

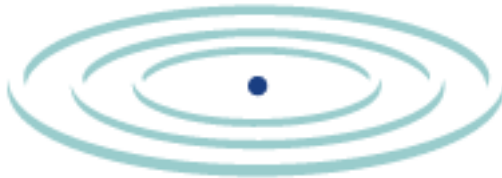
VI User Guide

Working in LabVIEW

In this section you will find a detailed explanation of the concepts and techniques you need to understand and be able to use in order to successfully and efficiently publish your instrumentation system. This section begins with an overview of the process of communication between your remote instrument and your local instrument running in your local LabVIEW system. You will need this knowledge in order to understand what communication tasks are handled automatically by AppletVIEW and what you must program yourself. The second part of this section takes you through the MAIN techniques you need to master in LabVIEW to prepare your instruments for the Web. The third part of this section explains, in detail, how you wire your VIs using the AppletVIEW sub-VIs.

AppletVIEW enables LabVIEW users to:

- * use standard web browsers (no plug-ins, modifications, or LabVIEW run-time necessary) to control and monitor remote LabVIEW applications
- * integrate virtual instrumentation systems with standard web servers and services, allowing construction of powerful web sites for virtual instrumentation systems
- * use XML as a data repository and description of virtual instrumentation systems
- * use nearly any Java Bean as a control or indicator to a remote LabVIEW application
- * integrate Java applications with LabVIEW



Importing the AppletVIEW Menus into LabVIEW

In the AppletVIEW installation directory, the **labview6** subdirectory has four *.mnu files. These are files LabVIEW uses to add AppletVIEW to the Function Palette To install the icons onto your LabVIEW function palette:

- open a VI.
- open the diagram for the VI (Ctrl-E).
- on the Functions palette, select "User Libraries".
- on the User Libraries palette, click the "Options" button.
- the *Function Browser Options* dialog appears. click "Edit Palettes".
- wait for a bunch of flashing windows.
- move the Functions and Controls palettes around until you can find the User Libraries palette again underneath them.
- right click blank space in the palette and choose Insert -> Sub Menu.
- choose "Link to an existing .mnu file".
- press "OK".
- navigate to the AppletVIEW <installdir>/labview6 directory and choose the AppletVIEW.mnu file.
- press "Save Changes".
- wait for a bunch of flashing windows.

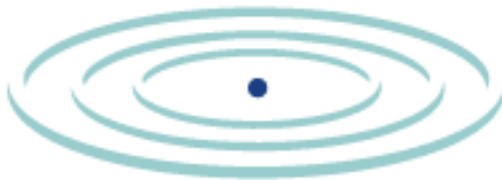
Now, when on a LabVIEW diagram, you can press the "User Libraries" button on the Tools palette to get to the AppletVIEW VI's.

Overview of Communication Between a Local VI and Remote Applet

Before you can begin to effectively wire your LabVIEW programs with AppletVIEW it is helpful to understand the complete process of communication between your local and remote VIs over the Web through the browser.

The communication process between LabVIEW or LabWindows and a remote instrument is fairly transparent once it is established, but it is a more complex "under the hood". All of this communication process is handled automatically by AppletVIEW. You need to understand this process in order to effectively wire the AppletVIEW sub-VIs into your LabVIEW programs.

Remember that your remote instrument is a configuration file (.jvi file) that you created and saved with Applet Builder and that this configuration file is referenced by an <APPLET> tag inserted into an HTML page. This page is then served from a Web (HTTP) server.



Establishing the Connection

The following figure illustrates the establishment of a communication connection between the AppletVIEW instrumentation libraries and a Java applet:

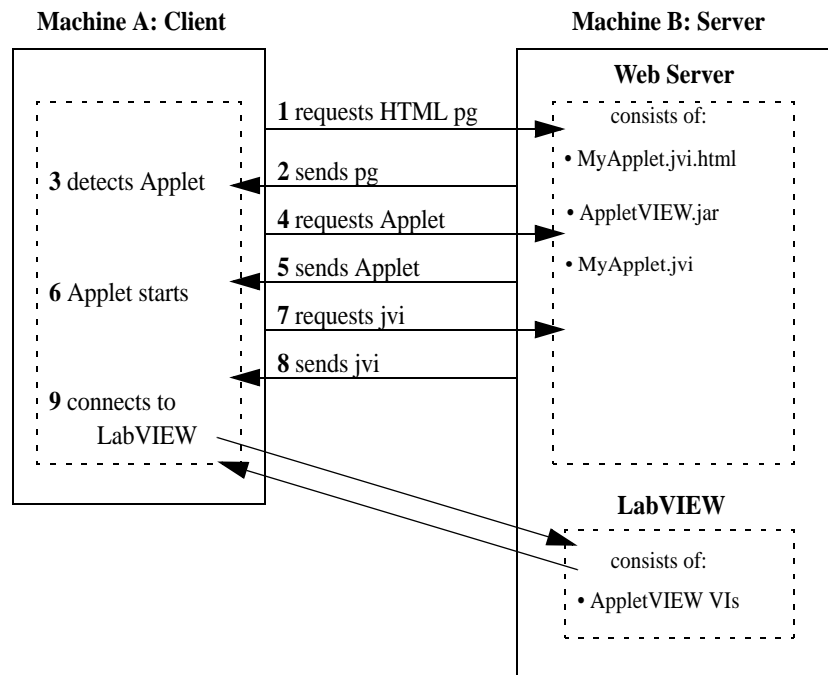


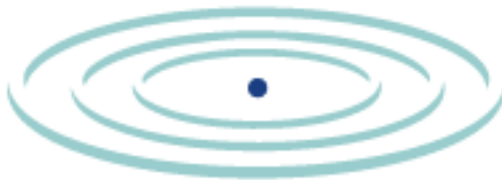
Figure 7. Communication process between an AppletVIEW applet and LabVIEW server

1-2. When a client requests an HTML page with an embedded applet, "MyApplet.jvi.html" for instance, the Web server pushes up this page to the requesting browser.

3-4. On the browser, the <APPLET> tag in the HTML specifies that a Java applet is to be loaded, so the Web browser's Java Virtual Machine (VM) starts and requests the server to serve the Java class files to go with the applet. The Java classes it requests are in the JAR file "AppletVIEW.jar"

5. The server pushes up AppletVIEW.jar to the Web browser (this process takes the longest, but is only needed once per session).

6-7. Once AppletVIEW.jar is loaded, the applet begins running. The applet's first step is to request the file specified in the parameter "ConfigFile", which in this example is called "MyApplet.jvi".



At this point the status LED (upper left corner of the applet) appears and turns yellow.

8. The binary applet configuration file (MyApplet.jvi), created previously with Applet Builder, is sent to the applet on the Web client.

9. The applet attempts a TCP/IP socket connection over the data port that was specified in Applet Builder (default is 4747). If LabVIEW is running with the "Create Applet Listener.vi" running, it will accept the TCP/IP connection from the applet, and communication begins. If the connection is successful, the status LED on the remote VI turns green. If communication fails or is interrupted, the status LED turns red.

Handshaking

Once LabVIEW and the applet make a connection, some handshaking ensues, and, if specified in LabVIEW, an AppletID verification is performed. After these initial steps the connection stays open as long as the Web browser is open and LabVIEW does not kill the connection. The Web server (port 80) is free to listen for other requests. If a new client request comes in to LabVIEW with the same applet, it will not connect to LabVIEW. Instead, it must wait until the previous client has disconnected.

In order for this process of communication to initiate and continue successfully you need to make sure your LabVIEW VIs have been set up properly and are running before a client attempts to load an applet. This is the topic of the next section.

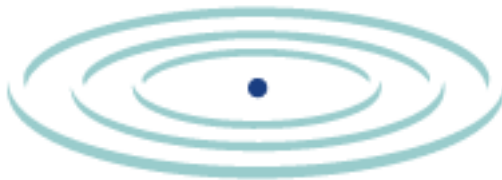
Communicating with an Applet from LabVIEW

AppletVIEW handles the low level communication between your local and remote instruments. You only need to specify what data you are reading/writing and which component(s) on the applet you are communicating with. Communicating with the applet means reading data from a component and writing data to a component.

Writing to components on the applet is straightforward: you can use the AppletVIEW VI, Write Applet, to update the value of any specific component. You can also update the values of multiple components simultaneously using the Write Multiple Components to Applet VI. (These VIs, and the suite of supporting VIs, are described later in this section.)

Reading from components has more implications. An applet can be in one of two modes: either in event mode or poll mode.

In Event mode, the default mode for all applets, anytime a value changes on a writable component (e.g., someone turns a knob on the applet), the



new value is sent over the data port. An AppletVIEW VI (Read Applet) can read these values as they arrive over the network. In event mode, the VI must be called repeatedly in a loop to "capture" the events from the applet and remove them from the TCP/IP buffer.

In Poll mode no data is ever initiated by the applet. When a value on a component changes, the value is simply recorded in the data structure of the applet. It is never automatically sent. Instead, it is up to the LabVIEW program to decide when and what component to poll (read from).

You can switch between poll and event modes at any time using AppletVIEW.

Table 3 Comparison of Event Mode and Poll Mode

Event Mode	Poll Mode
<p>Advantages:</p> <ul style="list-style-type: none">•Events on the applet can be captured as soon as they occur•Useful for real-time synchronization between LabVIEW and the applet•Less bandwidth <p>Disadvantages:</p> <ul style="list-style-type: none">•The Read Applet VI must be continuously called in a loop to capture events•Too many successive events can pile up and take a long time to get through the queue in the TCP/IP buffer•No way of knowing ahead of time which component was changed; more complex case structures required in LabVIEW	<p>Advantages:</p> <ul style="list-style-type: none">•No data is ever initiated by the client; thus no data piles up on the TCP/IP buffer•Specific components can be monitored while ignoring others•Less complex LabVIEW code <p>Disadvantages:</p> <ul style="list-style-type: none">•Not suitable for real-time synchronization with client data input•More bandwidth used•Possible undetected events between polled values

AppletVIEW VIs

The following listing is a summary of the top-level VIs provided with AppletVIEW. For a detailed description of all the inputs and outputs of these VIs, see the Reference section.

Open Connection To Server:

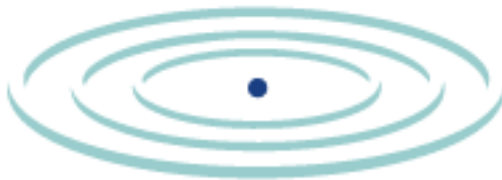
establishes a communication session with the AppletVIEW Server

Create Applet Listener:

creates a listener for clients on a specified TCP/IP port

Wait on Applet Connection:

waits a specified amount of time for a client to connect and establish a communication session



Poll/Event Mode:

switches the applet between polling mode and event-driven mode for reading values

Read Applet:

reads values from the communication session when applet is configured in event mode

Poll Applet:

polls the communication session for the current value of the selected component.

Write Applet:

writes a value to a selected component on the applet

Write Multiple Components to Applet:

writes values to several selected components simultaneously

Ping Applet:

pings the communication session to find out if it is still connected

Close Applet:

closes the communication session

Handling Applet Connections

With an understanding of the low level communication between local and remote instruments, an understanding of the general methods of reading and writing applets (including the implications of event VIs poll mode) and a basic familiarity with the top level AppletVIEW VIs, it is possible to describe the process of wiring a VI for communication.

Setting up an instrument to communicate with AppletVIEW involves making it either a communication client or server.

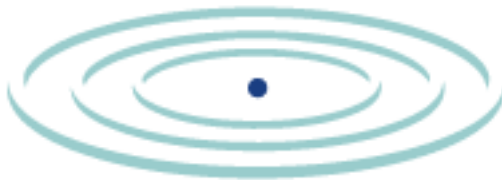
If the application is a client, the following step is required:

- Use Open Connection To Server to establish the communication session

If the application is a server, the following steps are required:

- Create a listener for the applet with Create Applet Listener.
- Wait for an applet client to connect with Wait for Applet Connection.

Each of these two scenarios is explained below.



Open Connection to Server.vi

Open Connection To Server is used when your LabVIEW application is a communication client - it initiates the communication session. This is useful when you are developing your application and expect to be stopping it frequently.

Create Applet Listener.vi

Create Applet Listener is used when your LabVIEW application is a communication server - it waits for another application to initiate the communication session. This is useful when you want to be able to connect to the application from a variety of sources.

The following diagram illustrates a simple sequence of the Create Applet Listener and Wait For Connection VIs.

[pic]

Figure 8. AppletVIEW VIs

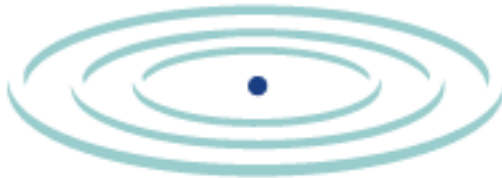
Setting up the Listener

The first step to setting up your LabVIEW server to handle incoming applet connections is to create a TCP/IP listener on the same TCP/IP port that the incoming applets will connect to. By default this port is port 4747. You do this by calling the Create Applet Listener VI. This VI should only be called once every time LabVIEW is run, since the listener remains active as long as LabVIEW is running. If a client disconnects, you should not call this VI again, but call Wait on Applet Connection instead. If you stop LabVIEW, however, you must call this VI again to re-enable the listener.

Wait on Applet Connection.vi

The second step in the process of communicating with an applet is to wait for an applet connection. You do this by calling Wait On Applet Connection. You can either specify a time-out or wait indefinitely (the default is to wait indefinitely, -1). This VI will not exit until the applet connects or the time-out has passed. If you detect a client disconnecting, you can call this VI again to wait for the next client session. Once a client is connected, this VI creates a LabVIEW TCP/IP network refnum that is stored in a global variable and is used by the other AppletVIEW VIs. This VI returns the remote IP address of the client.

Once a client is connected, you can begin reading and writing to the applet's components using the Read Applet and Write Applet VIs. To terminate a session with a client in a "clean" fashion, you should call "Close Applet Connection".



Writing to and Reading from an Applet in LabVIEW

Writing to the Applet

Writing to the applet is straightforward. Every component on the applet has been assigned a componentID, and every component can change its value (or in the case of images, other properties as well) in response to a command from LabVIEW over the network.

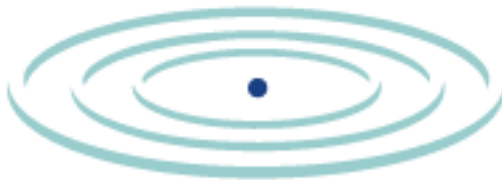
The Write Applet VI writes to a single, specific component. You must specify the componentID of the applet component you are writing to. The Write Applet VI has four data inputs, each of which matches one of the four datatypes (int32, SGL, Boolean, string). You should wire the data to be sent to one of these inputs, matching the data type of the applet component. For example, if you are writing to a chart, you would use the data (SGL) input. If you are writing to a switch, use the data (Boolean) input.

Always make sure the LabVIEW datatypes you send to an applet are of type int32, SGL, Boolean, or string. These are the only four datatypes that AppletVIEW applets handle.

The Write Applet VI allows you to write to one specific component, but there are situations when it may be efficient to update multiple components simultaneously. For example, you may want to update an array of LEDs. You can do this without calling the Write Applet function over and over by using Write Multiple Components to Applet. Because this VI can update multiple components of different datatypes, you must flatten all your data first into a binary string and provide the VI with LabVIEW type descriptors so it knows what type of data you are sending to each component.

For example, let's say you want to write to an LED with componentID 4, and a switch with componentID 2, and a slider with componentID 7. The steps to use the Write Multiple Components to Applet VI are as follows:

- First, make an array of the componentIDs you are updating. (In this example you would make an array [4, 2, 7]).
- Then, use the Build Cluster function to group your data sources together. You must cluster them in the same order as the componentIDs in the array. (In this example, the first cluster element must be the LED data source, the second element the switch, etc.).
- Then use the Flatten To String function (from the Advanced palette) to flatten the data. Connect the outputs of this LabVIEW function, "type descriptors" array and "flattened data" string to the Write Multiple Components to Applet VI.



[pic]

Figure 9. Writing Multiple Components to Applet

Writing data to an applet is straightforward. Reading data has more implications.

Reading data from the Applet into LabVIEW

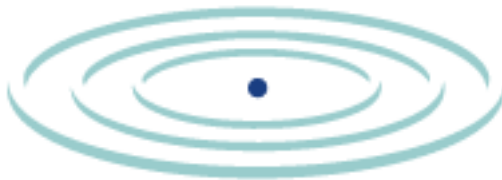
The Java applets created with AppletVIEW will send data every time a component changes its value because of user interaction (for example, flipping a switch). The applet sends the componentID, the data type, and the data value to LabVIEW. This is the event mode discussed above. It's possible to "turn off" the applet, stopping it from automatically sending data, by changing the applet from event mode to poll mode, using the Poll/Event Mode VI. This VI can be called anytime when the applet is connected.

In poll mode, data is only sent upon request from LabVIEW--this is referred to as "polling a component".

The Read Applet VI receives all incoming data from the applet (assuming it has not been set into poll mode). Therefore, you should normally call Read Applet repeatedly in a While Loop to retrieve data sent by the applet. This VI returns only one component's value at a time: it returns the componentID of the component that has sent data, and it returns the component's data in one of the four data outputs (int32, SGL, Boolean, string). The data is returned in the output that matches the applet component's data type. If you do not know or remember what data type a particular componentID is, you can always check it with the data type output on this VI. You can also poll an applet for its components values. You can poll one, several, or all of the components at a time. To do this, you can first set the applet in poll mode by calling Poll/Event Mode VI (although this is not strictly necessary, since you can poll an applet at any time in either mode), then call Poll Applet.

You can poll the entire applet with all of its components by setting "poll all components" to TRUE (default). In this case, you do not have to specify any componentIDs, since all of them are returned.

To poll specific components, you give this VI the array of componentIDs you are polling. The data values are returned in an array of clusters. Each element corresponds to a componentID, and each cluster is a set of the four datatypes (int32, SGL, Boolean, string), one of which contains the corresponding value.



Additional Tips Here are some additional tips for using the AppletVIEW VIs:

- If a client becomes disconnected, you don't have to run Wait for Applet Connection again. You can set the input "Wait for Connection?" on the Read Applet and Write Applet VIs to TRUE, and these VIs will wait for a connection.
- Always make sure your LabVIEW datatypes that are sent to an applet are of type int32, SGL, Boolean, or string. These are the only four types that AppletVIEW applets handle.
- If a Web client disconnects, you normally will get an error in the output error cluster. However, the TCP/IP functions do not always return an error if a client disconnects, so you can also check if a client is still alive with Ping Applet
- Note that there are no refnums or other variables to pass between the communication applets. This information is stored in a global variable. However, it is recommended you always wire the error clusters between the AppletVIEW VIs, and handle errors appropriately.
- You can turn off the applet sending values every time a component's value changes by switching from event mode to poll mode, using the Poll/Event Mode VI. When you are in poll mode, the applet will not send any values, and you can use the Poll Applet VI to request the value of a specific component. You can still use the Poll Applet VI in event mode.
- If you are updating several components at a time, it is more efficient to use Write Multiple Components to Applet than call several instances of Write Applet

The above discussion is the general process of handling communication between your local and remote instruments. In the next section you will find a number of examples of actually working in LabVIEW. See these examples or the Reference section for more information.

LabVIEW Examples

All of the examples you opened with Applet Builder above can communicate with a LabVIEW server. Each of the following examples shows you some variety in how you can use AppletVIEW VIs.