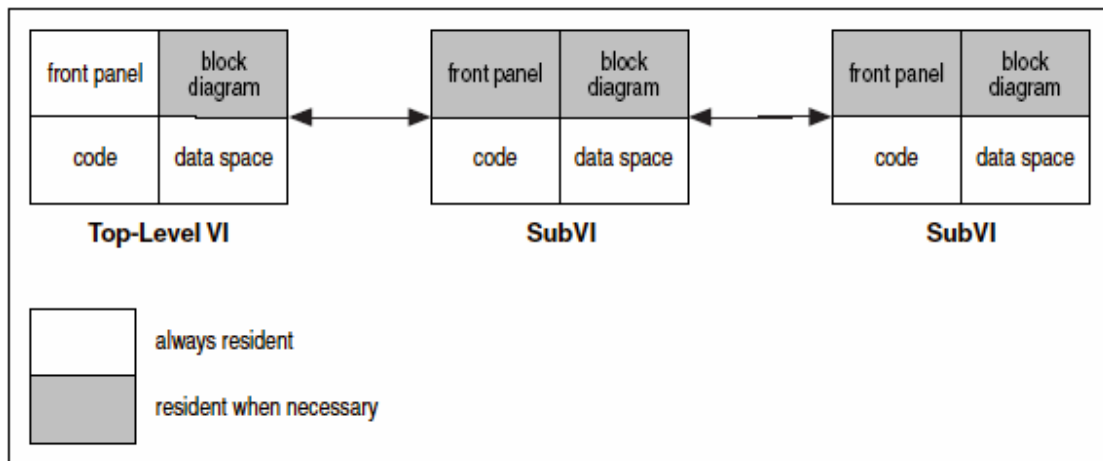
RAM storage. On the Macintosh, you allocate virtual memory using the Memory Control Panel. Windows, Sun, and UNIX automatically manage virtual memory allocation. LabVIEW does not differentiate between RAM and virtual memory. However, accessing data stored in virtual memory is much slower than accessing data stored in physical RAM. Virtual memory can help run larger applications, but it is probably not appropriate for applications that have critical time constraints.

VIs have the following major components:

- Front panel
- Block diagram
- Code—block diagram compiled to machine code
- Data—control and indicator values, default data, block diagram constant data, and so on.

When you load a VI, LabVIEW loads the front panel, the code, if it matches the platform, and the data for the VI into memory. If the VI needs to be recompiled because of a change in platform or in the interface to a subVI, LabVIEW also loads the block diagram into memory. LabVIEW also loads the code and data space of subVIs into memory. Under certain circumstances, LabVIEW also loads the front panel of some subVIs into memory. For example, if the subVI uses Property Nodes, LabVIEW must load the front panel because Property Nodes manipulate the characteristics of front panel controls.

As shown in the following figure, you can save memory by converting some of your VI components into subVIs. If you create a single, large VI with no subVIs, the front panel, code, and data for that top-level VI end up in memory. If the VI is broken into subVIs, the code for the top-level VI is smaller, and only the code and data of the subVIs are in memory. In some cases, you might actually see lower run-time memory use.



Note When monitoring VI memory use by selecting **File»VI Properties»Memory Usage**, be sure to save the VI before examining its memory requirements. The Undo feature makes temporary copies of objects and data, which can increase the reported memory requirements of a VI. Saving the VI purges the copies generated by Undo, resulting in accurate reports of memory information.

在使用**File»VI Properties»Memory Usage**监视内存使用情况前，先保存VI。因为Undo会带来对象和数据的临时副本，保存VI可以消除这些副本，从而提高内存信息报告的准确性。

Guidelines for Better Memory Use

Use the following guidelines to create VIs that use memory efficiently.

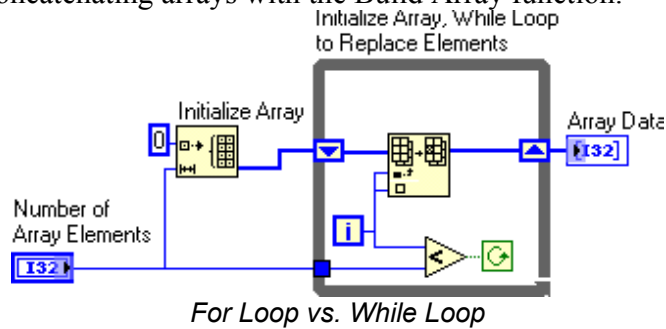
- Break VIs into subVIs wherever it is practical. This enables LabVIEW to reclaim subVI data memory when subVIs are not executing.
- Do not overuse global and local variables. Reading global or local variables generates copies of the data in the variable.
- On open front panels, display large arrays and strings only when necessary. Indicators on open front panels retain a copy of the data that they display.
- If the front panel of a subVI is not displayed, do not leave unused Property Nodes on the subVI. Property Nodes cause the front panel of a subVI to remain in memory, which increases memory use.
- When designing block diagrams, watch for places where the size of an input is different from the size of an output. For example, frequently increasing the size of an array or string using the Build Array or Concatenate Strings function, generates copies of data, which increases the number of memory allocations LabVIEW must perform. These operations can fragment memory.
- Use consistent data types for arrays and watch for coercion dots when passing data to subVIs and functions. When LabVIEW changes data types, the output is a new buffer.
- Do not use complicated, hierarchical data types, such as arrays of clusters containing large arrays, strings, or clusters containing large arrays or strings. Refer to the *Simple Versus Complicated Data Structures* section later in this lesson for more information about designing efficient data types.

- 将VIs分解为SubVIs,以便LabVIEW适时收回数据空间。
- 读取global or local variables前要先产生副本，所以不要过度使用。
- Indicators在显示时会在内存中保留副本，所以仅当必要才显示大数组和字符串。
- Property Nodes会导致SubVI的前面板留在内存中，增加内存开销，所以当前面板不显示时不要遗留未使用的Property Nodes在SubVI中。
- 注意输入和输出大小不同的地方，将产生副本，增加内存的分配，会造成内存碎片。例如，频繁使用Build Array或Concatenate Strings函数来增加array或string的大小。
- 为数组使用相同的内存类型。注意强制类型转换时，使用新的缓冲区作为输出。
- 不要使用复杂，多级的数据类型，比如包含大数组，字符串，簇的数组，或包含大数组，字符串的簇。

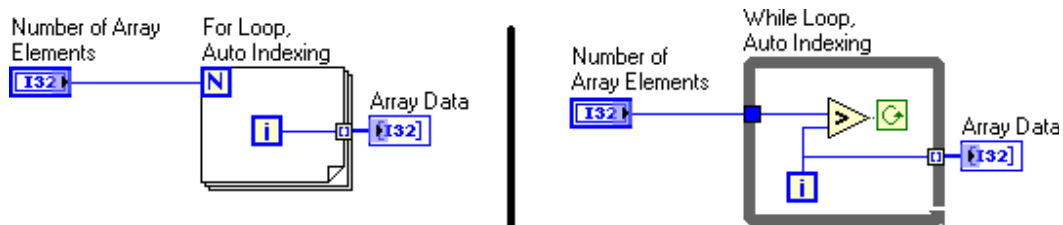
Assembling and Processing Arrays

LabVIEW stores arrays of numeric elements in contiguous blocks of memory. If you use For Loops to assemble these arrays, LabVIEW can determine the amount of memory

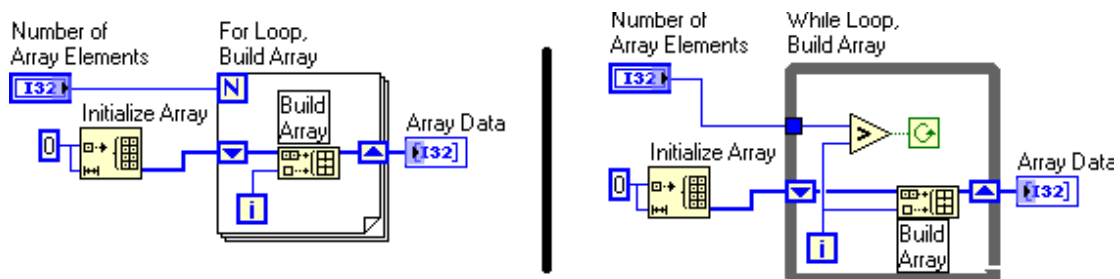
needed and allocate the necessary space before the first iteration. However, if you use While Loops, LabVIEW cannot predetermine the space requirements you will need. LabVIEW might need to relocate the entire buffer as the array grows in size. Each resizing requires the operating system to allocate a new buffer and then copy the contents of the old buffer into the new, larger buffer. The more resizing that occurs, the more time execution takes. The time needed for relocation increases with the size of the array. Therefore, you should use For Loops to assemble arrays when possible rather than using While Loops or concatenating arrays with the Build Array function.



If you can predetermine the maximum size of the array to be built, then you can initialize this array and pass it into the While Loop, as shown in the following block diagram. Once inside the While Loop, each element can be updated or replaced as needed. This is somewhat efficient because the array does not need to be reallocated once it is passed into the While Loop. This is still not as efficient as the For Loop or Initialize Array function.

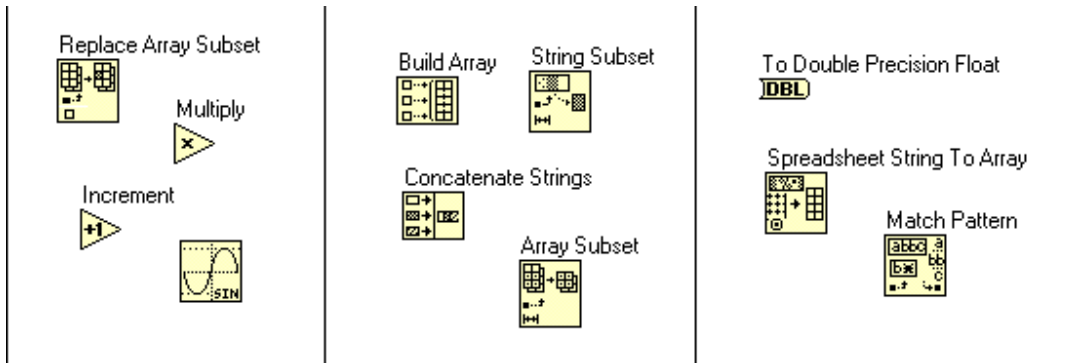


Avoid using the Build Array function inside a loop, as shown in the following figure. Every time a new value is appended to the array, LabVIEW must reallocate the buffer and copy the entire array to the new location. Thus, execution times for the Build Array function are the slowest.



Inplaceness

When possible, the LabVIEW compiler reuses a function's input buffers to store its



Functions that Reuse Buffers Functions that May Allocate New Buffers Functions that Allocate New Buffers

output. This buffer sharing is called **inplaceness**. A function output reuses an input buffer if the output and the input have the same data type, representation, and, in arrays, strings, and clusters, the same structure and number of elements. Functions capable of inplaceness include Trigonometric and Logarithmic functions, most Numeric functions, and some string and array functions, such as To Upper Case and Replace Array Element. A function that shifts or swaps elements of an array, such as Replace Array Element, can reuse buffers. Some functions, such as Array Subset and String Subset, might not copy data but might pass pointers to the sub arrays or substrings. In the following illustration, A, B, C, and D identify individual buffers. Build Array and Concatenate Strings are special functions. They operate in place when they can, but sometimes they must allocate new buffers.

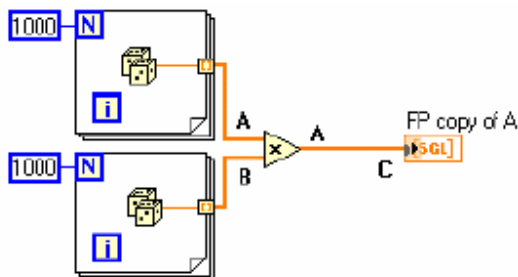
Coercion and Consistent Data Types

Of the following three methods to create these SGL arrays, one is correct and two are incorrect. Recall that each DBL value requires 8 bytes of memory, while each SGL value requires 4 bytes of memory.

下列三图建立单精度数组，两错一对。须知双精度需要8字节，而单精度需要4字节。

Method 1 (Incorrect)

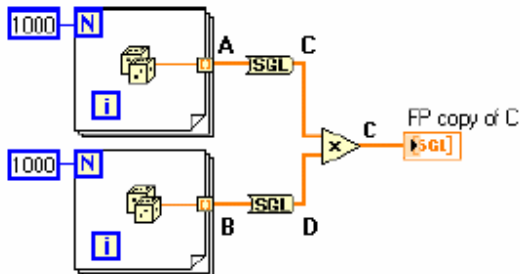
The following block diagram might be your first attempt to save data space memory. You might think changing the representation of the array on the front panel to SGL can save space memory. However, this method does not affect the amount of memory needed by the VI because the function creates a separate buffer to hold the converted data. The coercion dot on the SGL array terminal indicates the function created a separate buffer to hold the converted data.



Data 24.3KB

Method 2 (Incorrect)

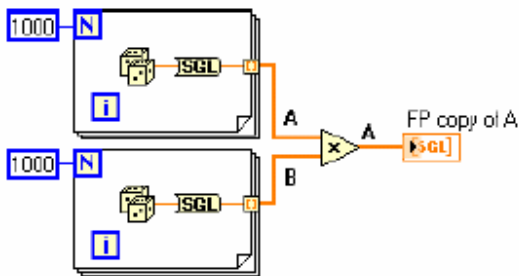
Method 2 is an attempt to remove the coercion dot by converting each array to SGL using the **To Single Precision Float** function located on the **Functions»Numeric»Conversion** palette. However, this method also increases the size of the data space because the function creates two new buffers, C and D, to hold the new SGL arrays.



Data 28.3KB

Method 3 (Correct)

Method 3 reduces the size of the data space considerably, from 28.3 KB to 12.7 KB. This method converts the Random Number (0-1) function output to SGL before the array is created. Therefore, this method creates two SGL arrays at the border of a For Loop rather than two DBL arrays.



Data 12.7KB

Simple versus Complicated Data Structures

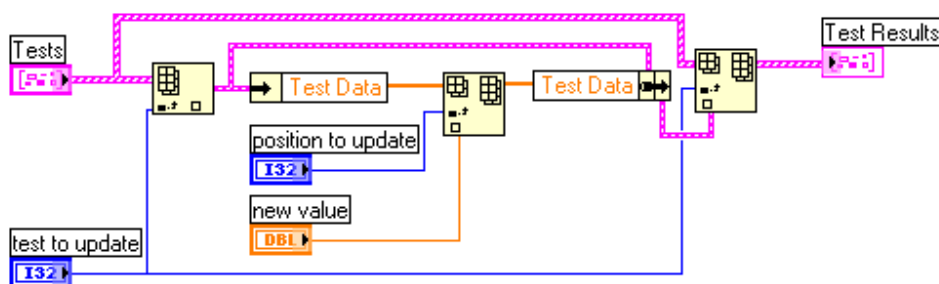
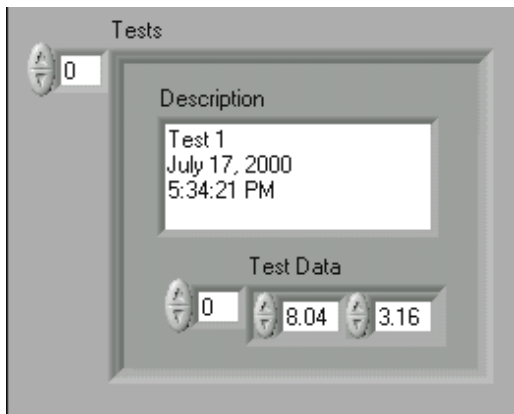
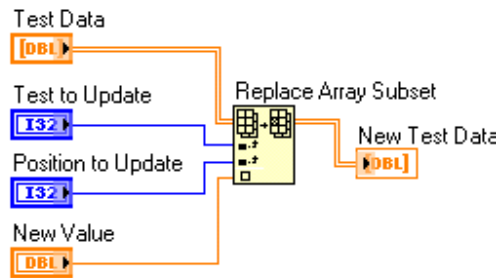
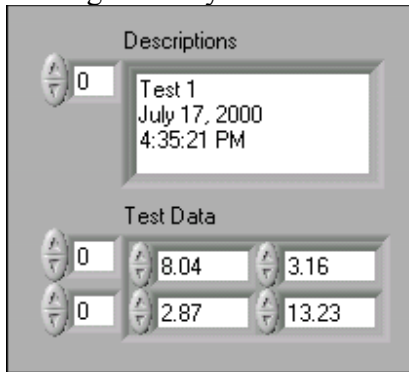
Simple data types, which include strings, numbers, Boolean data types, clusters of numbers or Boolean data types, and arrays of numbers or Boolean data types, are referenced in memory. Other data, referred to as nested, or complicated data, is more difficult to reference.

For the best performance, avoid creating complicated data structures. Performance can suffer because it is difficult to access and manipulate the interior elements without generating copies of data. Therefore, keep your data structures as flat as possible. Flat data structures can generally be manipulated easily and efficiently.

为了最佳性能，避免使用复杂的数据结构。因为难以访问和操作。

Consider an application in which you want to record the results of several tests. In the results, you want a string describing the test and an array of test results. One data type that you might use to store this information is an array of clusters containing a description string and an array of test data, as shown in the following front panel.

Now, consider what you need to do to change an element in the Test Data array. First, you must index an element of the overall Tests array. For that element, which is a cluster, you must unbundle the elements to get to the array. You then replace an element of the array and store the resulting array in the cluster. Finally, you store the resulting cluster in the original array.



An example of this is shown in the following block diagram. Copying data is costly both in terms of memory and performance. The solution is to make the data structures as flat as possible. In this case, you could store the data in two arrays, as shown in the following block diagram. The first array is an array of strings. The second array is a 2D array, where each row is a given test's results. In the following example, the front panel contains new controls to store the data, and the block diagram performs the same change to the test data, as shown in the previous block diagram.

Speed up VIs

- Local variables increase memory requirements and slow run speed, especially when they are accessed in loops.
- Create arrays in an efficient manner. When generating arrays inside loops whose representation must be changed, change the representation inside the loop, not after.
- Minimize the amount of file I/O operations performed in an application. Open and close a file only when necessary.
- Avoid unnecessary computations and data conversions, especially in loops.
- Minimize and simplify front panel displays.
- Use simple data structures that are as flat as possible.
- Avoid using autoscaling on graphs and charts.
- Update graphs and charts several points at a time, not one at a time.
- Force less important parallel tasks to wait, using the **Wait (ms)** function, so more crucial ones have more processor time.

- Local variables增加所需的内存，降低运行速度，特别是在循环中访问。
- 用高效的方式建立数组。当循环中产生的数组的数据类型必须改变，那么在循环内改变，而不是在循环之后。
- 尽量减少文件读写操作，仅保留必要的。
- 避免不必要的计算和数据类型转换，特别是在循环中。
- 减少和简单化前面板的显示。
- 使用简单的，尽可能单调的数据结构。
- 避免使用graphs和charts的自动标度(autoscaling)功能。
- 每次更新graphs和charts上的多个点，而不是每次一个。
- 用Wait(ms)函数强迫不重要的平行任务等待，这样重要的任务可以获得更多的处理器时间。

Lesson 3 TCP/IP

Transmission Control Protocol (TCP) and the Internet Protocol (IP). TCP and IP are the basic tools for network communication. TCP is usually the best choice for network communication between application, because it ensures all data is delivered to its destination. TCP is the protocol supported by all the different operating systems. The Address Resolution Protocol (ARP) maps Ethernet to TCP/IP internet addresses and was incorporated into TCP/IP in 1982. The Internet Protocol (IP) transmits data across the network. This low-level protocol takes data of a limited size and sends it as a datagram across the network. IP is rarely used directly by applications because it does not guarantee that the data will arrive at the other end. Also, when you send several datagrams, they sometimes arrive out of order or are delivered multiple times, depending on how the network transfer occurs.

Layer	Protocol
Application Presentation Session	SMTP (Simple Mail Transfer Protocol) FTP (File Transfer Protocol) Telnet
Transport	TCP (Transmission Control Protocol)
Network	IP (Internet Protocol) ARP (Address Resolution Protocol)
Datalink (HW)	Ethernet

TCP uses IP to transfer data. TCP is connection-oriented. It establishes a session between the user processes, breaks data into components that IP can manage, encapsulates the information, transmits the datagrams, and tracks the datagram progress. Lost datagrams are retransmitted, and data arrives in order and without duplication. For these reasons, TCP is usually the best choice for network applications. The IP protocol, on which TCP is based, is responsible for error checking. Hence, both the protocols support error checking, and a data packet is not delivered unless it passes error checks.

TCP是面向连接的，利用IP传输数据。两个协议都支持错误检测，数据包必须经过错误检测后才能通过。

To establish a TCP connection, you must specify both the address and the port number. The address identifies a computer on the network and can be expressed in IP dot notation or as the hostname. The port is an additional number that identifies a communication channel on the computer that the server uses to listen for communication requests, which is between 0 and 65535. Port numbers less than 1024 are reserved for certain applications. For example, port 80 is reserved for HTTP. Different ports at an address identify different services at that address, making it easier to manage multiple simultaneous connections.

The port numbers are broadly classified as follows:

0 – 1023 Well-known ports

1024 – 49151 Registered ports, including DSTP (3015). You should not arbitrarily pick ports from this range.

49152 – 65535 Private or dynamic ports.

建立TCP连接，必须指定地址和端口号。端口号在0-65535之间，小于1024的保留给某些应用程序。1024-49151是已注册的端口，不要随意用这些端口。49152-65535是私人或动态端口。

The IP address might have the following format:

Class A 0 <7-bit network number> <24-bit host number>

Class B 10 <14-bit network number> <16-bit host number>

Class C 110 <21-bit network number> <8-bit host number>

Class D 1110 <number>

Class E 1111 <number>

Class D is used to support IP multicasting, and Class E is primarily reserved for experimental use.

D类用来支持IP多播，E类主要保留给实验用途。

Client/Server Model

The client/server model is a common model for networked applications. In this model, one set of processes (*clients*) requests services from another set of processes (*servers*).

Clients:

1. Opens a connection to a server.
2. It sends commands to the server.
3. It receives responses from the server.
4. Repeat 2~3 to process more commands.
4. Finally, it closes the connection and reports any errors that occurred during the communication process.

Servers:

1. Initializes the server.
2. If the initialization is successful, enters a loop that waits for a connection.
3. Once the connection is made, waits to receive a command.
4. Executes the command and returns the results. It might send back a response indicating that a command is invalid, but it does not display a dialog when an error occurs.
5. Closes the connection and shut down after having received a command to end.
6. Write a log of transactions and errors to a file or a string.



You use the **TCP Open Connection** function on the *client* computer to open a connection to a server using the specified Internet address and port for the server. If the connection is not established in the specified timeout period, the function completes and returns an error. **connection ID** is a network connection refnum that uniquely identifies the TCP connection. **error in** and **error out** clusters describe any error conditions.

A *server* needs the ability to wait for an incoming connection. The procedure is to create a listener and wait for an accepted TCP connection at a specified port.



First, use the **TCP Create Listener** function to create a listener on a computer to act as a server.



Then use the **Wait on Listener** function to listen for and accept new connections. The advantage of using this method is that you can cancel a listen operation by calling **TCP Close Connection**. This is useful when you want to listen for a connection without using a timeout, but you want to cancel the listen when some other condition becomes true. For example, when the user clicks a button. When a connection is established, you can read and write data to the remote application using the functions explained in the following sections.



The **TCP Read** function receives up to the number of bytes specified by **bytes to read** from the specified **TCP connection ID** and returns the results in **data out**. If the operation is not complete in the specified **timeout** period, the function completes and returns an error. **error in** and **error out** clusters describe any error conditions. The **mode** input specifies the behavior of the read operation for the following four different options:

- **Standard**—If you use the standard mode, the function returns the number of bytes received so far. If less than the requested number of bytes arrive, it reports a timeout error.
- **Buffered**—If you use the buffered mode, the function returns the number of bytes requested or none. If less than the requested number of bytes arrive, it reports a timeout error.
- **CRLF**—If you use the CRLF mode, the function returns the bytes read up to and including the CR (carriage return) and LF (line feed) or nothing. If a CR or LF are not found, it reports a timeout error.
- **Immediate**—If you use the Immediate mode, the function waits until any bytes are received. This function waits the full timeout if no bytes are received.



The **TCP Write** function writes the string **data in** to the specified **TCP connection ID**. If the operation is not complete in the specified **timeout** period, the function returns an error. **bytes written** identifies the number of bytes transferred. **error in** and **error out** clusters describe any error conditions.

Notice that all the data written or read is in a string data type. The TCP/IP protocol does not state the type or format of the data transferred, so a string type is the most flexible method. You can use the Type Cast and Flatten to String functions to send binary or complicated data types.

However, you must inform the receiver of this information of the exact type and representation of the data to reconstruct the original information. Also, when you use the TCP Read function, you must specify the number of bytes to read. A common method of handling this is to send a 32-bit integer first to specify the length of the data string that follows. The TCP example VIs provided with LabVIEW and the exercises in this lesson provide more information on these topics and on how the data typically is formatted for TCP/IP communications.



The **TCP Close Connection** function closes the connection to the application associated with **connection ID**. **abort** determines whether LabVIEW closes the connection normally or aborts the connection. **error in** and **error out** clusters describe any error conditions.

If there is unread data and the connection closes, the unread data might be lost. This behavior is dependent on your operating system. For example, on the Sun operating system, unread data is kept even after the remote application closes the connection. However, Windows NT immediately deletes any unread data when a close connection is received. Connected parties should use a higher level protocol to determine when to close the connection. Once a connection is closed, you can not read or write from it again.

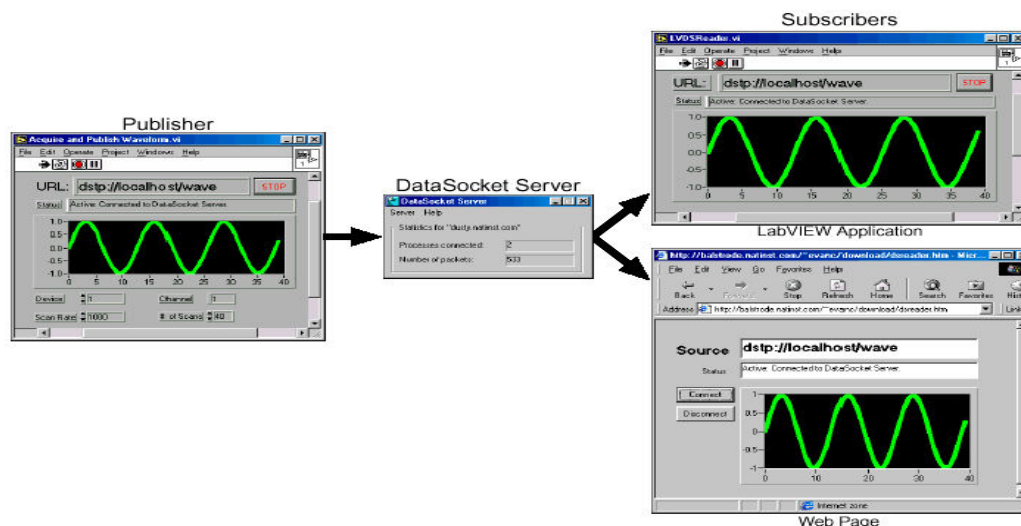
Lesson 4 DataSocket

译者注：DataSocket虽然易用，但有可能丢失数据，而TCP/IP方式不会丢失。视需要而选择。

DataSocket is a programming tool that enables you to read, write, and share data between applications and/or different data sources and targets across a network. You can use DataSocket to access data in local files and data on HTTP and FTP servers. With general purpose file I/O functions, TCP/IP functions, and FTP/HTTP requests to transfer data, you must write separate code for each protocol. With DataSocket, you need very little or no code to transmit and receive data over the Internet.

DataSocket consists of the following four components:

- **DataSocket Server**—The DataSocket Server acts much like a Web server and continuously listens on port 3015 for client requests. After the client request arrives, the DataSocket Server checks if the client is allowed to access the server. If the client is allowed access, it processes the read/write request. You must have the DataSocket Server running for the clients to connect to the server for read/write operations.
- **DataSocket Publisher**—The DataSocket Publisher writes data to the DataSocket Server. The publisher sends the data to the DataSocket Server, which publishes that data to any clients that want to subscribe to that data.
- **DataSocket Subscriber**—The DataSocket Subscriber reads data the DataSocket publisher writes to the DataSocket Server.
- **Data Item**—The data item is the unique name in the DataSocket URL that identifies the data you want to read or write from. Data items can carry only one type of data or many types of data bundled together.



DataSocket Transfer Protocol (DSTP)

The URL (Uniform Resource Locator) for DataSocket is similar to the URL you normally use for Web pages. It identifies the location of the data that you want to write to or read from. The first part of the URL identifies the computer name or IP address, and the last part identifies the data item to which you want to read from or write to.

Note You cannot directly enter DataSocket URLs in most Web browsers. The IP address in the DataSocket URL always corresponds to the computer running the DataSocket server.

不能在大多数浏览器直接键入DataSocket地址，两个都运行了DataSocket的计算机才能通讯。

`dstp://127.0.0.1/wave`
└──┬──┬──┘
Protocol IP Address Data Item Name

`dstp://dsmachine.com/wave`
└──┬──┬──┘
Protocol Machine Name Data Item Name



You use **DataSocket Write VI** to publish data to the DataSocket connection.

- **URL**—Specifies the computer name and the data item to which you want to publish (write) data.

- **Data**—This input is polymorphic, so you can connect any data type to it.



You use the **DataSocket Read VI** to subscribe to data from the DataSocket connection.

- **URL**—Specifies the computer name and the data item from which you want to read.

- **type(Variant)**—This terminal is polymorphic and determines the data type that is expected from that data item. You can connect a dummy variable of the expected data type to the **type(Variant)**, and the output **data** terminal changes accordingly.

The DataSocket Server Manager allows you to combine hosts into groups, specify the group or hosts allowed to subscribe to data items from the DataSocket Server, and specify the group or hosts allowed to publish data items to the DataSocket Server.

In the DataSocket Server Manager, Server Settings has the following two setting that you can configure:

- **MaxConnection**—The maximum number of clients that can connect to the DataSocket Server at one time. The default value is 50.

- **MaxItems**—The maximum number of data items that are available in the DataSocket Server. The default value is 200. In the DataSocket Server Manager, Permission Groups defines security using the following three built-in groups:

- **DefaultReaders**—The hosts that can read data items from the DataSocket Server. The default is `everyhost`, which means any host can read a data item.

- **DefaultWriters**—The hosts that can write to a data item on the DataSocket Server. The default is `localhost`, which means that only the host running the DataSocket Server can write data to a data item. You must change this if you want other clients to publish (write) data using the server.

- **Creators**—The hosts allowed to create data items. The default is `localhost`, which means that only the host running the DataSocket Server can create data items. A host can

belong to writer group but can not belong to creators group.

You also can create custom groups by clicking **New Group** and assigning a name to the group. After you assign the name to a group, add the hosts to the group by specifying their name or IP address. When you create groups, you combine hosts that belong to a certain group. They do not define the actual permission assigned to the group.

A data item is a unique name or place in the DataSocket Server from which you read or write data. You can create a data item dynamically by the client at runtime if it belongs to the creators group, or you can create the data items in the DataSocket Server Manager ahead of time. Data items created ahead of time in the DataSocket Server Manager are called predefined data items. To create a new data item, click **New Item** and enter a name for the data item. You can then specify the group that has access to read from the data item and the group that has access to write data to the data item. You also can allow multiple hosts to write data to the data item at a time and specify the initial value. The DataSocket Server returns the initial value to a reader if no one is writing data to the data item.

Note Every time you make changes to the Data Socket Server Manager, you must restart the DataSocket Server for the changes to take effect.

每次对于Data Socket Server Manager的修改后，必须重启它以生效。

Front Panel DataSocket

Front panel DataSocket allows you to publish and subscribe to data directly from front panel controls and indicators.

Once you publish the data from a control or indicator, the data is transmitted when the VI runs. You can write to an existing data item, or a new one is dynamically created if it does not already exist.

On the publisher side, LabVIEW publishes the data using the front panel DataSocket. Once you run the VI, an LED indicator on the top right corner of the control indicates the DataSocket connection status. The green color of connection status indicates successful connection.

On the subscriber side, all you have to do is subscribe to data from the front panel indicator and keep the VI running. The indicator displays the data from the data item.

On the publisher side, LabVIEW publishes the data using the front panel DataSocket. Once you run the VI, an LED indicator on the top right corner of the control indicates the DataSocket connection status. The green color of connection status indicates successful connection.

On the subscriber side, all you have to do is subscribe to data from the front panel indicator and keep the VI running. The indicator displays the data from the data item.

Variant Functions



Each attribute is set using the Set Variant Attribute function. The attribute is set as a name value pair. The name of the attribute is unique, and you use it to retrieve the data from the attribute.

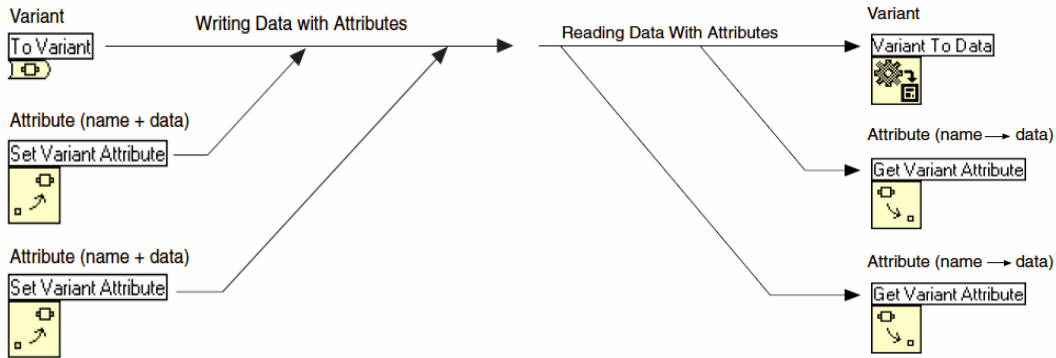


To read the attribute you need to specify the name of the attribute and the

expected data type of that attribute. You can specify the data type of the attribute by wiring the appropriate data type constant to the **default value** input of Get Variant Attribute function to determine the type of data expected from that attribute.

Note Attribute names are case sensitive.

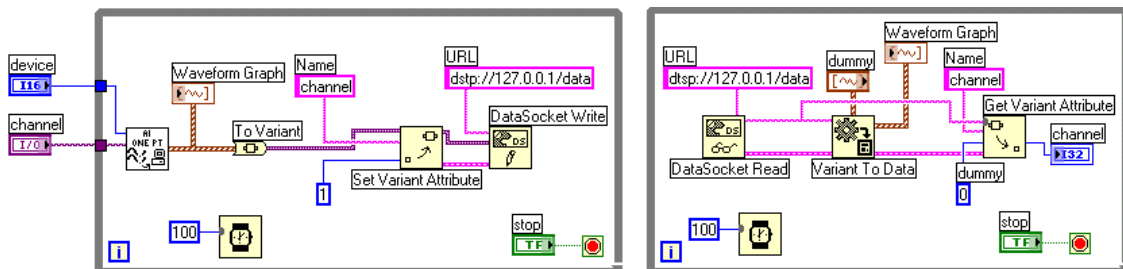
This section describes how to send multiple data types bonded together over one DataSocket connection. The data is bonded together using the Variant functions. OLE Variant data can be of any type, so you can combine several data types together as one data type, even ActiveX.



Data are first converted to variant data using the To Variant function and then each additional piece of data is written as an attribute. Each attribute is added as a name-data pair using the Set Variant Attribute function. The unique name of each attribute differentiates a particular attribute and retrieves the attribute data on the reader side.

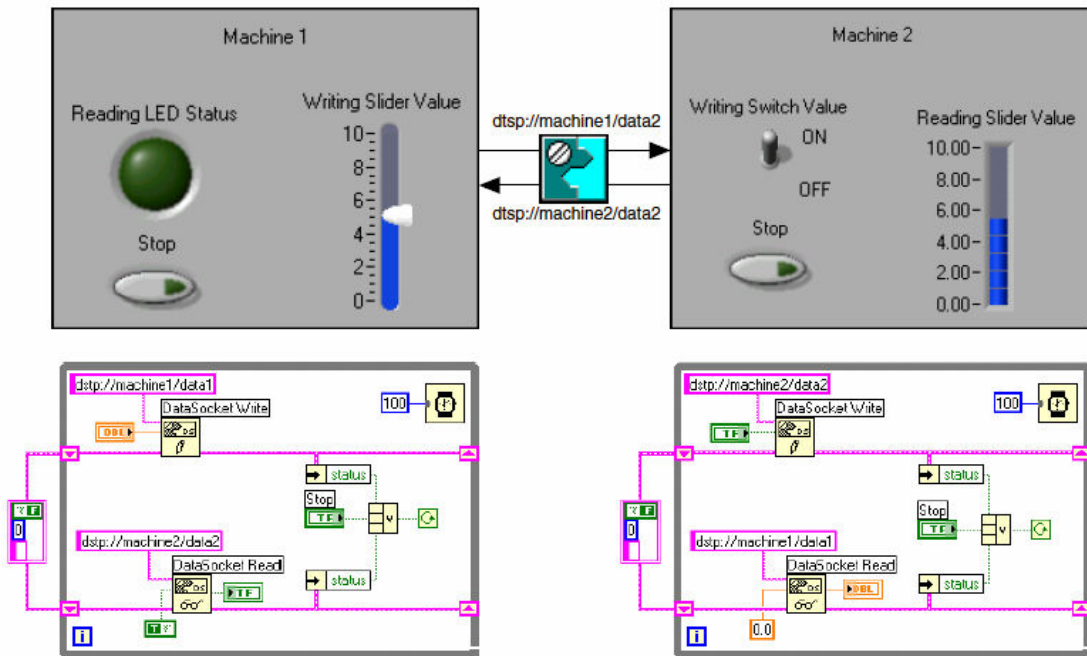
The primary reason for writing data with attributes is that it ensures that all pieces of information arrive at the same time on the reader side. It is also more efficient than sending multiple items on different DataSocket connections where you do not know if they will arrive at the same time. If different ones are lost, you do not know which ones are associated with one another. With attributes, the packets can still get lost, but what gets through is logically bonded.

The secondary reason is that you can dynamically delete an attribute, thereby saving bandwidth.



Bi-directional Communication

Bi-directional communication is when two VIs can communicate both ways over the network. You can use bi-directional communication to control VIs remotely over the Internet.



Reading and Writing Data from Other Sources

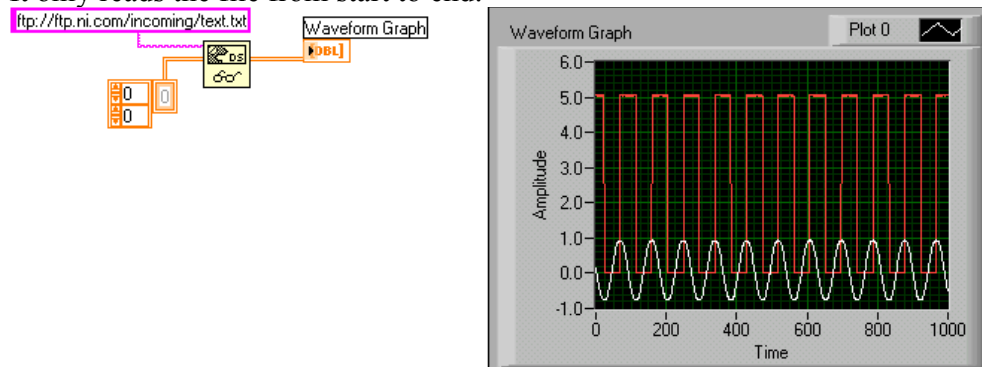
You can use the **DataSocket Read** function to read the following sources:

- Data on HTTP servers (Windows only)
- Data on FTP servers (Windows only)
- Local files (TXT, WAV, tab delimited TXT files)
- Data on OPC servers

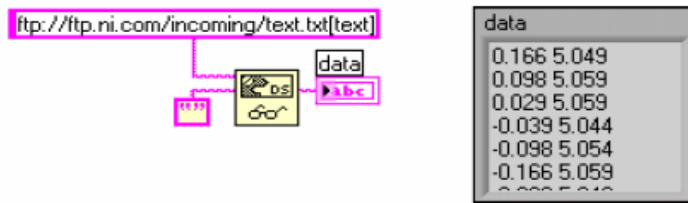
The files are read as ASCII text.

The **DataSocket Read** function reads the entire file. You cannot read small chunks of data from the file at a time or randomly access data in a file.

It only reads the file from start to end.



To read the text from the tab delimited file or a text file, you have to override the default by specifying `[text]` at the end of your file path.



Lesson 5

VI Server

VI Server is an object-oriented, platform-independent technology that provides programmatic access to LabVIEW and LabVIEW applications. It allows you to perform the following operations:

- Call a VI remotely.
- Configure LabVIEW on a remote computer to be a server that exports VIs that you can call from another computer on a network. For example, you might have a remote computer that acquires data on a local computer. By setting the preferences in LabVIEW, you make VIs on the remote computer accessible on the network so transferring the data is as easy as calling a subVI.
- Edit the properties of a VI and LabVIEW. For example, you can determine the position of a window or scroll a front panel so that a part of it is visible. You also can save any changes to disk.
- Change the properties of multiple VIs.
- Load VIs into memory dynamically, rather than having them statically linked into your application, and call them just like a normal subVI call using VI Server functions. This is useful if you have a large application and want to save memory or startup time.
- Create a plug-in architecture for your application to add functionality to your application after it is distributed to customers. For example, you might have a set of data filtering VIs, all of which take the same parameters. By designing your application to dynamically load these filters from a plug-in directory, you can ship your application with a partial set of these filters and make more filtering options available to users by simply placing the new filtering VIs in the plug-in directory.
- Retrieve information about an instance of LabVIEW, such as version number and edition. You also can retrieve environment information, such as the platform on which LabVIEW is running. For example, you can use this capability if you have an application that uses ActiveX and there is the possibility a user might try to run it on a non-Windows platform. Your application could check if the operating system is Windows and if not generate an error message.

VI Server是一种面向对象，与平台无关的技术，提供对于LabVIEW及应用程序的编程权限。允许完成以下的操作：

- 远程调用VI
- 将远程计算机上的LabVIEW配置为服务器，上面的VI可以被网络上的另一台计算机调用。例如有一台远程计算机在本地计算机上采集数据。通过设置LabVIEW属性，使远程计算机上的VI可以访问，这样传输数据如同调用SubVI一样简单。
- 编辑VI及LabVIEW的特性。例如，可以决定窗口的位置或滚动前面板，这样使部分可见。还可以把这些改变存盘。
- 改变若干VI的特性。
- 将VI动态装入内存，利用VI Server函数像普通VI那样调用，而不仅是静态的连接到应用程序上。这样当程序很大，需要节省内存或缩短启动时间时很有用处。
- 建立插件式结构，以便将应用程序发布给用户后可以增加功能。例如，你有一套参数相同的数据滤波器VI,通过设计应用程序从插件目录中动态调用这些滤波器，可以使程序运行一部分滤波器，而且通过简单地把新滤波器放入插件目录，能够提供更多的滤波器供用户选择。
- 得到LabVIEW实例的信息，如版本号。还可以得到环境信息，如LabVIEW当前的运行平台。例如，当你的程序运用了ActiveX而某个用户尝试将程序运行在非Windows平台，它可以检测操作系统是否是Windows,如果不是就产生错误信息。

Object-Oriented Terminology

A class defines what an object is able to do, what operations it can perform (methods), what data it contains, and what properties it has. Object-oriented programming is based on objects. An object is a instance of a class(type). Objects can have methods and properties. Methods are equivalent to functions, in that they perform an operation. Properties are the attributes of an object.

VI Server Clients

VI Server has a set of methods and properties that are accessible through the different clients.

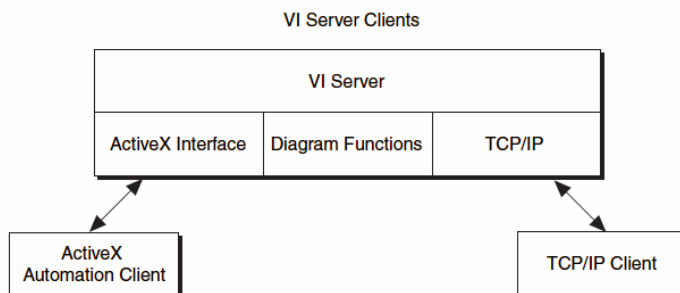


Diagram Access—LabVIEW has a set of built-in functions located in the **Functions»Application Control** palette you can use to access VI Server on a local or remote computer.

Network Access—If you are using VI Server on a remote computer, LabVIEW uses the TCP/IP protocol as the means of communication.

ActiveX Interface—ActiveX clients such as Visual Basic, Visual C++, and Excel applications can use VI Server to run LabVIEW applications.

Application and VI Objects

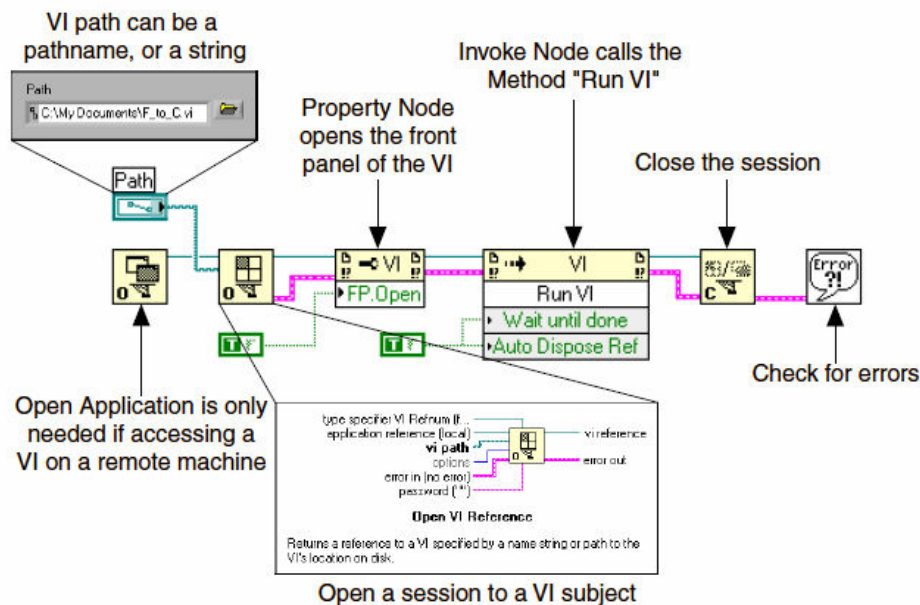
VI Server is exposed through references to two classes of objects—the Application object and the VI object. These classes of objects allow you to perform operations using **methods** and **properties**. An application class reference refers to a local or remote LabVIEW environment. This is the actual LabVIEW application itself, not a program written using LabVIEW. The **properties** and **methods** of the LabVIEW application object can, for example, change LabVIEW preferences and return system information.

A virtual instrument class reference refers to a specific VI running in the LabVIEW application. For example, the **properties** and **methods** of the VI object can change the VI's execution and window options.

VI Server通过两种对象产生作用:

- 应用程序对象涉及本地或远程 LabVIEW 环境。这是实际的 LabVIEW 应用程序本身，而非用 LabVIEW 编写的程序。例如，它的**属性**和**方法**可以改变 LabVIEW 的首选项和返回系统信息。
- 虚拟仪器对象涉及一个运行在 LabVIEW 应用程序中的特定的 VI。例如，它的**属性**和**方法**可以改变 VI 的运行和窗口选项。

VI Server Programming



First, you create a reference to either the Application object or the VI object using the Open Application Reference or Open VI Reference functions. This ensures that the correct resources are allocated and that they are reserved.

Then, pass the reference to the Property Node or Invoke Node functions, so that they know which application or VI they are operating on. Finally, close the reference to release the object resources.



When you open a reference to the Application object using the **Open Application Reference** function, the reference is either to LabVIEW running on the local computer or to a LabVIEW installation running a remote computer across a network. If you leave the computer name input unwired or empty, the function creates a reference to the current application. Otherwise, LabVIEW treats the computer name as a TCP/IP address and can be in dotted notation, such as 123.23.45.100, or domain name notation, such as remotemachine.ni.com.

The port number input allows you to specify multiple servers on a single computer.

You use the application reference output as an input to **Property Nodes** and **Invoke Nodes** to get or set properties and invoke methods on the LabVIEW application. You also use the application reference as the input to the **Open VI Reference** function to get references to VIs that are running in the LabVIEW application.



The **Open VI Reference** function returns a reference to a VI specified by a name string or path to the VI location on disk.

You can get references to VIs in another LabVIEW application by wiring an application reference, obtained from the Open Application Reference function, to this function. If you leave the application reference input unwired, the **Open VI Reference** refers to the local installation of LabVIEW.



You use **Property Nodes** to modify properties of the LabVIEW application or VI Object you define using the **Open Application Reference** or **Open VI Reference** function. Once you wire the VI reference to the **Property Node**, you can access all the properties available for that Application or VI reference.

You can resize **Property Nodes** to have more than one input or output. To select whether a property reads or write data, right-click the property node and select **Change to Read/Write**. If a property is read-only, this menu item is dimmed.

Property Nodes execute from the top to the bottom. If an error occurs midway down the node, the remaining properties are ignored, and an error is returned.

属性节点从上至下执行，当其中某个产生错误，其余的属性被忽略并返回错误。

Most Application Class properties are read-only. They allow you to check a whole range of parameters, such as what VIs are loaded into memory, properties of all the display monitors, the name of the operating system, the version of operating system, whether the LabVIEW Web Server is active, and so on. The properties that can be written are for the printing options. This allows you to customize what is and is not printed. You can use this information to ensure that the PC running the LabVIEW program is configured correctly and prompt the user to re-configure the PC if necessary.

Many of the properties of the VI that are exposed correspond to the properties available in the **File»VI Properties** dialog box. Most of the properties are read and write. Properties such as name, path, type, metrics, and so on are read-only. Some properties are transient, such as window position, title, and so on.



You use **Invoke Nodes** to perform methods, or actions, on an application or VI. A single Invoke Node can execute only one method on an application or VI.

每个调用节点只能执行一个方法。

There are three Application Class methods. They perform the following functions:

- **Bring To Front**—Brings all the windows of the application to the front.
- **Disconnect From Slave**—Disconnects the LabVIEW Real Time (RT) development system from the target RT engine. This only applies to LabVIEW RT development systems.
- **Mass Compile**—Allows you to mass compile a set of VIs in a specific directory.

Some of the important VI Object methods exported by the VI Server are `Export VI Strings`, `Set Lock State`, `Run VI`, `Save Instrument`, and so on. The `Export VI Strings` method exports strings pertaining to VI and front panel objects to a tagged text file. The `Set Lock State` method sets the lock state of a VI. The `Run VI` method starts VI execution. The `Save Instrument` method saves a VI.



The **Close Application or VI Reference** function releases the application or VI reference. If you close a reference to a specified VI and there are no other references to that VI, LabVIEW can unload the VI from memory.

This function does not prompt you to save changes to the VI. By design, VI Server actions should avoid causing user interaction. You must use the `Save Instrument` method to save the VI programmatically.

Note If you do not close the application or VI reference with this function, the reference closes automatically when the top-level VI associated with this function finishes execution. However, it is a good programming practice to conserve the resources involved in maintaining the connection by closing the reference when you finish using it.

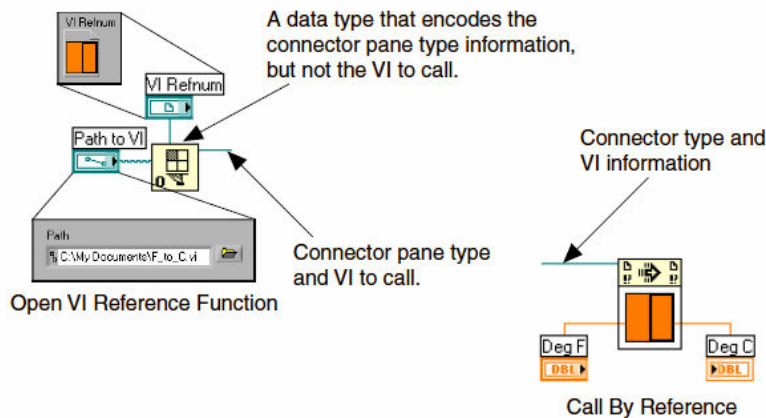
当关联的顶层的VI结束执行后，会自动关闭应用程序或VI的引用。但手动关闭引用来保存连接时调用的资源，才是好的编程习惯。**Close Reference**不提示保存VI的改变，必须利用`Save Instrument`方法编程保存VI，所以VI Server应尽量避免用户交互。

Strictly Typed VI Refnums

Use strictly typed VI refnums to call VIs dynamically, which saves memory if the application is very large. It also allows a large application to be started more quickly, as

LabVIEW does not need to load all the VIs at the beginning.

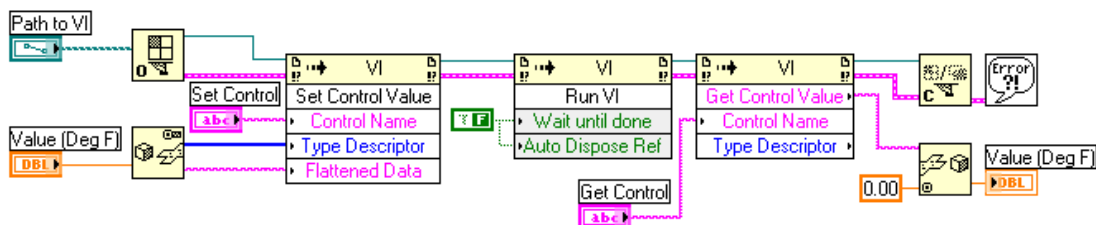
You can pass the strictly typed VI refnum to sub VIs, just like any other data type. You need to make sure that the strictly typed VI refnum control in the subVI is mapped to the connector pane.



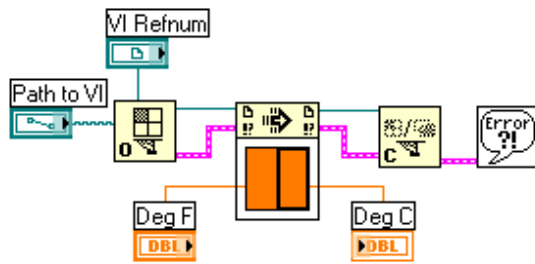
To create a **strictly typed refnum**, place an Application or VI Refnum control, located on the **Controls»Path&Refnum** palette, on the front panel of a new VI. Right-click the refnum and select **Select VI Server Class» Browse** from the shortcut menu and select a VI from the dialog box that appears.

You can also create a **strictly typed refnum** by dragging and dropping a VI icon onto the refnum.

Strictly typed refnums is a data type that contains the connector pane information of a VI. The type specifier displays its connector pane. Note that you are opening a reference to a VI that has a connector pane of the type you have just chosen. It does not store any link to the VI you select. LabVIEW provides methods that allow you to modify the values of controls in a VI and read the values of indicators. Using methods also requires you to flatten data to strings before it can be passed to the method. As the following block diagram shows, using this paradigm can cause your block diagram to look very messy and confusing if you are modifying a lot of controls or indicators.



Using a Call by Reference Node, as show in the following block diagram, you can write or read data to a VI in a much simpler manner.



strictly typed refnum means that the connector terminals of a called VI and the data type they can accept are fixed. It does not define the name of the VI being called—**Open VI Reference** does that. When you use **Open VI Reference** and you supply a VI refnum, it checks to see if the VI has the same connectors and data types as defined in the VI refnum. If it does not, an error is generated.

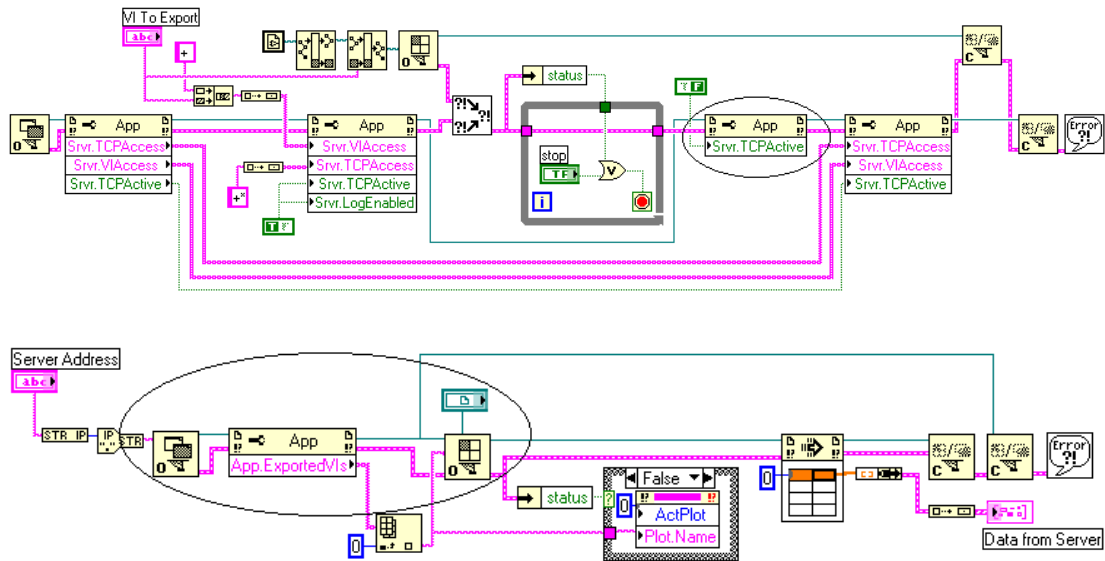
Strictly typed refnum 意味着某 VI 的连接端子和可接受的数据类型，并没有定义要调用的 Vi 的名字——那是 **Open VI Reference** 的工作。当利用 VI 引用打开一个 VI,它检查 VI 的连接和数据类型是否和 VI 引用定义的一样。

When you call a VI using the Call by Reference Node, no checking or interpretation of data types is carried out. This is known as strong typing. Using methods, the application must ensure that any data written to or read from a VI is the correct data type when it is flattened and unflattened. The advantage of strong typing is speed. Removing the overhead of interpreting raw binary data into the correct form, (integer, floating point, string, and so on) reduces the execution time of an application, especially if it is large.

When you open a strictly typed refnum, the referenced VI is reserved for running and cannot be edited state, it means that the VI has been checked to make sure it is not bad, that it is not currently running as a top level VI, and that it has been compiled (if necessary), as well as a few other checks. A VI referenced by a strictly typed VI reference can be called using the Call By Reference Node at any moment without having to check all these conditions again. Thus, in this state you cannot edit the VI or do anything to it that would change the way it would execute.

Remote Communication

LabVIEW allows most VI Server operations in a remote version of LabVIEW across a TCP/IP network. To open an application reference to a remote version of LabVIEW, you must specify the computer name input to the Open Application Reference function. Then LabVIEW attempts to establish a TCP connection with a remote VI Server on that computer on the specified port.



VI Server Configuration for External Applications

To configure VI Server for external applications, select **Tools»Options** on the server computer and select **VI Server:Configuration**. The options specify whether applications access the VI Server through TCP/IP or ActiveX protocols. For a remote computer, enable **TCP/IP** and enter a **Port** number that client applications can use to connect to the server. Once you enable TCP/IP, configure which Internet hosts have access to the server.

When you allow remote applications to access VIs on the VI Server, you should specify which VIs these applications can access. To configure the exported VIs, select **Tools»Options** on the server computer and select **VI Server:Exported VIs** from the pull-down menu. You can use the **?**, *****, and ****** characters as wildcard characters. The **?** and ***** wildcards do not include the path separator. ****** includes the path separator. When you allow remote applications to access the VI Server using the TCP/IP protocol, you should specify which Internet hosts have access to the server. Configure the TCP/IP access permissions in the **VI Server:TCP/IP Access** dialog box from the **Tools»Options** menu. The conversion from an IP address to its domain name is called name lookup. A name lookup or a resolution can fail when the system does not have access to a DNS (Domain Name System) server or when the address or name is not valid.

Strict Checking determines how the server treats access list entries that cannot be compared to a client's IP address because of resolution or lookup problems. When **Strict Checking** is enabled, a denying access list entry that encounters a resolution problem is treated as if it matched the client's IP address. When **Strict Checking** is disabled, an access list entry that encounters a resolution problem is ignored.

To specify an Internet host address, you can specify either its domain address or IP address. You can also use the * wildcard when specifying Internet host addresses. **Note** If the VI Server runs on a system that does not have access to a DNS server, do not use domain name entries in the TCP/IP access list. Requests to resolve the domain name or an IP address will fail, slowing down the system. For performance reasons, place frequently matched entries toward the end of the TCP/IP Access List.

Lesson 6

Calling and Creating Shared Libraries (DLLs)

A shared library is a segment of code that contains exported parameters that an application can access at run time. The library is stored as a binary file. A dynamic link library (DLL) is a type of shared library specific to Microsoft Windows. On Macintosh and UNIX, you also can access shared libraries by a C/C++/Visual Basic application just like you can access a DLL Windows.

A DLL is an executable file that cannot run on its own. The operating system loads the DLL into memory when the actual application requests it. Then the requesting application uses the code from the DLL as if it was included in its own binary file without knowing how or what language in which the DLL was written. The DLL provides a list of exported functions that developers can call with a simple application programming interface (API) that provides no hint as to how the functionality is implemented internally. DLLs consist of a few special functions and a number of programmer-defined functions to accomplish common tasks,

All shared executables reference the same copy of the library, which need not be included in executable images stored on disk. As a result, shared executables are smaller than static executables, and shared libraries reduce memory use.

When a shared library is updated, all programs that use it immediately benefit from the change, without have to be rebuilt. The disk and memory savings of shared libraries is offset by a slight performance penalty when a shared executable starts up. References to shared library routines must be resolved by finding the libraries containing those routines. However, references need be resolved only once, so any performance penalty is small.

DLL是一种被封装起来不可单独运行的执行文件。开发者可以用应用程序接口(API)在不知其内部如何实现得情况下来调用其内部的函数。所有的执行者共享库的一个副本，共享执行小于静态执行，减少内存使用。当库更新时，所有调用的程序不需要重新编译就可获益。磁盘和内存的节省需要一点很小的性能代价，即寻找包含那些程序的库，不过只需一次就够了。

Note DLLs can be placed in files with different extensions such as EXE, DRV, or DLL. In Windows, you can use QuickView to view the exported function names within a 32-bit DLL. If you have the Visual C++ compiler installed on your computer, you also can use the Dumpbin utility to view exported function names.

DLL可以用EXE,DRV或DLL作扩展名。在Windows中可以用[快速查看](#)来查看32位DLL中的输出函数名，也可以用VC++自带的**Dumpbin**来查看。

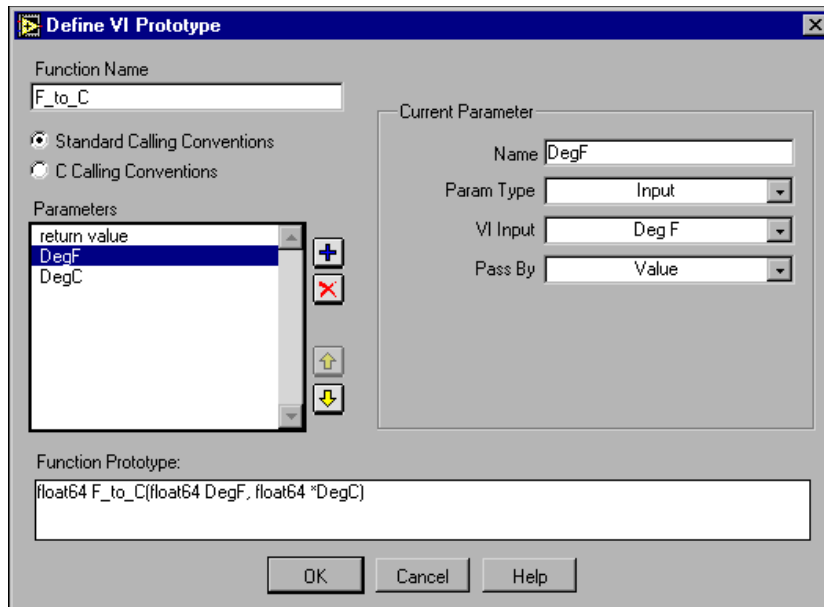
The following code shows the basic structure of a DLL.

```
BOOL WINAPI DllMain (HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
    switch (fdwReason)
    {
        case DLL_PROCESS_ATTACH: /* Init Code here*/
            break;
        case DLL_THREAD_ATTACH: /*Thread-specific code here */
            break;
        case DLL_THREAD_DETACH: /*Thread-specific cleanup code here. */
            break;
        Case DLL_PROCESS_DETACH: /*Cleanup code here */
            break;
    }
    /* The return value is used for successful DLL_PROCESS_ATTACH */
    return TRUE;
}
/* One or more functions */
__declspec (dllexport) DWORD Function1(...){}
__declspec (dllexport) DWORD Function2(...){}
```

When you compile a VI to DLL in LabVIEW, you need to the following work:


- a. Click **Add Top-Level VI** to define the top-level VI for your application.
- b. Click **Add Exported VI** to add a VI that will be exported as a function in the shared library.
- c. Click the **Define VI Prototype** button to define the prototype for the exported VI.

编译DLL时，需要指定应用程序的顶层VI，指定哪些VI作为库的输出函数，并定义它们的原型。



The function `float64 F_to_C(float64, float64)` in the dialog above may be called in VB as following:

```
Private Declare Sub DegFtoDegC Lib "C:\Tmp\app\Convert_Temp.dll"
    Alias "F_to_C" (ByVal degF As Double, ByRef degC As Double)
Private Sub Start_Click()
    Dim DegreeF As Double
    Dim DegreeC As Double
    DegreeF = FTextBox.Text
    Call DegFtoDegC(DegreeF, DegreeC)
    CTextBox.Text = DegreeC
End Sub
Private Sub Quit_Click()
End
End Sub
```

 The Call Library function allows you to select the following return types and parameters for your DLL:

- **Void**—**Void** is accepted only for the return value, whereas **Adapt to Type** (included two similar items: **Handles by Value** and **Pointers to Handles**) is accepted only for parameters (*void*, *void**)
- **Numeric data**—For numeric data types, you must specify the exact numeric type out of the following items:
 - Signed and unsigned versions of 8-bit, 16-bit, and 32-bit integers. (*char*, *short int*, *long*)
 - 4-byte, single-precision numbers. (*float*)
 - 8-byte, double-precision numbers. (*double*)
 You must use the format ring to indicate if you pass the **value** or a **pointer to value**.
- **Arrays**—You can indicate the data type of arrays (using the same items as for

numeric data types), the number of dimensions, and the format to use to pass the array. Use the Format item to pass an **Array Data Pointer** or an **Array Handle** or an **Array Handle Pointer**. If you use the **Array Data Pointer**, pass the array dimension as a separate parameter(s).

- **Strings**—You should specify the format for strings. The items can be **C string pointer**, **Pascal string pointer**, **String handle** or **String handle pointer**. If the library function you call is written specifically for LabVIEW, you might want to use the **String Handle** format, which is a pointer to a pointer to four bytes for length information, followed by string data. The Win32 API uses the **C string pointer**. (*CStr, Pstr, LstrHandle, LstrHandle**)
- **Waveforms**—you can indicate the data type of **Waveform**, **Digital Waveform**, **Digital Table**, even the number of dimensions of **Waveform**. (*Hwave*)
- **ActiveX**—There Special pointer for variant: **ActiveX Variant Pointer**, **IDispatch* Pointer**, **IUnknown* Pointer** (*Variant**, *IDispatch***, *IUnknown***)

By default, the **Call Library** function runs in the user interface thread or a single thread. Hence, when you first select the **Call Library** function, it appears orange in color. When you mark your DLL as thread safe(reentrant), the Call Library function changes to a yellow colored icon, indicating that the function in the DLL is thread safe. A DLL is defined to be thread safe if it can be reliably called from two or more separate threads. You will not need to type the entire path to the DLL unless the DLL is stored in a location that does not appear in the `PATH` variable. All input terminals to the Call Library function must receive data, especially pointers, or may cause errors in Windows that might result in a crash or incorrect behavior.

缺省情况下，**Call Library** 函数运行在用户交互线程或单线程中，这时显示为橙色。当 DLL 能被两个以上的不同线程可靠调用时，可以把 DLL 标记为安全线程(重入)，这时图标变为黄色。

另外，如果 DLL 路径出现在 `Path` 环境变量中，就不需要输入完整路径。参数顺序要和函数原型一致。输入输出类型要匹配（但不强求完全一致，兼容即可）。每个输入参数都必须收到值，特别是指针一定要初始化,否则导致系统异常甚至崩溃。

Note In Win32, you can use `_cdecl` and `_stdcall` calling conventions. The calling convention determines the order in which arguments passed to the functions are pushed onto the stack. It also determines which function, calling or called, removes the arguments from the stack. The standard (`_stdcall`) calling convention is used to call Win32 API functions. Parameters are passed by a function onto the stack from right to left and are passed by value unless a pointer or reference type is passed. Function arguments are fixed, and a function prototype is required. The callee is responsible for popping its own arguments from the stack.

The C calling convention is the default calling convention for C and C++ programs, it passes arguments in order left to right, and the stack is cleaned up by the caller. The C calling convention creates a larger executable because it requires each function call to include stack cleanup code.

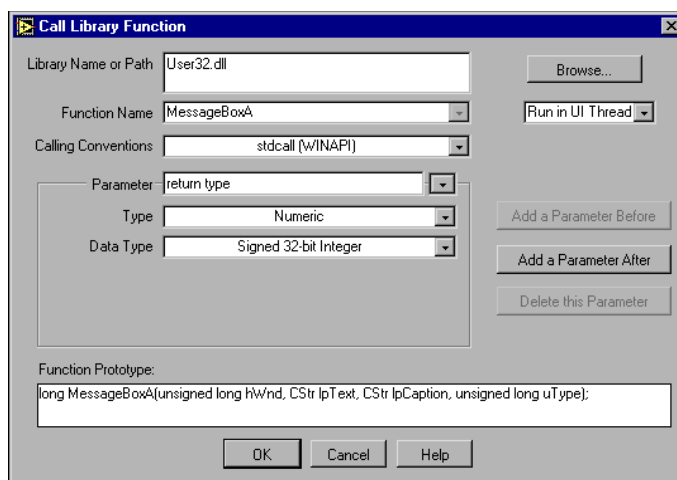
在 Win32 中，可使用 `_cdecl` 和 `_stdcall` 调用习惯。

标准调用习惯(`_stdcall`)用来调用 Win32 API 函数，函数将参数以数值形式除非是指针或引用，从右至左传入堆中。函数参数表是确定的，并要求函数原型，被调用者负责清理堆。

而 C 调用习惯(`_cdecl`)是调用 C/C++程序的缺省方法。它从左至右传递参数表，且调用者负责清理堆。每个调用函数都需要增加清理代码，这会造成整个执行代码的增加。

Examples: Several WinAPI in `User32.dll` by `stdcall`

1. `int MessageBoxA (hWnd, lpText, lpCaption, uType)`

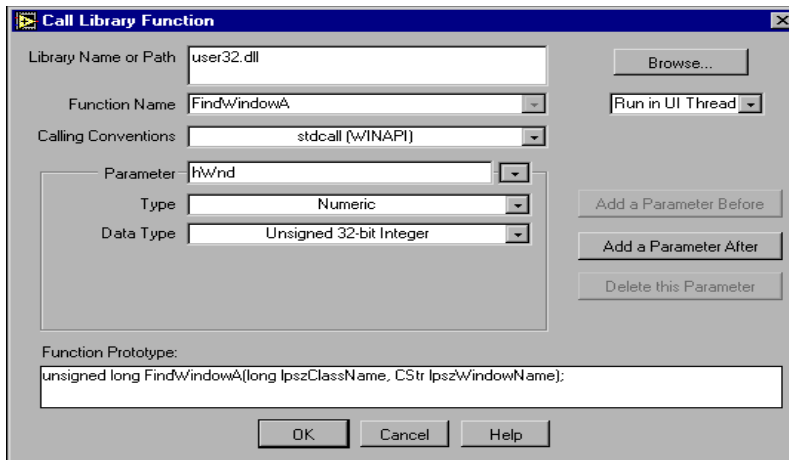


Message Box Button Type	uType
OK	0
OK CANCEL	1
ABORT RETRY IGNORE	2
YES NO CANCEL	3
YES NO	4
RETRY CANCEL	5

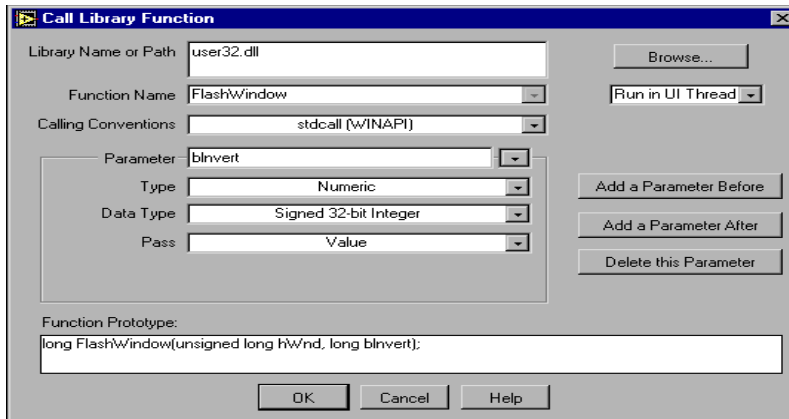
2. `HWND FindWindowA (lpClassName, lpWindowName)`

If `lpClassName` is NULL, all class names match. In this case, you must pass a NULL to this parameter. In LabVIEW, when you pass an empty string to a DLL, you do *not* pass a NULL pointer—just a pointer to a 0-byte string. Because a NULL pointer has a numeric value of zero, the easiest way to pass it is to send an integer value of zero. Hence, you will pass a 0 as a long integer.

在C/C++中 "" 表示空指针NULL,等同于0x00.然而在LabVIEW中，一个空字符串并不代表空指针NULL，而是一个指向0长度字符串的有效指针。所以想传递空指针可以直接发送整数形式的0x00，相应的参数类型要从String改为Integer。



3. BOOL FlashWindow (hWnd, bInvert)



4. int32 SetWindowTextA(uInt32 hWnd, CStr windowName)

Thread-safe DLLs

As long as the VI does not explicitly violate the following conditions, LabVIEW creates the DLL to be thread-safe:

- The VI does not have global storage data (global variables, files on disk, and so on).
- The VI does not access hardware (registers).
- The VI does not call functions, shared libraries, or drivers that are thread-unsafe.
- The VI does not protect access to global resources with semaphores or mutexes.
- The VI is not called from a non-reentrant VI.

只要不明显违反下列条件，LabVIEW创建的DLL就是安全线程：

- VI没有包含全局数据（全局变量，磁盘文件等）
- VI没有访问硬件（寄存器）
- VI没有调用不安全线程的函数，共享库或驱动程序。
- VI没有用旗语或互斥来保护全局资源的访问权限。
- VI没有被非重入的VI调用。

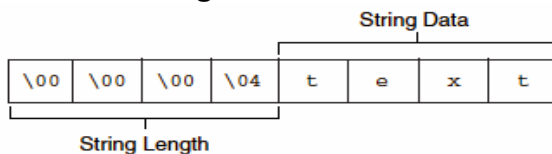
Array Options with DLLs

Arrays of numeric data can be of any integer type or single-precision (4-byte) or double-precision (8-byte) floating-point numbers. When you pass an array of data to a DLL function, you have the option to pass the data as an **Array Data Pointer** or as a LabVIEW **Array Handle**. When you pass an **Array Data Pointer**, you also can set the number of dimensions in the array, but you do not include information about the size of the array dimension(s). DLL functions either assume the data is of a specific size or expect the size to be passed as a separate input.

The LabVIEW Build utility allows you to choose any one of these methods at build time. When an application calls the same DLL, you must know which option was selected so that the call and the DLL are compatible.

如果用**Array Data Pointer**来传递数组，只可以设置数组维数，不包括每维的大小。DLL函数要么事先确定数据大小，要么通过单独的输入端传递进来。这两种方法LabVIEW都允许。调用时必须知道用的哪种方法才能使调用过程和被调用的DLL兼容。

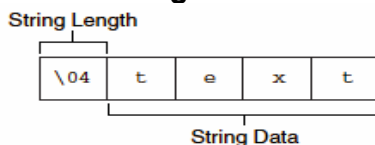
LabVIEW String Handle



LabVIEW stores a string in a special format in which the first four bytes of the array of characters form a signed 32-bit integer that stores how many characters appear in the string. Thus, a string with n characters will require $n + 4$ bytes to store in memory. The advantage of this type of string storage is that NULL characters are allowed in the string. Strings are virtually unlimited in length, up to 2^{31} characters.

LabVIEW用字符串句柄指向的数组来保存字符串，这样的好处是允许空串NULL存在。前四字节保存串长度，后面才是字符串，串长可达 2^{31} 个字符。长度为 n 字节的串占用 $n+4$ 字节空间。

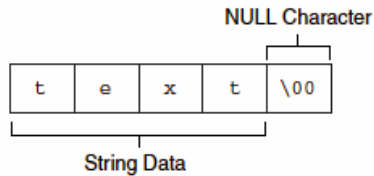
Pascal String Format



The Pascal string format is nearly identical to the LabVIEW string format, but instead of storing the length of the string as a signed 32-bit integer, it is stored as an unsigned 8-bit integer. This limits the length of a Pascal style string to 255 characters. A Pascal string that is n characters long will require $n + 1$ bytes of memory to store.

Pascal字符串格式类似于LabVIEW字符串句柄。但它用一字节来保存串长度，限制在255个字符。长度为n字节的串占用n+1字节空间。

C String Format



The similarities between the C-style string and normal numeric arrays in C becomes much clearer when you notice that C strings are declared as `char *`. C strings do not contain any information that directly gives the length of the string, as do the LabVIEW and Pascal strings. Instead, C strings use a special character, called the NULL character, to indicate the end of the string, as shown in the following figure. NULL is defined to have a value of zero in the ASCII character set. Note that this is the number zero and not the character 0. Thus, in C, a string containing n characters requires $n + 1$ bytes of memory to store. The advantage of C-style strings is that they are limited in size only by available memory. However, if you are acquiring data from an instrument that returns numeric data as a binary string, as is common with serial or GPIB instruments, values of zero in the string are possible. For binary data where NULLs might be present, you probably should use an array of unsigned 8-bit integers. If you treat the string as a C-style string, your program will assume incorrectly that the end of the string has been reached, when in fact your instrument is returning a numeric value of zero.

C字符串格式被声明为`Char*`,这与数组类似。它不包含串长度信息，而是用NULL字符指示串的结尾。NULL被定义为数值0(0x00)，而不是字符'0'(0x30)。C字符串的好处是串长只受限于可用内存。但是，当从仪器(如GPIB仪器)采集数据时，返回的二进制字符串可能包含数值0。这时应该使用数组。如果仍看成C字符串，当返回数值0时，程序会错误地认为到达了串尾。长度为n字节的串占用n+1字节空间。

Lesson 7 ActiveX Automation

OLE

The term OLE pertains to the technologies associated with linking and embedding, including OLE containers, OLE servers, OLE items, in-place activation (or visual editing), trackers, drag and drop, and menu merging. With OLE, you can create and edit compound documents that contain data of different formats, created by multiple applications. OLE objects consist of data and a set of methods for manipulating that data. OLE objects maintain the data and provide an interface through which other objects can communicate. The following are some OLE technologies:

- **Linking and Embedding**—Placing an object or a link to that object inside another object. For example, embedding is placing a spreadsheet inside a Word document. Linking is

saving a link to the spreadsheet file in the document.

- **In-Place Activation(or visual editing)**—Allowing a user to modify an embedded object using the native application. For example, if a table from a spreadsheet is embedded in your document, you could edit the tables with the container application document.
- **Automation**—Manipulating an object or application within another application or object. For example, programming MS Word from another application such as LabVIEW or a C program.
- **Compound Files**—How objects are stored. This technology is useful in implementing a structured storage technology for creating disk files and improving performance.
- **Uniform Data Transfer**—Data transfer mechanism for objects. For example, an application can handle clipboard transfers that deal with disk-based storage mediums.
- **Drag and Drop**—Enabling objects in a container application to respond to mouse clicks.
- **Monikers**—Internal objects containing information about the path to a linked object.
- An application that contains these compound documents is called a container application. A container application can also contain OLE custom controls or OCX, which are now called ActiveX controls.

OLE是一种与嵌入和链接有关的技术。OLE对象由数据和操作这些数据的方法组成，提供了与其他对象通讯的接口。以下是一些OLE技术：

- **链接和嵌入**—在对象内放置一个对象或其链接。举例来说，嵌入就是在你的Word文档中放置一个电子表格；链接就是保存电子表格的链接到文档中。
- **现场激活(可视化编辑)**—允许用户用本地程序修改对象。例如你的文档中嵌入了电子表格，你可以用容器应用程序文档来修改它。
- **自动操作**—用对象或程序操纵其他对象或程序。例如，在LabVIEW或C中编辑MS Word。
- **复合文件**—对象如何存储。在实现创建磁盘文件和改进性能的结构存储技术时很有用。
- **统一的数据传输**—数据传输机制。例如，程序可以操纵剪贴板转移者来处理磁盘存储。
- **拖放**—使容器应用程序中的对象能对鼠标点击做出反应。
- **绰号**—包含链接对象的路径信息的内部对象。
- 包含OLE控件和ActiveX控件(过去叫OCX)的容器应用程序。

ActiveX

ActiveX is a diverse set of technologies based on COM (Component Object Model). The COM standard allows developers to create code and applications from a multitude of different languages and build a defined interface to that code, making it easily accessible by other applications. Applications can access the functionality of other applications through the standard interface. The following are common ActiveX technologies:

- **ActiveX Controls**—Interactive objects that can be used in containers such as a Web site.
- **ActiveX Documents**—Enables users to view documents, such as Microsoft Word or Excel files, in an ActiveX container.
- **Automation**—Enables communication of data and commands between different applications.
- **Active Scripting controls**—The integrated behavior of a lot of ActiveX controls and/or Java programs from a browser or server.

ActiveX是微软倡导的网络化多媒体对象技术，一种基于COM(组件对象模型)的多样性技术。COM标准允许开发者用不同的语言创建代码和应用程序并定义接口，从而使其他程序可以容易

的访问。应用程序通过这种标准接口可以访问其他程序的函数。以下是一些ActiveX技术：

- ActiveX控件—用于容器（如网站）的交互对象。
- ActiveX文档—使用户能在Active容器中查看MS Word或Excel文档。
- 自动操作—实现不同应用程序之间的数据和命令通讯。
- ActiveX脚本控制—许多ActiveX控件及浏览器中的Java程序的综合行为。

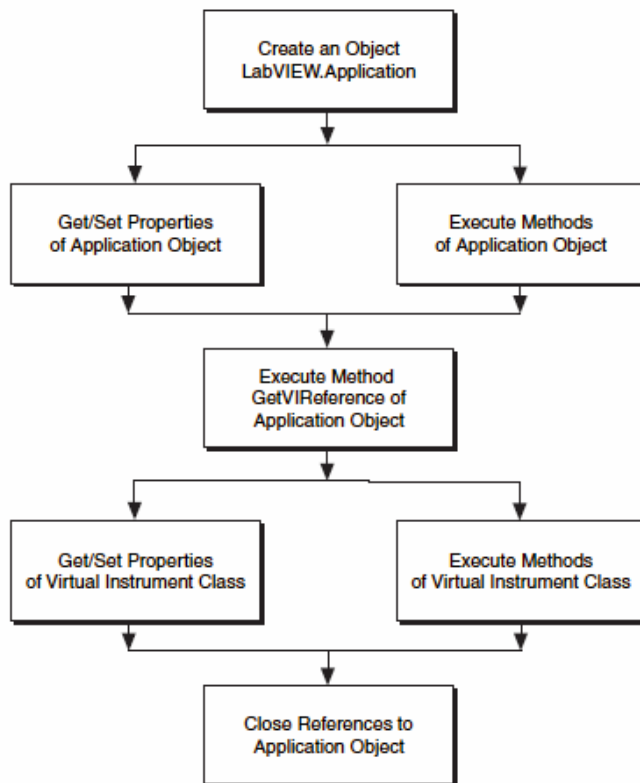
ActiveX Automation

ActiveX automation defines the communication protocol between two applications. One application acts as the server and the other as a client. An automation server exposes methods or actions that can be controlled by a client application. An automation client creates and controls objects exposed by a server application. An ActiveX automation object is an instance of a class that exposes properties, methods, and events to ActiveX clients.

To create and access objects, automation clients need information about a server's objects, properties, and methods. Often, properties have data types, and methods return values and accept parameters. A list of exposed objects is provided in the type library of the application. A type library contains specifications for all objects, methods, or properties exposed by an automation server. Also, the documentation of the server application contains information about exposed objects, properties, and methods. The type library file usually has a TLB filename extension.

ActiveX自动操作定义了两个程序的通讯协议。一个作为服务器另一个作为客户。服务器提供可被客户控制的方法或行为，而客户建立和控制服务器提供的对象。ActiveX自动操作对象是提供属性，方法和事件的类的实例。通常，属性带有数据类型，方法返回值和接受参数。建立和访问对象，自动操作需要属性和方法的信息，这些规格由类型库(*.TLB)或服务提供。

LabVIEW Automation Programming Model



下面是一个例子。在LabVIEW中，选择**Tools»Options»Server Configuration**允许ActiveX协议。在**Tools»Options»VI Server:Exported VIs**下，允许Frequency Response VI的访问权限。并确保LabVIEW设置为自动登录。安装MS Excel，然后打开labview7.0\examples\comm\freqresp.xls
如果宏被禁用，在**工具»宏»安全性**中设置安全级别为中即可。查看其中的LoadData()宏：

```

Sub LoadData()
  \ LoadData Macro
  \ Keyboard Shortcut: Ctrl+L
  \ This example demonstrates LabVIEW's Active-X server capabilities.
  \ Executing this macro loads an example VI "FrequencyResponse.vi",
  \ runs it and plots the result on an Excel Chart.
  Dim lvapp As LabVIEW.Application
  Dim vi As LabVIEW.VirtualInstrument
  Dim paramNames(4), paramVals(4)
  Set lvapp = CreateObject("LabVIEW.Application")
  viPath = lvapp.ApplicationDirectory + "\examples\apps\freqresp.llb\Frequency Response.vi"
  Set vi = lvapp.GetVIReference(viPath) \Load the vi into memory
  vi.FPWinOpen = True \Open front panel
  \ The Frequency Response vi has
  \ 4 inputs - Amplitude, Number of Steps, Low Frequency & High Frequency
  \ 1 output - Response Graph.
  \ To run the Frequency Response VI, invoke the Run method with names of
  \ inputs and outputs passed along with their values.
  paramNames(0) = "Amplitude"
  paramNames(1) = "Number of Steps"
  paramNames(2) = "Low Frequency"
  paramNames(3) = "High Frequency"
  paramNames(4) = "Response Graph"
  \initialize input values to the vi

```

```

paramVals(0) = Sheet1.Cells(4, 5) 'Amplitude value from cell (4, 5)
paramVals(1) = Sheet1.Cells(5, 5) '# steps value from cell (5, 5)
paramVals(2) = Sheet1.Cells(6, 5) 'Low Frequency value from cell (6, 5)
paramVals(3) = Sheet1.Cells(7, 5) 'High Frequency value from cell (7, 5)
' paramVals(4) contains the value of Response Graph after running the vi.
' run the vi
Call vi.Call(paramNames, paramVals)
' paramVal(4) contains value for Response Graph - a cluster of 2 arrays
' In Active-X, a cluster is viewed as an array of variants,
' so a cluster of 2 elements x & y is an array of 2 variant elements
x = paramVals(4)(0) ' x co-ordinates
y = paramVals(4)(1) ' y co-ordinates
' Fill the excel columns 1 & 2 with the graph co-ordinates
' These columns are used by Excel to plot the chart
first = LBound(x, 1)
last = UBound(x, 1)
Sheet1.Columns(1).Clear
Sheet1.Columns(2).Clear
For i = first To last
Sheet1.Cells(i - first + 1, 1) = x(i)
Sheet1.Cells(i - first + 1, 2) = y(i)
Next i
End Sub

```

该宏建立了LabVIEW.Application类，打开\labview7.0\examples\apps\freqresp.llb\FrequencyResponse.vi，运行VI，把返回的数组绘制在Excel里。

还有一个宏的例子：

```

Sub Acquire_Data()
' This macro changes the Front Panel title of the VI
' It also saves the VI in HTML format.
' 10 Points are acquired from the Temperature sensor.
Dim lvapp As LabVIEW.Application
Dim vi As LabVIEW.VirtualInstrument
Dim ParamVals(4)
Set lvapp = CreateObject("LabVIEW.Application")
viPath = lvapp.ApplicationDirectory + "anlogin.llb\Acquire 1 Point from 1 Channel.vi"
Set vi = lvapp.GetVIREference(viPath) 'Load the vi into memory
vi.FPWinOpen = True 'Open front panel
vi.FPWinTitle = "Acquire A Point"
Call vi.PrintVIToHTML ("c:\Exercises\LV Advanced\
Acquire.htm",0,0,eJPEG, 256, "C:\Exercises\LV Advanced")
Call vi.SetControlValue("device", "1")
Call vi.SetControlValue("channel", "0")
For i = 1 To 10
Call vi.Run
Sheet1.Cells(i) = vi.GetControlValue("sample")
Next i
End Sub

```

```

Sub Clear_Chart()
' Clear_Chart Macro
' Keyboard Shortcut: Ctrl+M
Sheet1.Rows(1).Clear
End Sub

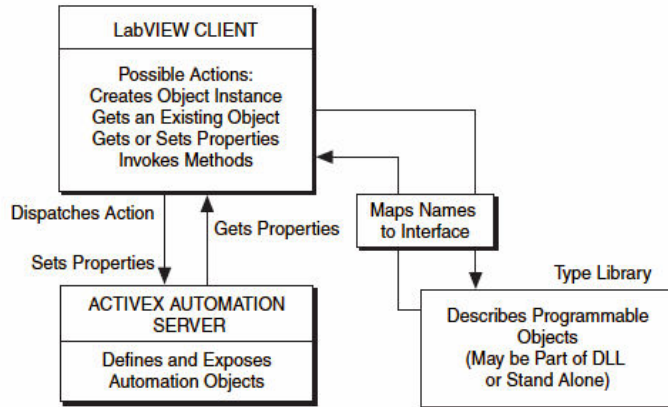
```


该宏建立了LabVIEW.Application类，打开anlogin.llb\Acquire 1 Point from 1 Channel.vi，运行VI，读写控件和指示器的值，改变程序标题，输出VI信息到HTML文件，把采集10个点绘制在Excel里。


LabVIEW提供两种对象类—Application class and Virtual Instrument class.

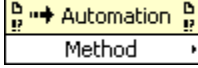
Lesson 8

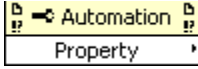
LabVIEW ActiveX Automation Client and ActiveX Container



 The **Open Automation Refnum** function opens an automation refnum that refers to a specific ActiveX Automation object. You can select the class of the object by right-clicking the function and selecting **Select ActiveX class**. You should select only creatable classes as inputs to this function. This list of creatable objects is generated by accessing the Windows Registry. After you create a refnum, you can pass it to other ActiveX functions.

 The **Close Automation Refnum** function closes an automation refnum. Make sure that you close every automation refnum when you no longer need it.

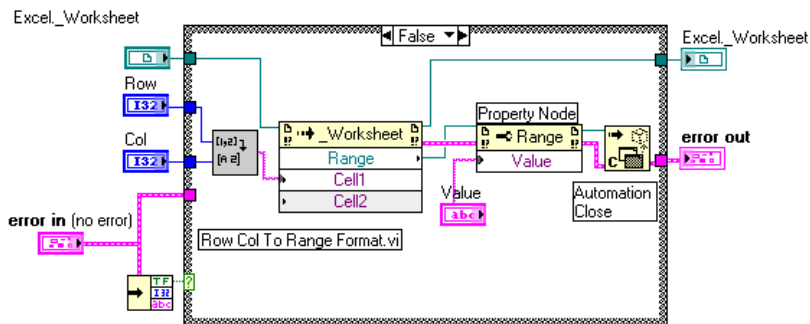
 The **Invoke Node** function invokes a method or an action on an ActiveX automation object. To select an ActiveX class object, right-click the Invoke Node and select **Select Class»ActiveX** or wire an automation refnum to the input. To select a method related to the object, right-click the bottom of the Invoke Node and select **Methods**. After you select a method, the appropriate parameters appear automatically below it. Parameters with white backgrounds are required, and the parameters with gray background, are optional.

 The **Property Node** function sets or gets ActiveX object property information. To select an ActiveX class object, right-click the Property Node and select **Select Class »ActiveX** from the shortcut menu. Then, select a property related to that object by right-clicking the bottom of the node and selecting **Properties** from the shortcut menu. To set property information, right-click the Property Node and select **Change to Write** from the shortcut menu. To get property information, select **Change to Read**. Some properties are read-only or write-only. In these cases **Change to Write** or **Change to Read**, respectively, is dimmed in the shortcut menu. To add items to a Property Node, right-click the node and select **Add Element** from the shortcut menu, or click and drag to expand the node.

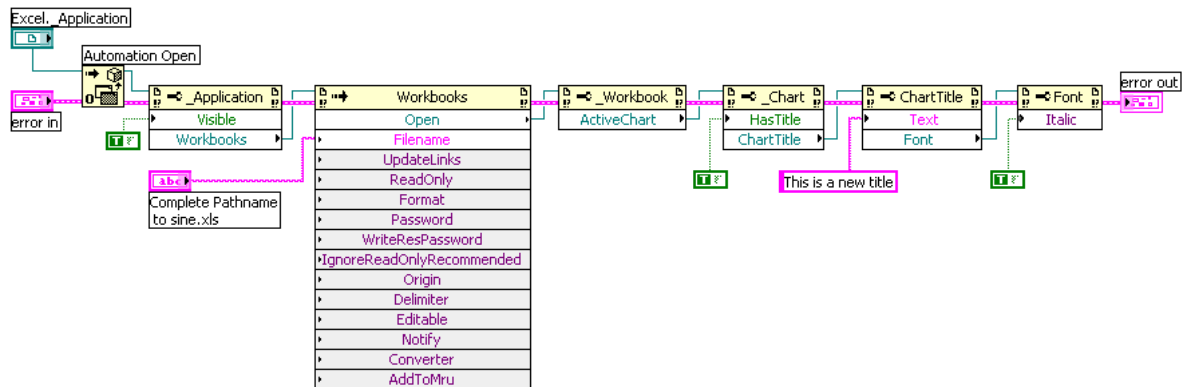
Some applications provide ActiveX data in the form of a self-describing data type called

an ActiveX or OLE Variant. To view the data or process it in LabVIEW, you must convert it to the corresponding LabVIEW data type using **Variant to Data** function. **Note** If you are writing a property or indicator of ActiveX variant type wiring the LabVIEW data type ,which can be automatically converted to variant data types. This conversion is indicated by a coercion dot.

有些应用程序以ActiveX或OLE Variant的独特描述方式提供ActiveX数据。你必须用**Variant to Data**函数转换为LabVIEW数据类型才能查看。如果是将LabVIEW数据类型写入Variant数据类型的指示器或属性节点，LabVIEW会自动强制转换。



上图访问MS Excel，设置某些单元格的数值。



上图访问MS Excel，设置指定文件的图表的标题。

You can find the Microsoft Excel object hierarchy in the Microsoft Visual Basic reference in the Help menu. Excel objects are arranged hierarchically with an object named **Application** at the top of the hierarchy. All other objects fall under **Application**. To call the properties and methods of an object in Excel, you must reference all objects that lie on the hierarchical path to that object. For example, to access the **Chart** object, you must first access the **Application** object, the **Workbook** object, and then the **Chart** object.

Excel对象按层次组织。最顶层是**Application**对象，其他对象在下面。要调用某个对象的属性和方法必须引用对象路径上的所有对象。比如，要访问**Chart**对象，必须先访问**Application**对象，**Workbook**对象，最后才是**Chart**对象。

Remote Automation

DCOM (Distributed Component Object Model) is a Microsoft technology that allows software applications to communicate directly with each other across networks, supported only on the Windows operating systems. DCOM allows you to communicate over the network using ActiveX remote automation to build distributed applications. LabVIEW supports remote automation by using DCOM. Thus, ActiveX clients can use DCOM to communicate with LabVIEW running on a remote computer.

DCOM is a complex technology, and it can be confusing while you are trying to use it to configure an application. There are several security issues that must be considered. There are two limitations of DCOM and LabVIEW. You cannot communicate between two copies of LabVIEW on different computers. The client LabVIEW will intercept all calls to the server. Also, you cannot perform remote activation if the server is on a Windows 98 computer. LabVIEW must be launched manually on the server computer.

DCOM(分布式对象组件模型)仅支持Windows操作系统，是一种允许软件程序利用ActiveX远程自动操作建立分布式应用程序，直接通过网络互相通信。ActiveX客户可利用DCOM与远程计算机上的LabVIEW通讯。DCOM是种复合技术，当你试图使用它配置应用程序会产生混乱，有几个安全性问题必须考虑。特别地，两台不同计算机上的LabVIEW副本不能通讯；如果服务器在Win98系统上，不能进行远程活动，必须手动地启动服务器。

ActiveX Containers

- **Create Document**—Select a document registered with your system. Documents are objects that you can either link to or embed in a container.

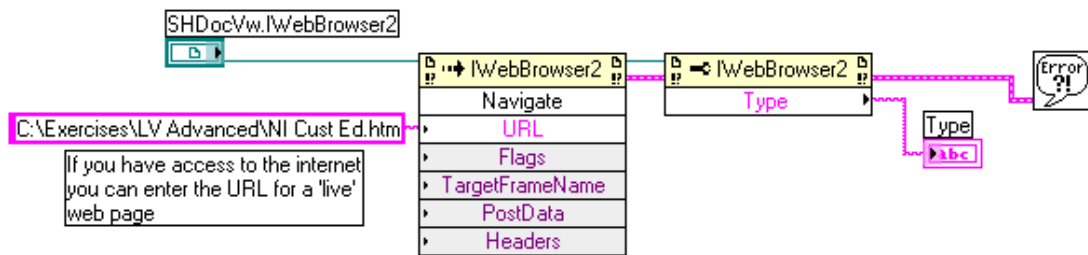
When you link to a document with a container by selecting **Link To File** option, you establish a view to that document. Any changes to that document are reflected in the container application, which updated when the front panel object is updated.

When you embed a document with **Link To File** option unselected, the document now becomes a part of the container application. If you make any changes to the document outside of the container application, the container will not see these changes. An example of an ActiveX document is a Microsoft Word document embedded in a PowerPoint application.

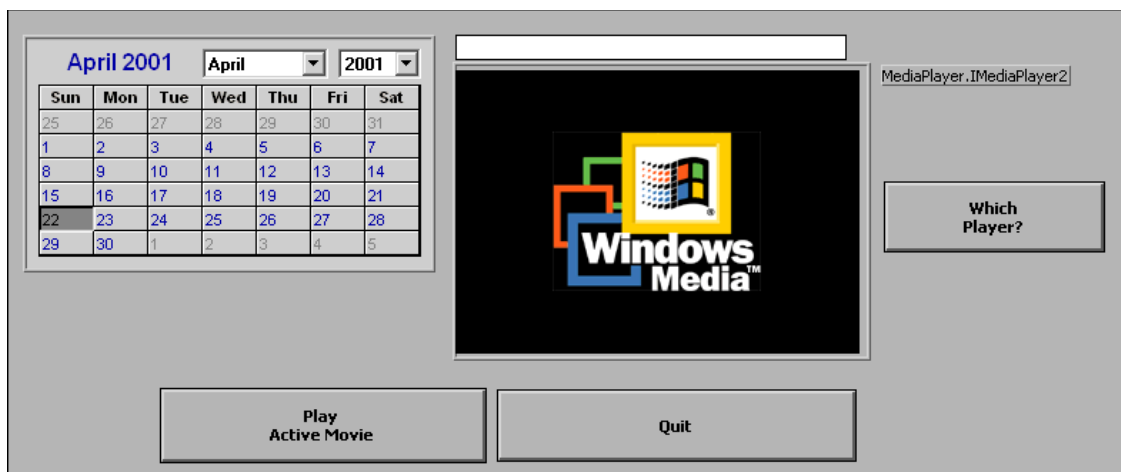
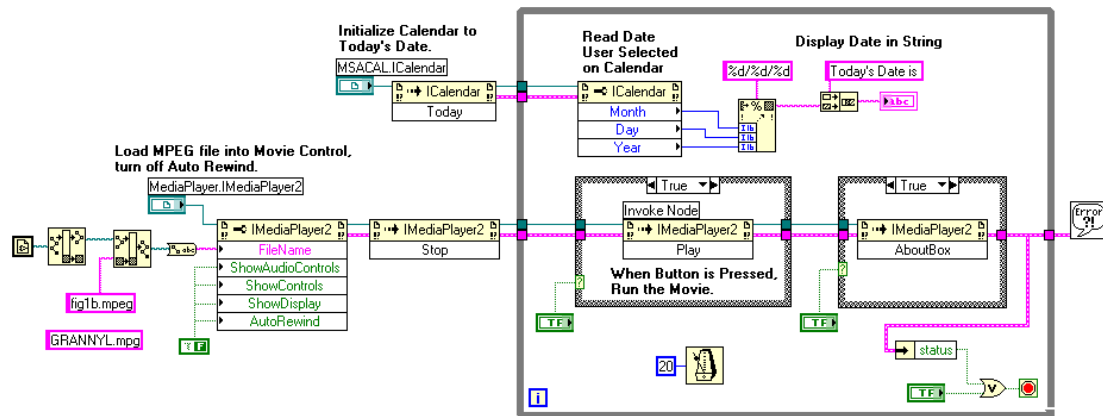
- **Create Object from File**—Select a document from a file located anywhere on your file system. This object can either be linked to the file or can be statically copied into the panel.

- **Create Control**—Select a control registered on your system. An example of an ActiveX control could be the Microsoft Web browser. All controls have an automation interface that allows you to work with them programmatically.

- 建立文档—选择注册在计算机上的文档对象。如果选择链接式文档，即选择**Link to File**选项，你建立了一个文档的视图。文档的任何改变会反映到容器中，文档会随着前面板对象的更新而更新。如果选择嵌入式文档，即不选择**Link to File**选项，文档成为容器的一部分。在容器外的改变不会影响容器内文档。比如嵌入PowerPoint的MS Word文档。
- 从文件建立对象—选择文件系统上文档对象。可选择是链接式或嵌入式。
- 建立控件—选择注册在计算机上的控件对象，所有控件带有允许编程的自动操作接口。例如，下图是用ActiveX控件控制浏览器对象的例子：



又如，下图控制多个ActiveX对象的例子。



Lesson 9

Error Trapping Techniques

With complex applications, proper operation of the system depends on a lot more than the program itself. Proper operation is a result of many factors, such as the following:

- Good program design based on sound principles.
- Reliable hardware interfaces and drivers.
- Appropriate configuration of hardware, network, and operating system variables.
- Robust design to account for uncertainties.

在复杂的应用程序中，恰当的系统操作不只依赖程序本身，而是许多因素的结果，比如：

- 建立在合理原则上的好的程序设计。
- 可靠的硬件接口和驱动程序。
- 硬件，网络和操作系统变量的适当配置。
- 能解决不确定因素的健壮性设计。

Debugging—Requirements

Good applications are ones that not only perform well but are also easy to deploy and maintain. What makes them so is the fact that their form and function are simple and well-defined. For example, including a Stop button in an application with a For Loop provides the user with an easy abort option if the loop is taking too long to execute. This is an ease of use feature for the end user.

The following are some common questions to consider about the requirements of your application:

- Does your application have a well-defined user interface?
- What hardware and system resources does the application use?
- Can the application exclude others from using resources?
- Are all the components accepting and processing information?
- Do you have limits on memory use?
- Do you have limits on execution time?
- Do you have other domain-specific restrictions?

好的程序不仅性能良好而且易于配置和维护，这取决于形式和函数的简单而良好的设计。

考虑下列一些常见问题：

- 你的程序有良好的用户界面设计吗？
- 你的程序用了哪些硬件和系统资源？
- 程序能否排它地使用资源？
- 是否所有的组件接受和处理消息？
- 有没有限制内存使用？
- 有没有限制执行时间？
- 有没有其它特殊范围的限制？

Debugging—Behavior

You need to identify what part of the program is operating and what parts apparently do not. Subtle factors might cause a component to behave erroneously, even though the actual error might be in a different hidden component. The best way to find such hidden errors is to evaluate the system behavior with an open mind and record all aspects of the behavior.

The following are some common questions to consider when debugging the behavior of your application:

- What inputs, outputs, or other entities does the application have?
- Are you able to interact with them?
- What other applications or entities does your application call?
 - Hardware
 - Networks
 - Third-party programs or DLLs
- How do other applications run?

你需要确定程序哪部分正在操作，哪部分没有。细微的因素可能造成组件错误的运转，即使实际的错误可能在一个不同的隐藏组件中。找出隐藏错误的最好办法是用开放的头脑和记录每个方面的方法评估系统运转情况。

考虑下列一些常见问题：

- 程序有哪些输入输出或其它实体？
- 你的程序调用了其它哪些程序或实体？硬件，网络，第三方程序或DLLs？
- 其它程序运转得如何？

Debugging—State

In addition to the basic behavior of the system in terms of whether all components seem to be working, you should check the state of the system to verify program correctness.

The following are some common questions to consider when debugging the state of your application:

- Do all variables have appropriate values?
- Are there hidden variables, and do they have correct values?
- Is the application executing the appropriate code segments?
- Does the system have the correct IP address, device or channel number, and so on?

除了根据是否所有组件看上去在工作来判断系统的基本运转情况，你应该检查系统的状态来检验程序的正确性。

考虑下列一些常见问题：

- 所有的变量都拥有适当的值吗？
- 有没有隐藏变量？它们的值正确吗？
- 程序正在执行适当的代码段吗？
- 系统的IP地址，设备或通道号等等正确吗？

Debugging—Resolution

Errors that are hard to catch manifest themselves in similar circumstances,

such as the following:

- When a certain combination of components execute.
- When a certain resource is in a particular state.
- As an incorrect sequence of events over time, although each single event might be within the bounds of normal operation of the application.
- As side effects of fixes for other, oftentimes simple, errors.

The following are some effective strategies to take when resolving bugs in your application:

- Look for discrepancies between requirements and actual behavior or state of the application.
- Look for evolution of the behavior and the state over time—does this satisfy the requirements?
- Make intelligent guesses and learn from mistakes.

难以抓住的错误会出现在相似的情况下，如下：

- 当某个组件组合在执行。
- 当某个资源处于特别的状态。
- 随时间推移，事件的顺序错误，尽管每一单独事件是在正常操作的范围内。
- 其它通常简单的错误所导致的副作用。

下列是一些纠错的有效策略：

- 寻找需求和实际运转或状态之间的差异。
- 随时间推移，寻找实际运转或状态产生的变化—这样满足需求吗？
- 做出智慧的猜测，并从错误中学习。

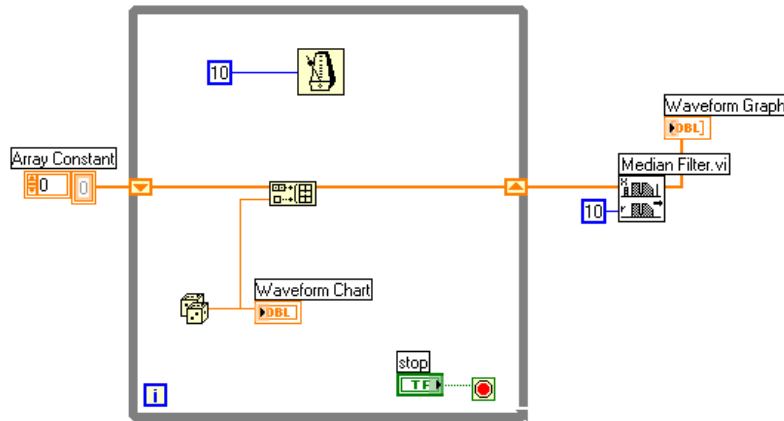
Memory Leak

Memory leaks are primarily an issue in C/C++ and lower level languages, while Java has built-in garbage collection which is the process of automatically finding such memory blocks and recycling them. Typically, memory leaks can be an issue in applications with infinite loops (such as user interface loops), small memory (such as embedded computing), critical performance requirements (such as real time), and so on. System resource leaks can lead to hangs and crashes due to non-availability of resources to certain applications and can be dangerous in factory floor type operations.

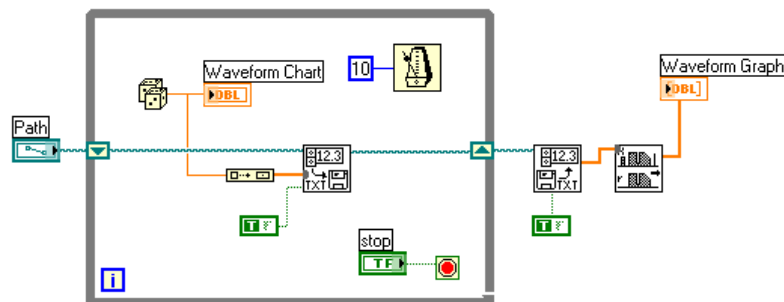
LabVIEW has been tested and is free of leaks. However, a LabVIEW VI can still display behaviors that act as leaks. In cases where a lower level application is being used alongside LabVIEW, that application could also be prone to leaks.

内存泄露主要是C/C++或低级语言的问题，而Java内建垃圾收集功能，就是自动寻找并回收内存块。典型的，内存泄露是那些带有无限循环(如用户界面循环)，小内存(如嵌入式计算)，严格的性能要求(如实时)，等应用程序的问题。由于无效资源导致的系统资源泄露可能导致当机和崩溃，对工厂操作是危险的。LabVIEW经过了测试不会泄露。但是，当低级程序用于LabVIEW时，VI仍可能表现出泄露的行为。

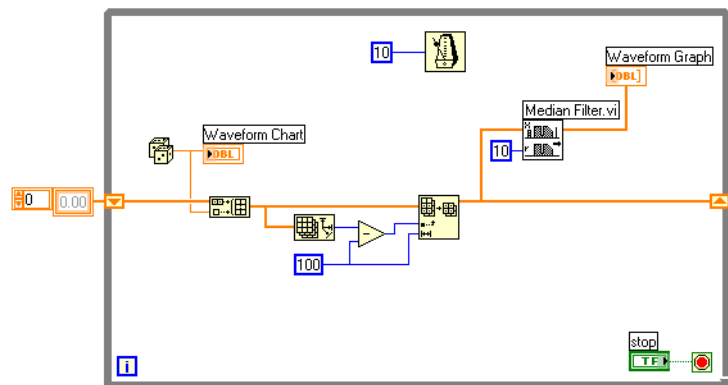
举个例子：



此图非常消耗内存。



做了改进，但却大量消耗磁盘空间。



这里只保存和处理一部分数据，
算是一个解决办法。

Multithreading Errors

Deadlock occurs when two or more processes are unable to complete because they are competing for the same system resource(s). An example of deadlock can be two applications that need to print a file. The first application, running on thread 1, acquires the file and locks it from other applications. The second application, in thread 2, does the

same with the printer. In a non-preemptive environment, where the operating system does not intervene and free a resource, both applications wait for the other to release a resource while neither one releases the resource they already hold.

在非抢先式环境中，操作系统不干预和释放资源，多个程序争夺同一系统资源，如果都不退让就会导致死锁。

Priority inversion occurs when lower priority threads seem to be getting more CPU time than higher priority threads. This typically happens because the higher priority thread might be trying to acquire a resource held by a lower priority thread that has been blocked from executing by another thread.

当低优先级线程比高优先级线程得到更多CPU时间，即如果一个其它线程阻塞了持有资源的低优先级线程，而这资源又是高优先级线程所需要的，就发生优先级倒置。

Thread starvation occurs when a thread does not seem to be getting any CPU time for an indefinite amount of time. For example, a lower priority thread, such as the user interface thread, might become unresponsive due to time critical threads consuming all the CPU time.

如果线程在不确定的时间内得不到任何CPU时间就会发生线程饥饿。例如，由于时间严格的线程消耗了所有的CPU时间，那么低优先级线程(如用户界面线程)就会无响应。

Thread problems typically occur in multithreaded systems with multiple priority levels and mutual exclusions(semaphores in LabVIEW). Semaphores need not be in the application—resources such as ports and hardware can also behave in the same manner. For example, serial ports acquired by other applications or DLLs might block other applications.

In most general purpose programming languages, a simple deterministic scheduling algorithm is used to accommodate a large variety of user tasks. So it is up to the user to allocate priorities and threads. In LabVIEW, priority scheduling and allocation of multiple threads is done by the run-time engine. The run-time engine looks at the state of the application to determine which threads are ready to run, which resources are available, and when the processor is available. Based on these, it specifies to each task in each thread when it will get to run.

当多优先级多线程系统带有互斥体(如LabVIEW中的Semaphores)以及端口和硬件等资源时就会发生线程问题。比如，某程序或DLL获得串口，就会阻塞其它程序。

多数程序语言，简单确定的时序安排算法是为了容纳大量的用户任务。所以它由用户分配优先权和线程。在LabVIEW中优先级安排和线程分配由run-time engine完成，它查看应用程序状态来决定哪个线程准备好运行，哪些资源可用，何时处理器可用。基于此，它指定每个任务到每个将执行的线程中。

Using Configuration Files

While not an error trapping technique by itself, using configuration files can be a very useful feature when building large applications that need to be customized when deploying on multiple target computers.

The following are some cases where a configuration file can be useful:

- In programs that depend on a large number of support files distributed

in multiple folders. Configuration files ensure that the support files are accessed from the correct location. It is easier to ensure the accuracy of one configuration file, rather than searching multiple directories.

- When you need to specify well-defined, tested values for controls so that the program always starts from the same state and behaves deterministically.
- In cases where a lot of data is required from the user, it might be easier to provide one centralized location where the data can be entered. This reduces the number of user dialog boxes and times files are read within the program. However, there might be cases where the reverse method of letting the user pick from menus is preferable.
- User names and passwords, and certain parameters that you might want to hide from the user, or that the user might not be aware of are best set in a configuration file.

当构建配置在多个计算机上，并需要定制的大型应用程序时，如果自身没有捕获错误的技术，那么使用配置文件是种很有用的方法。下列情况配置文件很有用：

- 程序依赖大量分布在多个文件夹中的支持文件。配置文件确保支持文件从正确的位置被访问。维护配置文件的正确性要比在多个文件夹中搜索要容易。
- 当你需要为控件特别设计的测试过数据，这样程序总能从同一个状态开始，有着确定的运转。
- 当需要从用户处获取大量数据时，很容易提供一个可以集中输入数据的位置。这减少了用户对话框的数量和读取文件的次数。然而，某些情况下让用户从菜单选取的做法才是更可取的。
- 用户名及密码，和一些你可能要对用户隐藏的或用户不关心的参数，最好放入配置文件。

——全文完

LabVIEW™ 高级性能和通讯课程手册(摘译) 2004.12. (第一版)
mebusw@163.com