Using Apple Events and the PPC Toolbox to Communicate with LabVIEW™ Applications on the Macintosh

Introduction

This application note describes how to use Apple Events, which are high-level message-based methods of interapplication communication (IAC), and the program-to-program communications (PPC) Toolbox, which is a lower level, stream-oriented form of IAC.

Both of these forms of communication work only on Apple Macintosh systems running Mac OS 7 or later, and both require you to enable Program Linking in the File Sharing Control Panel.

Apple Events

Apple Events are message-based forms of IAC. The communication between the applications occurs as discrete messages with a highly structured internal format defined by Apple. There are a large number of standardized Apple Events for requesting operations that are common to many applications, such as those that work with text, e-mail, or graphics. The Mac OS 8.5 Registry, available in the AppleScript 1.3.4 SDK (**http://www.apple.com/developer**), describes the vocabulary of these messages in detail.

Applications can use Apple Events in two different ways – by responding to Apple Events and by sending Apple Events. Most applications written for Mac OS 7 and later, including LabVIEW, respond to at least the core set of Apple Events. The core set of Apple Events includes Open Document, Print Document, and Quit Applications. Many go further and respond to one or more of the standardized suites of Apple Events or to custom suites that apply only to that application. Applications that respond to Apple Events are generally called scriptable.

Applications that can send Apple Events are less common, although the Mac OS includes one called Script Editor. Other examples of scripting applications include Scripter (**http://www.mainevent.com/**) and FaceSpan (**http://www.facespan.com**). These scripting applications typically use the AppleScript language, although there are other scripting languages available, such as UserLand Frontier (**http://www.scripting.com/frontier5/**).

Apple Events in LabVIEW

LabVIEW uses Apple Events by responding to the core set of Apple Events and to a few LabVIEW-specific Apple Events. LabVIEW also provides easy-to-use VIs with which you can send several standard Apple Events (Open Document, Print Document, and Quit Applications) and all the LabVIEW custom Apple Events. You can use lower level VIs to send arbitrary Apple Events to other applications. The low level VIs are located on the **Communication»AppleEvent»Low Level Apple Events** palette.

LabVIEW-Specific Apple Events

LabVIEW also responds to the following LabVIEW-specific Apple Events – Run VI, Abort VI, VI Active?, and Close VI. With these events and the Open Documents AppleEvent, you can use other applications to programmatically tell LabVIEW to open a VI, run it, and close it when finished. A thorough understanding of Apple Events, as described in *Inside Macintosh*, *Volume VI*, and the Mac OS 8.5 Registry, available in the AppleScript 1.3.4 SDK (**http://www.apple.com/developer**), is a prerequisite for sending these Apple Events to LabVIEW from other applications. You can send these events between two or more LabVIEW applications by using the utility VIs.

Refer to *LabVIEW Help* for more information about the LabVIEW-specific Apple Events.

Using Apple Events to Communicate with Applications

The following list provides some tips for communication:

- Use the VI Server to communicate between a LabVIEW block diagram and LabVIEW itself, and to communicate between one LabVIEW block diagram and another instance of LabVIEW over a TCP/IP network. Instead of using Apple Events, use the VI Server functions, such as Open VI Reference, and the properties and methods available through the Property node and Invoke node.
- When communicating with LabVIEW from another Mac OS application, which is usually an AppleScript, send LabVIEW one of the eight Apple Events it understands. Refer to the Apple Events in LabVIEW section for a list of the Apple Events.
- When communicating with another Mac OS application from LabVIEW, use the Apple event VIs in vi.lib.
- When using high-speed, low-overhead communication between two LabVIEW applications on an AppleTalk-only network, use the PPC Toolbox. You must develop your own message protocol, but in most instances, this is not difficult. Refer to the PPC Toolbox section for more information.
- If you want to send an Apple event for which LabVIEW provides no VI to another application, use the low-level AESend VI. The **AppleEvent** palette also contains VIs you can use to create an Apple Event. However, creating and sending an Apple event at this level requires a detailed understanding of Apple Events as described in the Mac OS 8.5 Registry and the Creating Apple Event Parameters section of this application note.

Complete the following steps to communicate with an application using Apple Events.

Step 1. Determine if the Application Uses Apple Events

If the target application is not LabVIEW, use the PPC Browser VI to determine if the target application uses Apple Events. Run the VI and select the computer that is running the target application. The Prompt dialog box lists the applications on the computer that support Apple Events.

Step 2. Generate a Target ID for the Client

If the application uses Apple Events, call an AppleEvent VI with the target ID for the application. A target ID is a cluster that describes a target location on the network (zone, server, and supporting application). To generate a target ID, use one of the following VIs:

- Use the Get Target ID VI to programmatically create a target ID based on the application name and network location.
- Use the PPC Browser VI to create a target ID by selecting one from the list of applications that support Apple Events.

Step 3. Ensure That the Application is Running

To send a message to an application, you must first ensure that the application is running. If the application is not running, launch it programmatically using the AESend Finder Open VI, which sends a message to the Finder, to open a particular document.

If the application is on a remote computer, use the AESend Finder Open VI to indicate the network zone and the server name of the computer with which you want to communicate. The defaults for **Zone containing Finder** and **Server containing Finder** are the current computer.

Step 4. Send Messages to the Application

Use the LabVIEW VIs to send messages to the target application. If you send messages to an application on another computer, the operating system prompts you for your user name and password for the receiving computer.

You cannot avoid this prompt because it is built in to the operating system. This prompt can cause problems if you want the application to run on an unattended computer. When you design VIs that use IAC, remember that the Apple Event Manager does not include a programmatic method for bypassing the **User Identity** dialog box. For example, you cannot instruct an unattended remote computer to send an Apple Event to a third computer. Someone must enter user information into the **User Identity** dialog box that appears on the remote computer.

The PPC VIs allow unauthenticated sessions if guest access is enabled on the computer with which you need to communicate. For this reason, you may find the PPC VIs more useful for LabVIEW-to-LabVIEW communication that does not require authentication. The Apple Event Manager is the part of the operating system that implements Apple Events.

PPC Toolbox

The PPC Toolbox is a fairly low-level, stream-oriented form of IAC that is based on even lower level AppleTalk protocol. No structure is imposed on the data passed through the PPC Toolbox. The PPC Toolbox provides only a reliable byte-stream "pipe" by which the applications can efficiently send and receive arbitrary data of arbitrary length. The cooperating applications determine the format of the data.

The stream-oriented nature of the PPC Toolbox is very similar to TCP/IP. Writing a client and server application using the PPC Toolbox involves most of the same tasks as when using TCP/IP. The PPC Toolbox has similar performance characteristics as well.

A unique feature of the PPC Toolbox is that opening a PPC Toolbox port makes the name of that port visible to any other process on the AppleTalk network using an AppleTalk protocol called Name Binding Protocol (NBP). Opening a PPC Toolbox allows programs that use the **Apple standard PPC Toolbox Browser** dialog box, available in LabVIEW with the PPC Browser VI, to see the program name and select the program as a communication target.

Another useful feature of the PPC Toolbox is that the programmer can determine the logical end of a block of data without encoding that information in the data itself.

LabVIEW provides VIs for you to create PPC Toolbox servers and clients. PPC Toolbox Servers are programs that passively accept connections. PPC Toolbox clients are programs that actively connect to servers.

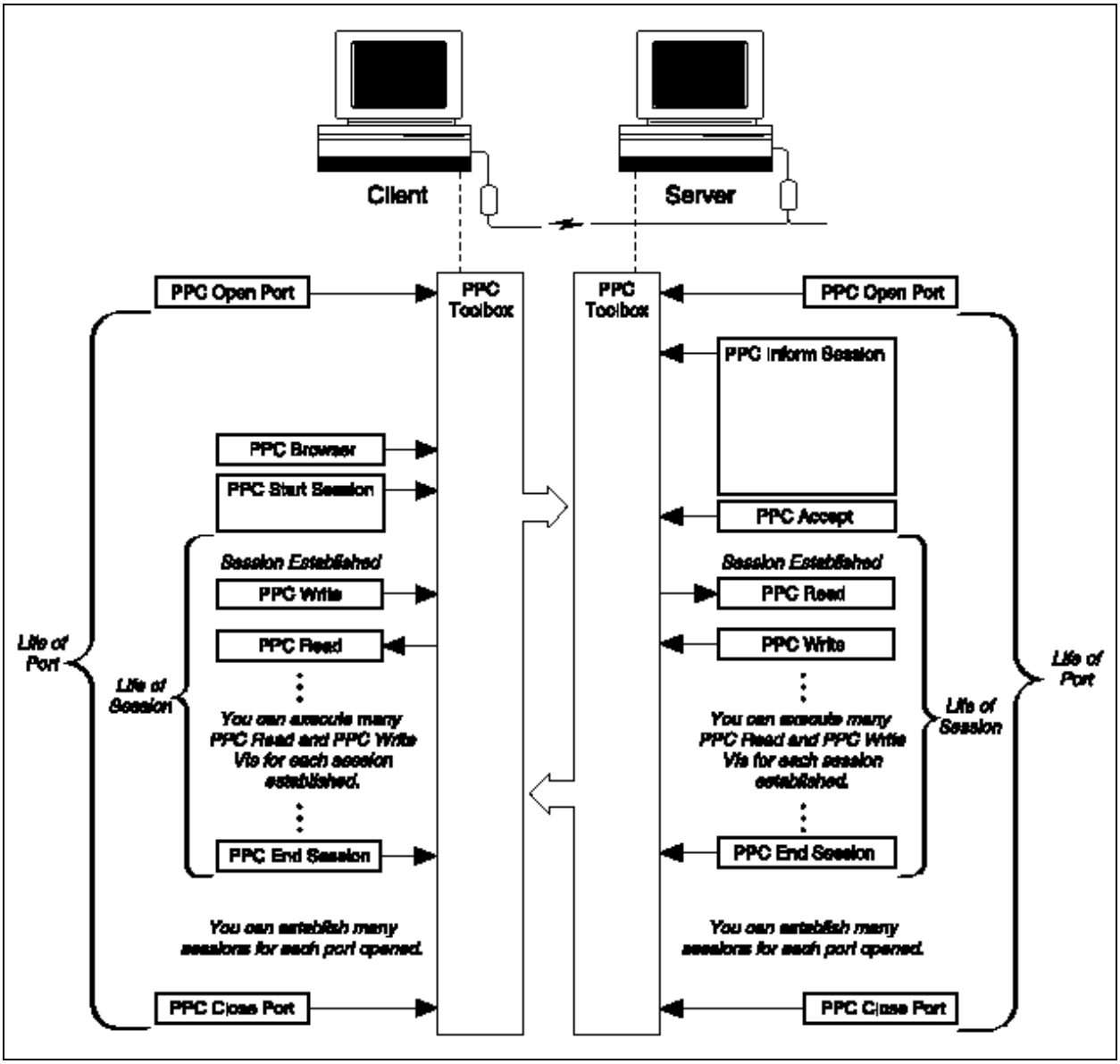
Communication between PPC Toolbox Clients and Servers

To communicate using the PPC Toolbox, the client and the server must use the PPC Open Port VI to open ports for subsequent communication. Ports are given names, and you can use these names to distinguish between different services. An application can use more than one port at a time to have access to multiple services. Each port, in turn, can handle multiple simultaneous sessions, or conversations.

A server announces its readiness to accept sessions by calling the PPC Inform Session VI. The server can choose to automatically accept all sessions or can retrieve information about the requesting client before deciding to accept or reject the session. Once it accepts or rejects a session, the server must call the PPC Inform Session VI again to accept subsequent sessions. A client opens a session with a server using the PPC Start Session VI.

Once a session is started, the client and server use the PPC Read and PPC Write VIs to transfer data.

The following illustration shows the general flow of operations between a server and client using the PPC Toolbox.



Building a PPC Client

Complete the following steps to communicate between PPC Toolbox clients and servers.

Step 1. Generate a Target ID for the Server

You communicate with the PPC server using a target ID. A target ID is a cluster that describes the server. To generate a target ID, use one of the following VIs:

- Use the Get Target ID VI to programmatically create a target ID based on the application name and network location.
- Use the PPC Toolbox Browser VI to create a target ID by selecting one from the list of applications that support Apple Events.

Step 2. Open a Port and Start a Session

Use the PPC Toolbox Open Port and PPC Toolbox Start Session VIs to open a connection to the server and start a session. These VIs return a port refnum and a session refnum, which you use to communicate with the server.

Step 3. Send a Request to the Server

To send a request to the server, use the PPC Write VI. Remember that you control the format of the requests. If you must send a command with multiple writes, you should set the **more** parameter on the PPC Write VI to TRUE on all writes except the last one. Set the **more** parameter to FALSE on the last write to signify that no more data is associated with this request.

If you are expecting a reply from the server, call the PPC Read VI repeatedly, and concatenate each block of data until the **more** parameter on the PPC Read VI returns FALSE.

Step 4. End the Session and Close the Port

Use the PPC Toolbox End Session and PPC Toolbox Close Port VIs to end the session and close the connection to the server.

Building a PPC Toolbox Server

Complete the following steps to use PPC Toolbox to fulfill each component of the general server model.

Step 1. Open a Port

During the initialization phase, use the PPC Open Port VI to open a communication port.

Step 2. Wait for a Connection

Use the PPC Inform Session VI to wait for a connection. With PPC Toolbox, you can automatically accept incoming connections. If you want to monitor connections, you can accept or reject the session by using the PPC Accept Session VI.

Step 3. Wait for a Command

When a session starts, you can read from that session to retrieve a command. As described in Step 3 of the Communication between PPC Toolbox Clients and Servers section, you must decide the format for commands. If you precede commands by a length field, you need to first read the length field and then read that amount of data.

Step 4. Respond to the Command

When finished, you pass the results to the next step.

Step 5. Return the Results

Use the PPC Toolbox Write VI to return the results. As described in Step 3 of the Communication between PPC Toolbox Clients and Servers section, the data must be in a format that the client accepts. Set the **more** parameter on the PPC Write VI to TRUE on all writes except the last one. Set the **more** parameter to FALSE on the last write to signify that no more data is associated with this request.

Step 6. End the Session and Close the Port

Use the PPC End Session and PPC Close Port VIs to end the session and close the connection to the client.

Deciding between Apple Events and PPC Toolbox

If you want to control a scripting application from LabVIEW, Apple Events are the only choice. If you want to control LabVIEW from another scripting application and have relatively simple needs, you also can use Apple Events.

If you want to control LabVIEW from within itself or control another LabVIEW application on a TCP/IP network from your own LabVIEW application, you use the VI Server functions. Refer to the *LabVIEW User Manual* and *LabVIEW Help* for more information about VI Server.

There are relatively few applications other than LabVIEW that use the PPC Toolbox. Igor Pro (**http://www.wavemetrics.com**) is one that accepts commands using a PPC Toolbox connection. The lack of compatible programs is a result of the lack of a standard format for the data that is transferred using this protocol.

If you want to communicate between LabVIEW applications, need high performance, and are willing to define your own data format for communicating between the applications, use the PPC Toolbox or TCP/IP.

The following list outlines the differences between PPC Toolbox and TCP/IP:

- TCP/IP has a simpler addressing mechanism, but the PPC Toolbox addressing mechanisms use browsing for services, for example, with the PPC Browser VI.
- The PPC Toolbox has a mechanism you can use to set a flag on any block of data transmitted to indicate whether there is more data to come. If you use TCP/IP, you must encode this information in the data itself.
- TCP/IP is available on almost every computer on the market, making it possible to communicate with applications that do not run on a Macintosh. The PPC Toolbox is limited to Macintosh only. Managing an all-Macintosh AppleTalk network is considerably easier than managing a TCP/IP network.

Constructing and Sending Other Apple Events

In addition to VIs that send common Apple Events, you can use lower-level VIs to send any Apple Event. Using these VIs requires more knowledge of Apple Events than using the VIs described in the Apple Events in LabVIEW section in this application note. If you use these VIs, you should be familiar with the discussion of Apple Events in AppleScript 1.3.4 SDK (**http://www.apple.com/developer**) and the Mac OS 8.5 Registry, available in the AppleScript 1.3.4 SDK (**http://www.apple.com/developer**).

When you send an Apple Event, you must include several pieces of information. The **event class** and **event ID** inputs identify the Apple Event you are sending. The **event class** is a four-letter code that identifies the Apple Event group. For example, an **event class** of `core` identifies an Apple Event as belonging to the set of core Apple Events. The **event ID** is another four-letter code that identifies the specific Apple Event that you want to send. For example, `odoc` is the four-letter code for the Open Documents Apple Event, one of the core Apple Events. To send an Apple Event using the AESend VI, combine the **event class** and **event ID** as an eight-character string. For example, to send the Open Documents Apple Event, pass the AESend VI the eight-character code `coreodoc` using the **event class** and **event ID** parameters.

If you send the Apple Event to another application, you must specify the **target ID** and **send options** inputs.

You also can specify an array of parameters if the target application needs additional information to execute the Apple Event. Because the data structure for Apple Event parameters is inconvenient for use in LabVIEW block diagrams, the AESend VI accepts these parameters as ASCII strings. These strings must conform to the syntax described in the next section. You can use this syntax to describe any Apple Event parameter. The AESend VI interprets this string to create the appropriate data structure for an Apple Event, and sends the event to the target you specify.

Creating Apple Event Parameters

In many cases, an Apple Event parameter is a single value. However, it can be quite complex, with a hierarchical structure that contains components that in turn can contain other components. In LabVIEW, a parameter is constructed as a string that has a simple syntax you use to describe many different kinds of data that an Apple Event parameter can be, including complex structures.

An Apple Event parameter string always begins with a keyword, which is a four-character string that names the parameter. For example, the primary parameter of any Apple Event, the direct object parameter, is preceded by a four-character string consisting of four dashes, ––––. This string is called the `keyDirectObject` keyword in Apple documentation. To determine the keywords for each parameter, refer to the documentation for the application from which you are sending Apple Events. Refer to the *LabVIEW Help* information about Apple Events you can send to LabVIEW.

The parameter data follows the keyword. Apple Events parameters can have many different data types, but you must specify the data types in LabVIEW in a string format. The AESend VI parses the string and compiles it into the appropriate form to send the Apple Event. You describe the data type of each parameter with a four-character type code. You can use shortcuts to describe the most common data types. For example, you can use quotes to specify `TEXT` data or enclose a comma-separated set of items in square brackets to specify a `list`. The syntax for specifying the data type is described below.

To specify an integer

Format the string as: Series of decimal digits, optionally preceded by a minus sign.

Parameter is of type code: `long` or `short`

Examples: `1234, –5678`

To specify an enumerated type

Format the string as: Four-letter code. If the code is too long, LabVIEW truncates it. If the code is too short, LabVIEW pads it with spaces. If you put single quotation marks (`'`) around the code, it can contain any characters. Otherwise, the code cannot contain `@ ' : - , ( [ { } ] )` and cannot begin with a digit.

Parameter is of type code: `enum`

Examples:
```
whos
'@all'
long
>=
'86it'
```

To specify a string

Format the string as: Enclose the appropriate sequence of characters within open and close curly quotation marks (" entered with <option-[>and " entered with <option-shift-[>).

Parameter is of type code: `TEXT`

Examples:
```
“put x into card field 5”
“Hi There”
```

To specify a record

Format the string as: Enclose a comma-separated list of elements in curly braces, where each element consists of a keyword followed by a colon followed by a value, which can be any of the types listed in this table. Use the AECreate Record VI to construct this string.

Parameter is of type code: `reco`

Examples:
```
{x:100, y:–100}
{'origin':{x:100, y:–100}, extent:{x:500, y:500}, cont:[1,5,25]}
```

To specify a list

Format the string as: Enclose a comma-separated list of descriptors in square brackets.

Parameter is of type code: `list`

Examples: `[123, –58, “test”]`

VI that can construct string: AECreate Descriptor List

To specify arbitrary data

Format the string as: Enclose an even number of hex digits between French quotation marks («entered with <option-\> and» entered with <option-shift-\>).

Parameter is of type code: There is no inherent type. You must convert the data to hex digits and you must coerce the data to a data type.

Examples: `«01 57 64 fe AB C1»`

To coerce the data to any other data type

Format the string as: Embed data created in one of the types listed in this table in parentheses and put the appropriate four-character type code before it. If the data is numeric, LabVIEW coerces the data to the specified type if possible and returns the `errAECoercionFail` error code if it cannot. If the data is a different type, LabVIEW replaces the old key word with the key word you specify.

Parameter is of type code: specified type code

Examples:
```
sing(1234)
alis(«hex dump of an alias»)
type(line)
rang{star: 5, stop: 6}
```

Null data

Format the string as: Coerce an empty string to no type.

Parameter is of type code: null

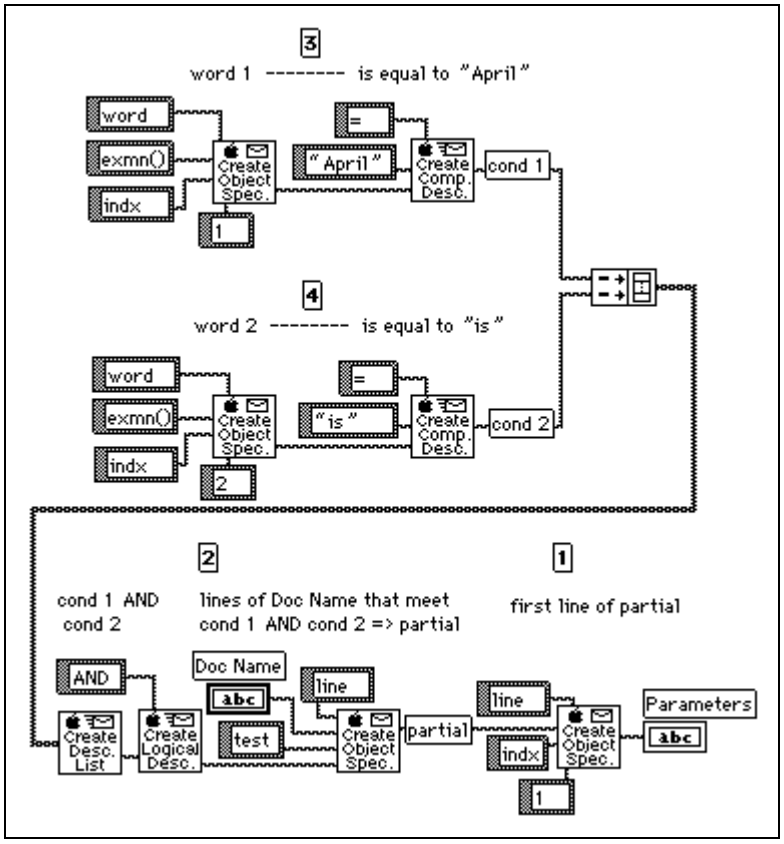
Examples: `( )`

Creating Apple Event Parameters Using Object Specifiers

The Object Support Library is a high-level interface for creating Apple Events. This interface is layered on top of the Apple Event parameter data structures described earlier in this application note. The interface helps you create common types of parameters, including range specifications. LabVIEW object support VIs are located on the **Communication»AppleEvent»Low Level Apple Events** palette.

Object Support VI Example

The following example uses the object support VIs to create an Apple Event parameter to send to a word processor. The parameter instructs the word processor to return the first line of a document you specify with the first word **`April`** and the second word **`is`**.



The following string that the previous block diagram creates is complicated. The string includes tabs to make it easier to read. Refer to the Mac OS 8.5 Registry, available in the AppleScript 1.3.4 SDK (**http://www.apple.com/developer**), for more information about the Object Support Library.

```
obj {
    want: type('line'),
    from: obj {
        want: type('line'),
        from: Doc Name,
        form: test,
        seld: logi {
            term:[
                cmpd{
                    relo:=,
                        obj1:"April",
                        obj2:obj {
                            want: type('word'),
                            from: exmn( ),
                            form: indx,
                            seld: 1
                        }
                    },
                cmpd{
                    relo:=,
                        obj1:"is",
                        obj2:obj {
                            want: type('word'),
                            from: exmn( ),
                            form: indx,
                            seld: 2
                        }
                    }
                ],
                 logc: AND
            }
        },
    form: indx,
    seld: 1
}
```

Replies to Apple Events

If LabVIEW is unable to respond to an Apple Event, the reply it sends to the application that sent the Apple Event contains an error code. If the error is not a standard Apple Event error, the reply also contains a string that describes the error. Refer to *LabVIEW Help* for more information about LabVIEW Apple Event error codes.

Event: Run VI

Description

Tells LabVIEW to run the specified VI(s). Before executing this event, the LabVIEW application must be running, and the VI must be open (you can open the VI using the Open Documents AppleEvent).

Event Class

`LBVW` Custom events use the Applications creator type for the event class.

Event ID

`GoVI` – – – –

Event Parameters

Description	Keyword	Default Type
VI or List of VIs	`keyDirectObject` (– – – –)	typeChar (`char`) (required) or list of typeChar (`list`)

Reply Parameters

none

Possible Errors

Error	Value	Description
`kLVE_InvalidState`	`1000`	VI is in a state that does not allow it to run.
`kLVE_FPNotOpen`	`1001`	VI front panel is not open.
`kLVE_CtrlErr`	`1002`	VI has controls on its front panel that are in an error state.
`kLVE_VIBad`	`1003`	VI is broken.
`kLVE_NotInMem`	`1004`	VI is not in memory.

Event: Abort VI

Description

Tells LabVIEW to abort the specified VI(s). Before executing this event, the LabVIEW application must be running, and the VI must be open. You can open the VI using the Open Documents AppleEvent. You can send this message only to VIs that are executed from the top level. SubVIs are aborted only if the calling VI is aborted.

Event Class

`LBVW` Custom events use the Applications creator type for the event class.

Event ID

`RsVI`

Event Parameters

Description	Keyword	Default Type
VI or list of VIs	`keyDirectObject` (– – – –)	typeChar (`char`) (required) or list of typeChar (`list`)

Reply Parameters

none

Possible Errors

Error	Value	Description
kLVE_InvalidState	1000	VI is in a state that does not allow it to run.
kLVE_FPNotOpen	1001	VI front panel is not open.
kLVE_NotInMem	1004	VI is not in memory.

Event: VI Active?

Description

Requests information on if a specific VI is currently running. Before executing this event, the LabVIEW application must be running, and the VI must be open. You can open the VI using the Open Documents AppleEvent. The reply indicates if the VI is currently running.

Event Class

LBVW Custom events use the Applications creator type for the event class.

Event ID

VIAc

Event Parameters

Description	Keyword	Default Type
VI Name (required)	keyDirectObject (----)	typeChar (char)

Reply Parameters

Description	Keyword	Default Type
Active? (required)	keyDirectObject (----)	typeBoolean (bool)

Possible Errors

Error	Value	Description
kAEvtErrFPNotOpen	1001	VI front panel is not open.
kLVE_NotInMem	1004	VI is not in memory.

Event: Close VI

Description

Tells LabVIEW to close the specified VI(s). Before executing this event, the LabVIEW application must be running, and the VI must be open. You can open the VI using the Open Documents AppleEvent.

Event Class

`LBVW` Custom events use the Applications creator type for the event class.

Event ID

`ClVI`

Event Parameters

Description	Keyword	Default Type
VI or list of VIs	`keyDirectObject` (––––)	typeChar (`char`) (required) or list of typeChar (`list`)
Save options (not required)	`keyAESaveOptions` (`savo`)	typeEnum (`enum`) possible values: yes and no

Reply Parameters

none

Possible Errors

Error	Value	Description
`kAEvtErrFPNotOpen`	`1001`	VI front panel is not open.
`kLVE_NotInMem`	`1004`	VI is not in memory.
`cancelError`	`43`	User cancelled the close operation.