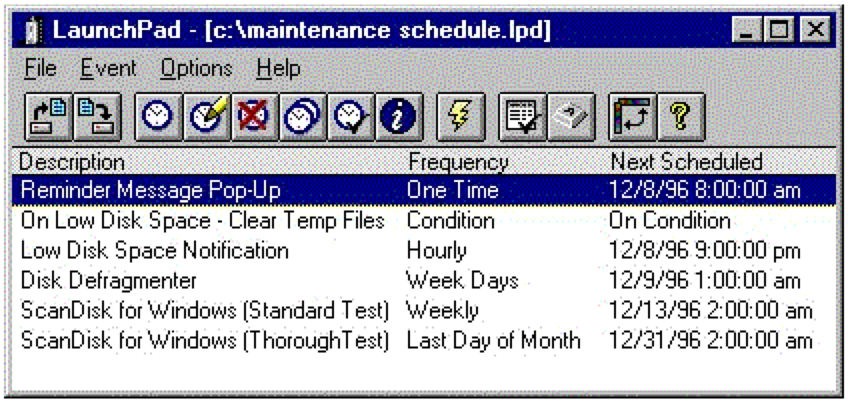
Web Site

You can find lots of great information from our Web Site. The latest news and downloads for our products, as well as answers to other customer's questions. We also have information and links to other Internet sites that you might find of interest. We will be expanding this information constantly to provide the best possible Internet support to our customers.

www.cypressnet.com

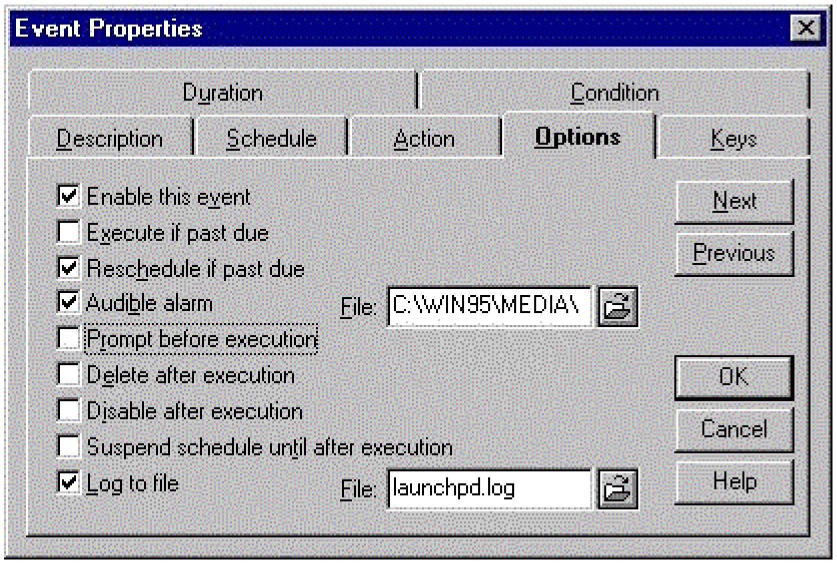
Other Cypress Technologies Products

LaunchPad, a scheduling utility for all your Windows and DOS needs.



Description	Frequency	Next Scheduled
Reminder Message Pop-Up	One Time	12/8/96 8:00:00 am
On Low Disk Space - Clear Temp Files	Condition	On Condition
Low Disk Space Notification	Hourly	12/8/96 9:00:00 pm
Disk Defragmenter	Week Days	12/9/96 1:00:00 am
ScanDisk for Windows (Standard Test)	Weekly	12/13/96 2:00:00 am
ScanDisk for Windows (ThoroughTest)	Last Day of Month	12/31/96 2:00:00 am

As you can see, LaunchPad provides a clear display of all the scheduled events, which you can sort either by ` you move around quickly. You can choose your own font for the display, and save your settings. Events are easily edited by double-clicking any event in the list.



- [✓] Enable this event
- [ ] Execute if past due
- [✓] Reschedule if past due
- [✓] Audible alarm — File: C:\WIN95\MEDIA\
- [ ] Prompt before execution
- [ ] Delete after execution
- [ ] Disable after execution
- [ ] Suspend schedule until after execution
- [✓] Log to file — File: launchpd.log

LaunchPad's tabbed display steps you through editing events from the simple once-only pop-up messages to the complex DDE scripts to a remote server scheduled on a condition trigger. A wide variety of scheduling options let you avoid pitfalls of fixed-time schedules (like those unfriendly jobs that take longer than expected and interfere with the next job). You can chose to retry the job next time around, or run it anyway, and delete or disable the job after it's done. If you've got a convoluted schedule to execute, this is the tool to handle it.

With its ability to save different schedule files, LaunchPad can be used on the same machine by multiple people with different needs, and switches between monthly shifts, etc., are easily made. Multiple log files are also supported, so tracking down problems (i.e. did that program run last night or not?) is much easier. LaunchPad can continually "sniff" for a new schedule file, so you don't have to stop it to make schedule changes.

New for LaunchPad v2.0 are new scheduling options like day of month, last day of month or work days (i.e. Mon., Wed, Fri.). In Addition, LaunchPad is now a DDE server. LaunchPad can both send and receive DDE commands to control its own events or control other applications. Send Keys now has more features including additional timing support, activation and control of applications and a stop command. LaunchPad also supports forced reboots for restarting a computer without being stopped by dialog boxes or even users. Support is now available for the Internet - simply enter your URL's at the command line for scheduled downloads. The message dialog box has a new timer feature that will display the pop-up message for a predetermined time. As you can see LaunchPad v2.0 sets new standards for application automation.

LaunchPad's versatility is there for your needs and imagination. Let the LaunchPad Scheduler work around the clock...so you don't have to. LaunchPad sells for $29.95 plus s/h. Site-licenses are welcome.

Shareware Agreement

Product Support

Product Support for GroundControl is available through one of the following:

Web Site: www.cypressnet.com
You can find lots of great information from our Web Site. The latest news for our products as well as question and answers. We also have information and links to other Internet sites that you might find of interest. We will be expanding this information constantly to provide to best stop for Product information to serve our customers.

Email support can be sent to one of the following address:

CompuServe	Direct questions to: James Hall	76172,3452
Internet	Direct questions to: Support	support@cypressnet.com

You can also write us at the following address:

Cypress Technologies
4450 California Ave. Suite K-165
Bakersfield, CA. 93309

Application Overview

GroundControl is a tool that lets you build your own Window batch files. You have at your disposal around 50 functions to do everything from basic Window controls to the more advanced DDE and SendKeys features. You use the Wizard to add commands to the macro window instead of typing them all in, or use the toolbars to add commands at the click of a button.

GroundControl saves each macro you write as a single file, and you can run other macro files from inside any given macro file. This means you can specialize your macros for certain tasks (like subroutines) and keep your files small. Using the flow control statements, you can make real-time decisions to launch those tasks or not. GroundControl comes with a full set of file functions, and a strong suite of diagnostic functions to monitor disk space and memory, track and log errors, etc.

GroundControl is not just a blind batch file - it comes with functions for user interaction through message boxes, question boxes, and input boxes. This means you can use GroundControl as a front end for other applications you don't want users to have access to, especially if you take advantage of the SendKeys and DDE functions.

All in all, GroundControl is a powerful instrument that is easy to use, easy to maintain, and far-reaching in its capabilities.

ActivateWindow

ActivateWindow(<*window title*>**)**

ActivateWindow brings a window to the foreground and makes it the active window.

Parameters: *window title - Specifies the name of the window to make the active window.*

Returns: TRUE if successful; otherwise FALSE.

Use ActivateWindow to bring a running application window to the front so it can receive action from the macro. The window title is not just simply the application name. For example if you were editing a text file called README.TXT in NotePad the window title would be "README.TXT - Notepad". GroundControl provides a browse button that will display a list of the window titles for all the currently running applications. If the window you need is not in the list, you can either type the window title in manually, or close the selection box, execute the application you need and then click the browse button again. This time the list will include the window title you need. Select it and click the OK button.

If the window is iconized when you execute this statement, it will be restored to its normal position.

Examples:

ActivateWindow("Untitled - Notepad")
ActivateWindow("Microsoft Word - Document1")

Also See:

{button ,JI(`GC.HLP',`IDH_FindWindow')} FindWindow
{button ,JI(`GC.HLP',`IDH_WindowActive')} WindowActive
{button ,JI(`GC.HLP',`IDH_CloseWindow')} CloseWindow
{button ,JI(`GC.HLP',`IDH_RunProgram')} RunProgram

SendKeysToWindow

SendKeysToWindow (*<window title>,<keys>***)**

SendKeysToWindow brings a window to the foreground and makes it active, then sends keystrokes to it.

Parameters:	*window title - Specifies the name of the window to make the active window and send keys to.*
	keys - Specifies the keystrokes to send to the activated window.
Returns:	TRUE if successful; otherwise FALSE.

Sending keys can be can be fraught with danger if you are not sure which open application is going to be active and receive the keystrokes. By using the SendKeysToWindow command you can bring a running application window to the front so it can receive the keystrokes from the macro. The window title is not just simply the application name. For example if you were editing a text file called README.TXT in NotePad the window title would be "README.TXT - Notepad". GroundControl provides a browse button that will display a list of the window titles for all the current running windows. If the window you need is not in the list, you can either type the window title in manually, or close the selection box, execute the application you need and then click the browse button again. This time the list will include the window title you need. Select it and click the OK button.

If the window is iconized when you execute this statement, it will be restored to its normal position.

You can also specify a delay between each character in the system Options dialog.

Examples:

SendKeysToWindow("Untitled - Notepad","This is a test")

Also See:

{button ,JI(`GC.HLP',`IDH_SendKeys_Overview')} **SendKeys Overview**
{button ,JI(`GC.HLP',`IDH_Key_Codes')} **Key Codes**
{button ,JI(`GC.HLP',`IDH_ActivateWindow')} **ActivateWindow**
{button ,JI(`GC.HLP',`IDH_SendKeys')} **SendKeys**
{button ,JI(`GC.HLP',`IDH_FindWindow')} **FindWindow**
{button ,JI(`GC.HLP',`IDH_SystemOptions')} **System Options**
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} **Stopping a running script**

CloseWindow

CloseWindow(<*window title*>**)**

CloseWindow closes a window based on its title.

Parameters: *window title - Specifies the name of the window to close.*

Returns: TRUE if successful; otherwise FALSE.

CloseWindow is not the opposite of ActivateWindow. CloseWindow is the same as clicking the close box in a window. CloseWindow, in most cases, quits the application. If there are unsaved open documents in an application you try to close, those applications will most likely display dialog boxes to prompt you to save changes before closing, this will prevent the window from closing until those dialogs are answered either manually or by using SendKeys. The window title is not just simply the application name. For example if you were editing a text file called README.TXT in NotePad the window title would be "README.TXT - Notepad". GroundControl provides a browse button that will display a list of the window titles for all the current running windows. If the window you need is not in the list, you can either type the window title in manually, or close the selection box, execute the application you need and then click the browse button again. This time the list will include the window title you need. Select it and click the OK button.

Examples:

CloseWindow("Untitled - Notepad")
CloseWindow("Microsoft Word - Document1")

Also See:

{button ,JI(`GC.HLP',`IDH_ActivateWindow')} ActivateWindow
{button ,JI(`GC.HLP',`IDH_QuitProgram')} QuitProgram

MessageBoxOkCancel

MessageBoxOkCancel (*<message>***)**

MessageBoxOkCancel creates a simple message box with text you specify, plus an OK button and a Cancel button. The button clicked (or the response) is captured by the ResponseOk or ResponseCancel functions.

Parameters: *message - Specifies the text that will be displayed in the message box prompt.*

MessageBoxOkCancel enables you to display a message to the screen. The execution of the macro stops until the OK or Cancel button is clicked. The button that is clicked is captured in the ResponseOk `or` ResponseCancel functions. You can then check to see which button was clicked by using an If statement, and execute different commands based on the user's response.

Examples:

MessageBoxOkCancel("File not found. Press OK to continue or Cancel to quit macro")
If(ResponseCancel(),ExitMacro())

Also See:

{button ,JI(`GC.HLP',`IDH_ResponseOk')} ResponseOk
{button ,JI(`GC.HLP',`IDH_ResponseCancel')} ResponseCancel
{button ,JI(`GC.HLP',`IDH_MessageBox')} MessageBox
{button ,JI(`GC.HLP',`IDH_InputBox')} InputBox

ResponseOk

ResponseOk()

Parameters: *None.*

Returns: TRUE if the last message box was closed with the Ok button, FALSE if not.

ResponseOk is a function used after a MessageBox, MessageBoxOkCancel, or InputBox to detect which key the user clicked to exit the dialog. This function simply checks the response from the last 'MessageBox type' command. This allows you to change the execution of the macro based on that response.

Examples:

```
InputBox("Enter Your Name:")
If(ResponseOk(),RunProgram("Notepad.exe"))
Delay(1000)
If(FindWindow("Untitled - Notepad"),SendInputToWindow("Untitled - Notepad")
```

Also See:

{button ,JI(`GC.HLP',`IDH_If')} If
{button ,JI(`GC.HLP',`IDH_MessageBox')} MessageBox
{button ,JI(`GC.HLP',`IDH_MessageBoxOkCancel')} MessageBoxOkCancel
{button ,JI(`GC.HLP',`IDH_InputBox')} InputBox

FindWindow

FindWindow(<*window title*>**)**

FindWindow is used in an If condition to check if a window is open.

Parameters: *window title - Specifies the name of the window to look for.*

Returns: TRUE if there is a window that has a title that matches the passed window title

Use FindWindow to determine if an application window is running. FindWindow will find a window even if it is not the 'active window.'

Examples:

If(FindWindow("Untitled - Notepad"), SendKeysToWindow("Untitled - Notepad","This is a test")

Also See:

{button ,JI(`GC.HLP',`IDH_ActivateWindow')} ActivateWindow
{button ,JI(`GC.HLP',`IDH_WindowActive')} WindowActive
{button ,JI(`GC.HLP',`IDH_CloseWindow')} CloseWindow
{button ,JI(`GC.HLP',`IDH_RunProgram')} RunProgram

WindowActive

WindowActive(<*window title*>**)**

WindowActive is used in an If condition to check if a window is the current active window.

Parameters: *window title - Specifies the name of the window to check for.*

Returns: TRUE if the current active window title matches the passed window title

Use WindowActive to determine if an application window is the 'active window.' Because SendKeys always sends keystrokes to the active window, you could use this command before a SendKeys command to be sure the keys are sent to the window you want. Of course, if the keys you send are constantly changing which window is active, you could use the SendKeysToWindow function instead, and not worry about using ActivateWindow over and over again.

Examples:

If(WindowActive("Untitled - Notepad"), SendKeys("This is a test"))

Also See:

{button ,JI(`GC.HLP',`IDH_ActivateWindow')} ActivateWindow
{button ,JI(`GC.HLP',`IDH_CloseWindow')} CloseWindow
{button ,JI(`GC.HLP',`IDH_RunProgram')} RunProgram

SetDirectory

SetDirectory(*<directory>***)**

SetDirectory changes the current directory.

Parameters: *directory - Specifies the name of the directory.*

Use SetDirectory to change the current working directory to another directory on your system. The current working directory where GroundControl will look for files if you don't specify a full path. If you use the command WriteLineToFile("This is a test","MyFile.txt") GroundControl would look for 'MyFile.txt' in the current working directory. If the file is not there the WriteLineToFile command would create 'MyFile.txt' in the current working directory.

Examples:

SetDirectory("c:\temp")

SetDirectory("c:\My Documents")

Also See:

{button ,JI(`GC.HLP',`IDH_LogToFile')} LogToFile

{button ,JI(`GC.HLP',`IDH_WriteToFile')} WriteToFile

{button ,JI(`GC.HLP',`IDH_WriteLineToFile')} WriteLineToFile

LogError

LogError(*<message>***)**

LogError writes an entry in the error log file

Parameters: *message - Specifies the text to be written to the log file.*

Use LogError to write custom error messages with a time and date stamp to the default log file, called macerr.log. Your macro can handle errors and post messages to the error log based on conditions in the macro. This is useful for tracing automated jobs. The log file looks something like this:

```
01/24/97 19:50:20 User said no to my dialog box
01/24/97 19:50:25 The Calculator program wasn't running.
```

If you want to use LogError as a means to trace program execution, but you don't want to clutter the error log with general messages, you can use LogToFile instead, which does exactly the same thing, except you specify which file to write the message to. This means you can have many different logfiles, each for different purposes.

Examples:

If(FileExist("C:\WIN95\win.ini"),Beep(),LogError("File Not found"))

Also See:

{button ,JI(`GC.HLP',`IDH_WriteToFile')} **WriteToFile**
{button ,JI(`GC.HLP',`IDH_WriteLineToFile')} **WriteLineToFile**
{button ,JI(`GC.HLP',`IDH_LogToFile')} **LogToFile**

FileExist

FileExist(<*file name*>**)**

FileExist checks to see if the specified file exists.

Parameters: *file name - Specifies the name of file to check for.*

Returns: TRUE if file is found; otherwise FALSE.

Use **FileExist** this command checks to see if a file exists. The application will look in the path defined for the file name. If that file is found then the command returns TRUE. A good use for this command might be to check for a file before moving it to a backup directory.

Examples:

If(FileExist("C:\WIN95\win.ini"),Beep(),LogError("File Not found"))

Also See:

{button ,JI(`GC.HLP',`IDH_WriteToFile')} <u>WriteToFile</u>
{button ,JI(`GC.HLP',`IDH_WriteLineToFile')} <u>WriteLineToFile</u>
{button ,JI(`GC.HLP',`IDH_SetDirectory')} <u>SetDirectory</u>

SetCaption

SetCaption(<*caption*>**)**

SetCaption set the window titles for macro dialogs

Parameters: *caption - Specifies the text for the macro window titles.*

Use SetCaption to customize the dialog titles in your macro. SetCaption sets the title text for the MessageBox and InputBox dialogs. It is visible for all the dialogs after that until you change it again.

Examples:

```
SetCaption("Going Once")
If(MessageBoxOkCancel("Do you want to run the daily backup?"),GoTo(":DoBackup"),Beep())
SetCaption("Going Twice")
If(MessageBoxOkCancel("How about the disk defragmenter instead?"),GoTo(":DoBackup"),Beep())
```

Also See:

{button ,JI(`GC.HLP',`IDH_MessageBox')} MessageBox
{button ,JI(`GC.HLP',`IDH_MessageBoxOkCancel')} MessageBoxOkCancel
{button ,JI(`GC.HLP',`IDH_InputBox')} InputBox

Delay

Delay(<*milliseconds*>**)**

Delay pauses the execution of the macro for a specified number of milliseconds

Parameters: *milliseconds - Specifies the number of milliseconds to wait.*

Use Delay to pause a macro and wait for an application to load before executing the next command. A millisecond is one thousandth of a second, so to pause a macro for ten seconds you would need to specify 10,000 milliseconds in the Delay parameter.

1 second = 1,000 ms
1 minute = 60,000 ms
1 hour = 3,600,000 ms

Examples:

RunProgram("WordPad.exe")
Delay(5000)
SendKeys("This is some text")

Also See:

{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} **Stopping a running script**

QuitProgram

QuitProgram(<*window title*>)

QuitProgram closes a window based on its title.

Parameters: *window title - Specifies the name of the window to close.*

Returns: TRUE if successful; otherwise FALSE.

QuitProgram is essentially the same as CloseWindow. QuitProgram quits (or exits) the application specified. If there are unsaved open documents in an application you try to close, those applications will most likely display dialog boxes to prompt you to save changes before closing. This will prevent the window from closing until those dialogs are answered either manually or by using SendKeys. The window title is not just simply the application name. For example if you were editing a text file called README.TXT in NotePad the window title would be "README.TXT - Notepad". GroundControl provides a browse button that will display a list of the window titles for all the current running windows. If the window you need is not in the list, you can either type the window title in manually, or close the selection box, execute the application you need and then click the browse button again. This time the list will include the window title you need. Select it and click the OK button.

Examples:

QuitProgram("Untitled - Notepad")
QuitProgram("Microsoft Word - Document1")

Also See:

{button ,JI(`GC.HLP',`IDH_ActivateWindow')} ActivateWindow
{button ,JI(`GC.HLP',`IDH_CloseWindow')} CloseWindow

RebootSystem

RebootSystem ()

RebootSystem executes a system reboot.

Parameters: *None.*

The RebootSystem command will shut down and then restart Windows. the computer will go to one of the following states:

Windows 95:	Exits Windows, then restarts the computer and Win95.
Windows NT:	Exits Windows, then restarts the computer and WinNT.

Normally if any program running requires confirmation before exiting, you would be required to respond to all the "Save Changes ..." dialogs before the applications will shutdown. You should try to close all the running windows before you execute this statement.

Examples:

If(FileExist("c:\temp\error.txt",RebootSystem())

Also See:

{button ,JI(`GC.HLP',`IDH_Logout')} <u>Logout</u>
{button ,JI(`GC.HLP',`IDH_ShutdownSystem')} <u>ShutdownSystem</u>
{button ,JI(`GC.HLP',`IDH_QuitProgram')} <u>QuitProgram</u>

ShutdownSystem

ShutdownSystem ()

ShutdownSystem executes a system shutdown.

Parameters: *None.*

This command will shut down Windows. The computer will go to one of the following states:

Windows 95: The "It's now safe to turn off your computer" screen.
Windows NT: The "Shutdown or Restart Computer" dialog.

Normally if any program running requires confirmation before exiting, you would be required to respond to all the "Save Changes ..." dialogs before the applications will shutdown. You should try to close all the running windows before you execute this statement.

Examples:

MessageBoxOkCancel("Do you want to perform a system shutdown?")
If(ResponseOk(),ShutdownSystem())

Also See:

{button ,JI(`GC.HLP',`IDH_Logout')} Logout
{button ,JI(`GC.HLP',`IDH_QuitProgram')} QuitProgram
{button ,JI(`GC.HLP',`IDH_ReBootSystem')} RebootSystem

Logout

Logout ()

Logout executes a system logout.

Parameters: *None.*

This command will attempt to close all open applications and Logout of the current Windows session. The computer will go to one of the following states:

Windows 95:	Windows Login dialog.
Windows NT:	The "Press Ctrl, Alt, and Delete to login" dialog.

Normally if any program running requires confirmation before exiting, you would be required to respond to all the "Save Changes ..." dialogs before the applications will shutdown.

Examples:

Logout()

Also See:

{button ,JI(`GC.HLP',`IDH_ShutdownSystem')} **ShutdownSystem**
{button ,JI(`GC.HLP',`IDH_ReBootSystem')} **ReBootSystem**
{button ,JI(`GC.HLP',`IDH_QuitProgram')} **QuitProgram**

RunMacro

RunMacro (*<macro file name>*)

RunMacro runs another macro file and then resumes execution.

Parameters: *macro file name - Specifies the name of the macro file to run*

The RunMacro command enables you create reusable macros. Once you write a macro that performs a particular function (i.e. backup files) then you do not have to duplicate that macro again and again within every macro you write later. You can use the RunMacro command to run existing macros. It is feasible to have one macro that only runs other macros based on certain conditions. To use the RunMacro command type the path and file name of the macro you want to run or you can use the Browse button to locate the macro file.

Examples:

If(FileExist("C:\WIN95\Calc.exe"),RunMacro("C:\Program Files\GroundControl\test2.gc"),Beep())

Also See:

{button ,JI(`GC.HLP',`IDH_RunProgram')} <u>RunProgram</u>
{button ,JI(`GC.HLP',`IDH_ExitMacro')} <u>ExitMacro</u>
{button ,JI(`GC.HLP',`IDH_If')} <u>If</u>

Beep

Beep()

Beep sounds the system beep.

Parameters: *None.*

Beep plays the default system beep.

Examples:

If(FileExist("C:\WIN95\win.ini"),Beep())

Also See:

{button ,JI(`GC.HLP',`IDH_PlaySound')} **PlaySound**

PlaySound

PlaySound(<*wave file name*>**)**

PlaySound plays a wave file.

Parameters: *wave file name - Specify the sound (.wav) file to be played*

PlaySound enables you to customize the way your macro sounds. Type the name of the .WAV file or use the Browse button to locate the file you want play.

Examples:

PlaySound("C:\WIN95\MEDIA\Chord.wav")

Also See:

{button ,JI(`GC.HLP',`IDH_Beep')} **Beep**

RunProgram

RunProgram(<program>**)**

RunProgram executes a program

Parameters: *program - The command line for the program to run*

Returns: TRUE if successful; otherwise FALSE.

The **RunProgram** command is used to run a program. The parameter is any valid program. The parameter can also be a document, if so, RunProgram will launch the associated program and open the document. Associations can be defined for any file extension on Windows NT or 95 interface through the Explorer under (View)(Options)(File Types). In NT 3.51, you define associations through the File Manager under (File)(Associate).

Examples:

RunProgram("c:\winnt\notepad.exe c:\autoexec.bat")
RunProgram("http://www.cypressnet.com")
RunProgram("myfile.doc")

Also See:

{button ,JI(`GC.HLP',`IDH_RunMacro')} <u>RunMacro</u>
{button ,JI(`GC.HLP',`IDH_WaitForProgram')} <u>WaitForProgram</u>

SendKeys

SendKeys(*<keys string>***)**

SendKeys sends keystrokes to the active window.

Parameters: *keys string - Specifies the keys to send.*

Returns: TRUE if successful; otherwise FALSE.

Use this command to send a keystroke or sequence of keystrokes to the active window. The active window will interpret the keys as it would if you actually had it up in the foreground, and were typing the same keys. If you brought notepad up into the foreground, and typed "Hello!", the characters would show up in the document body. However, if you then press ALT-F, the File menu drops down. Notepad knows how to interpret the different keystrokes automatically.

SendKeys works the same way. You can concatenate your keystrokes into one long string, or send multiple smaller strings, the end result is the same. To send the above example, you would use SendKeys("Hello!%F"). The %F is a special symbol for the ALT key. There are symbols for other special keys too.

SendKeys sends to the active window. You can use ActivateWindow to bring another window to the foreground to receive keystrokes, or you can use SendKeysToWindow to specify the window to send keystrokes to.

You can also specify a delay between each character in the system Options dialog.

Examples:

RunProgram("c:\winnt\notepad.exe")
Delay(2000)
SendKeys("This is an example of keys sent to notepad~")

Also See:

{button ,JI(`GC.HLP',`IDH_SendKeys_Overview')} **SendKeys Overview**
{button ,JI(`GC.HLP',`IDH_Key_Codes')} **Key Codes**
{button ,JI(`GC.HLP',`IDH_SendKeysToWindow')} **SendKeysToWindow**
{button ,JI(`GC.HLP',`IDH_RunProgram')} **RunProgram**
{button ,JI(`GC.HLP',`IDH_ActivateWindow')} **ActivateWindow**
{button ,JI(`GC.HLP',`IDH_SystemOptions')} **System Options**
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} **Stopping a running script**

ResponseCancel

ResponseCancel()

Parameters: *None.*

ResponseCancel is a function used after a MessageBox, MessageBoxOkCancel, or InputBox to detect which key the user clicked to exit the dialog. This functions simply holds the response from the last 'MessageBox type' command. This allows you to check which button a user clicked and continue the execution of the macro base on that response.

Examples:

MessageBoxOkCancel("File not found. Press OK to continue or Cancel to quit macro")
If(ResponseCancel(),ExitMacro(),)

Also See:

{button ,JI(`GC.HLP',`IDH_If')} **If**
{button ,JI(`GC.HLP',`IDH_MessageBox')} **MessageBox**
{button ,JI(`GC.HLP',`IDH_MessageBoxOkCancel')} **MessageBoxOkCancel**
{button ,JI(`GC.HLP',`IDH_InputBox')} **InputBox**

InputBox

InputBox(*<prompt>***)**

InputBox create a dialog box to prompt user for input

Parameters: *prompt - Text to prompt user*

InputBox is used to prompt a user for input. The text that is input can then be sent as keystrokes using the SendInput command. You can use SetCaption to customize the dialog title:

Examples:

```
ActivateWindow(“Untitled - NotePad”)
SetCaption("Personal Info")
InputBox("Enter your name:")
//send your name to notepad
SendInput()
```

Also See:

{button ,JI(`GC.HLP',`IDH_SendInput')} **SendInput**
{button ,JI(`GC.HLP',`IDH_SendInputToWindow')} **SendInputToWindow**
{button ,JI(`GC.HLP',`IDH_MessageBox')} **MessageBox**
{button ,JI(`GC.HLP',`IDH_MessageBoxOkCancel')} **MessageBoxOkCancel**
{button ,JI(`GC.HLP',`IDH_SetCaption')} **SetCaption**

SendInputToWindow

SendInputToWindow(<*window title*>**)**

SendInputToWindow send the text input from an InputBox command as keystrokes

Parameters: *window title - title of window to send text to*

SendInputToWindow is used to send the text that was input from the user in the InputBox command. The text is sent as keystrokes to the window specified.

Examples:

SendKeysToWindow("Untitled - Notepad",”This is a test message from ”)
InputBox("Enter your name:")
SendInputToWindow("Untitled - Notepad")

Also See:

{button ,JI(`GC.HLP',`IDH_InputBox')} <u>InputBox</u>
{button ,JI(`GC.HLP',`IDH_SendInput')} <u>SendInput</u>
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} <u>Stopping a running script</u>

If

If(*<condition>,<command if TRUE>[,<command if FALSE>]***)**

If is a condition tester that can be used to control program flow

Parameters: *condition - Specifies the conditional statement that is evaluated*
command if TRUE - The command that is executed if the conditional statement returns TRUE
command if FALSE - The command that is executed if the conditional statement returns FALSE

The **If** command is used to control program flow. The <condition> parameter can be any command that returns TRUE or FALSE. Almost all GroundControl commands can be used. The <command if TRUE> parameter is the command that will be executed if the condition returns TRUE. It can be any valid command. You may also nest If commands. The <command if FALSE> parameter is the command that will be executed if the condition returns FALSE. This parameter is optional. If this parameter is omitted and the condition returns FALSE, program execution will continue on the next line.

Examples:

```
// this if statement uses three parameters
MessageBoxOkCancel("Do you want to continue?")
If(ResponseOk(),GoTo(":process"),ExitMacro())

:process
LogToFile("c:\test.log","Processing....")

// this if statement uses only the first two parameter
If( FileExist("c:\autoexec.bat"),MessageBox("Old system settings found"))
```

Also See:
{button ,JI(`GC.HLP',`IDH_GoTo')} **GoTo**

LogToFile

LogToFile(*<file name>,<string to log>***)**

LogToFile Writes a string to a specified file with a time-date stamp. This is useful for tracking errors and events, and making sure that certain routines are being executed, etc. It works the same as LogError, but you specify your own log file instead of the system default “macerr.log”.

Parameters: *file name - name of the file to write to.*
String - the string you want to write.

Examples:

```
LogToFile("f:\gc\mylog.txt","This is before the delay")
Delay(10000)
LogToFile("f:\gc\mylog.txt","This is after the delay")
```

The results look like this (note the times):

```
01/24/97 20:32:12 This is before the delay
01/24/97 20:32:22 This is after the delay
```

WriteToFile

WriteToFile(*<file name>,<text>***)**

WriteToFile writes a string to a file

Parameters: *file name - The file name to write*
text - The text to write to the file

Returns: TRUE if successful; otherwise FALSE.

The **WriteToFile** command writes text to a file. If the file name specified does not exist, it will be created. The text is appended to the file without a carriage return. To write a line to a file with a carriage return use the command WriteLineToFile.

Examples:

The following script:

WriteToFile("f:\gc\x.txt","Hello my name is ")
WriteToFile("f:\gc\x.txt","John. ")
WriteToFile("f:\gc\x.txt","How are you?")

Produces the following in file x.txt:
Hello my name is John. How are you?
Compare this with the example given for WriteLineToFile

Also See:
{button ,JI(`GC.HLP',`IDH_WriteLineToFile')} WriteLineToFile
{button ,JI(`GC.HLP',`IDH_LogError')} LogError
{button ,JI(`GC.HLP',`IDH_LogToFile')} LogToFile

WriteLineToFile

WriteLineToFile(*<file name>,<text>***)**

WriteLineToFile writes a string to a file with a carriage return appended.

Parameters *file name - The file name to write*
text - The text to write to the file

Returns: TRUE if successful; otherwise FALSE.

The **WriteLineToFile** command writes text to a file. If the file name specified does not exist, it will be created. The text is appended to the file with a carriage return. To write a line to a file without a carriage return use the command WriteToFile.

Examples:

The following script:

WriteLineToFile("f:\gc\x.txt","Hello my name is ")
WriteLineToFile("f:\gc\x.txt","John. ")
WriteLineToFile("f:\gc\x.txt","How are you?")

Produces the following file:

```
Hello my name is
John.
How are you?
```

Compare this with the example given for WriteToFile.

Also See:
{button ,JI(`GC.HLP',`IDH_WriteToFile')} <u>WriteToFile</u>
{button ,JI(`GC.HLP',`IDH_LogError')} <u>LogError</u>
{button ,JI(`GC.HLP',`IDH_LogToFile')} <u>LogToFile</u>

ExitMacro

ExitMacro()

Parameters: *None.*

ExitMacro is used to end the execution of a macro immediately.

Examples:

```
OnError(":error")
// your macro commands
LogToFile("mylog.txt","Macro completed successfully")
ExitMacro

:error
LogError("Macro failed")
```

Tease Me Now!

Let's write a very quick, easy macro that beeps and shows a MessageBox that says “Hello, world!”. (This is the first program that almost everybody writes, so don’t scoff if you think it’s too easy!). Start with an empty file, and use the View menu to hide all the toolbars except the "Common" toolbar. Now lets get started:

Click on the little horn icon. This adds a line that looks like Beep() to the macro. Click it again if you want, and you'll beep twice. You won't hear them until you run the macro, though.

Now move the mouse along the toolbar until you see the "MessageBox" tip. It's the second tool from the end. When you click this, you get a dialog box that has MessageBox on the “Commands” line, but nothing in the “Message Text” line. Type "Hello, world!" into the second box and hit OK. Your macro should now look like this:

Beep()
MessageBox("Hello, world!")

Now look to the right where to the "Run" button is and press it. Your macro should beep, and show a message box saying “Hello, world!”. You can save this macro by selecting "File" and "Save As" from the main menu just like you would any other document.

To get a taste of what functions are available, check out the Function by Category section. There are about fifty statements you can execute, and with loops and branches, you can organize them in an infinite number of ways! You can get write message boxes that ask for information, and manipulate other programs via SendKeys and DDE.

Also See:

{button ,JI(`GC.HLP',`IDH_How_Do_I')} **How Do I ...**
{button ,JI(`GC.HLP',`IDH_Function_Descriptions_by_Category')} **Function Descriptions**
{button ,JI(`GC.HLP',`IDH_Loop_and_Branches')} **Loops and Branches**
{button ,JI(`GC.HLP',`IDH_Tip_SendKeys')} **Using SendKeys**
{button ,JI(`GC.HLP',`IDH_How_Do_I_Use_DDE_Effectively')} **Using DDE**
{button ,JI(`GC.HLP',`IDH_Asking_for_it')} **Interactive Dialogs**

Function Descriptions by Category

There are around fifty functions in the basic GroundControl package. Many of them are self-descriptive and obviously can be used alone, for example: Beep(). Others are more complicated and need to be used in the context of other functions, like OnError() or While().

The following sound functions can be used to add audible alarms or make your macro more interactive:

Beep	Beep a single tone
PlaySound	Play a .wav file

The next group allows you to manipulate already open windows. As long as you know the title bar of the window you want to control, you can use any of these functions:

ActivateWindow	Make the specified window active (bring to top)
CloseWindow	Close the specified window
MinimizeWindow	Minimize the active window
MaximizeWindow	Maximize the active window
NormalizeWindow	Put active window back in normal position (de-maximize or de-minimize)
FindWindow	Returns true if specified window is open (not necessarily active)
WindowActive	Returns true if specified window is active (on top)

You can start and stop programs with the following tools:

RunMacro	Starts another macro file, then comes back to the original file
ExitMacro	Stops the currently executing macro
RunProgram	Starts an application
QuitProgram	Sends an exit command to the specified window
RunDOSCommand	Executes a DOS command
WaitForProgram	Waits for the program started with RunProgram to exit

You can shutdown the system in a variety of ways:

Logout	Log the current user out
PowerOff	Power the system down (for laptops)
RebootSystem	Shutdown with restart
ShutdownSystem	Shutdown without restart

There are functions for changing flow control:

If	Checks a return value, and executes one of two options.
Repeat	Repeats a function for a specified number of times
Delay	Pauses for a specified time
While	Loops while a condition is true
OnError	Defines where to jump to if an error occurs

You can jump around in a macro using GoTo and labels, and use comments to skip over code.

GoTo	Jump to a specified label, e.g.: GoTo(":Jail")
:label	Defines a label, e.g.: ":Jail"
Comment	Defines a line of code to be ignored, e.g., “//PowerOff()”

There are functions to monitor system resources and provide diagnostics:

FileExist	Checks if the specified file exists
FileOlderThan	Checks the date of a file against a specified date.

IfLessThan	Compares a system resource against a numerical value and takes action
IfMoreThan	Compares a system resource against a numerical value and takes action
LogError	Writes a time-stamped message to the GroundControl error log file
LogEvent	Writes a time-stamped message to the GroundControl system log file
LogSystem	Writes a time-stamped system resource value into a specified logfile
SetDirectory	Changes the current working directory

File output is provided by:

WriteLineToFile	Write a single line of text to a file (with carriage return)
WriteToFile	Write a piece of text to a file (without carriage return)
LogToFile	Write a message to a file with time and date stamp (with carriage return)

User interaction is provided by:

SetCaption	Change the caption of all the dialogs from this point forward
MessageBox	Show a message box (with an OK button only)
MessageBoxOkCancel	Show an OK/Cancel message box, and return which was pressed
ResponseOk	Checks if the last MessageBox was closed with the Ok button
ResponseCancel	Checks if the last MessageBox was closed with the Cancel button

A powerful extension of GroundControl is the SendKeys function set, which sends a piece of text to a window as individual keystrokes. You can send individual keys, ask for a line of text from the user and send that, or send one of the environment or system variables. If you are sending lots of keystrokes, you would want to ActivateWindow() first. SendKeys can be tricky if you "misplace" the active window, but it allows you to control a program remotely as though you were actually there clicking menus and typing into dialogs.

SendKeys	Send a line of text to the active window
SendKeysToWindow	Send a line of text to the specified window
InputBox	Ask for a line of text, and store in the input buffer
SendInput	Send the input buffer to the active window
SendInputToWindow	Send the input buffer to the specified window
SendEnvironment	Send an environment variable to the active window
SendNow	Send the current time to the active window
SendKeysFromFile	Send the contents of a file as keystrokes.

The other extension to GroundControl is the DDE function set, which allows you to send messages to other programs, including those on other workstations if you are part of a network. DDE can be complicated and it requires you know the proper syntax understood by the program you are trying to communicate with. If you are running on a network, you may also have to preconfigure the DDE shares on the host system. On the other hand, a well written DDE server will make program control much cleaner and robust than SendKeys because DDE is not dependent on activating windows and making sure that the text you send goes to the right place.

DdeCommand	Send a single DDE command to an application and topic
DdeScript	Send multiple DDE commands from a file to an application and topic

Loop and Branches

Your first GroundControl program will probably be a simple series of commands that are executed from top to bottom, but as you probably can imagine, a really useful program is one which makes decisions, either independently of the user, or based on user input. This section explains how to use branches and loops effectively.

As an example, let's write a program that beeps ten times in a row. The first pass might be (and don't laugh):

```
Beep()
Beep()
Beep()
Beep()
Beep()
Beep()
Beep()
Beep()
Beep()
Beep()
```

This takes up lots of space and looks quite silly. The better way is to use a Repeat():

```
Repeat(Beep(),10)
```

The stuff in the middle is executed 10 times. The value of counter starts at one, and   increases by one each time until it reaches ten, where the loop exits. In our example, suppose we wanted to beep ten times (decabeep, or decibeep?), but play a sound clip and wait five seconds after each. GroundControl's repeat loop has a shortcoming: you can only repeat one statement. Thus, without knowing better, we could write this:

```
Beep()
PlaySound("WHITEZON.WAV")
Delay(5000)

Beep()
PlaySound("WHITEZON.WAV")
Delay(5000)
```

... and so on eight more times. What we need is a repeat that takes more than one function as a parameter. "But wait, ", you cry, "this is impossible!. Did not Aristotle decree it was so?". "Ah, grasshopper, ", we respond smugly, "watch as we shine light where there was only darkness".

If we put the first three statements into a macro file and save them as "NOTIFY.gc", we can call them like this from our main macro:

```
Repeat(RunMacro("NOTIFY.gc"),10)
```

We have turned the three statements into a subroutine that can be called anywhere by anyone. We have used the repeat function to run the other macro file ten times. You should now realize how powerful the technique of subroutines and loops are, and how they can be used together for almost infinite flexibility. If you need a For-Next loop, use this concept.

The next section shows some other useful techniques for looping that accommodate situations that the Repeat loop can't.

{button ,JI(`GC.HLP',`IDH_Improved_Branching')}   **Improved Branching**

Improved Branching

Suppose we have a situation where we want to beep to notify us that a file exists. Pretend that FLAG.DAT is a file that shows up on the network Q drive intermittently, and we want to hear an audible alarm when the file is there. The Repeat won't work because we don't know how many times to repeat the beep. We can use a While loop instead:

```
While(FileExist("Q:\FLAG.DAT"),Beep())
```

This might look like the solution, but in actuality, there are two problems. The above code will produce a siren, because the While loop executes as fast as it can as long as the condition is true. We will have a steady stream of beeps until the file goes away. It would be better to check the file only once a minute. By the way, if you test the above line, you'll see what we mean by siren. Be prepared to press CTRL-ALT-DELETE and end the GroundControl task to make it go away!.

The other problem is that if the file doesn't exist when we run this program, the loop exits immediately, and the macro continues with the statements following the While line. We need a condition to stay true all the time. A good way of doing this is to look for a program or window that is always there, and the best candidate is GroundControl! Seriously, you can't run macros without it, can you? Let's use the subroutine concept, and write two files: This is main.gc:

```
While(FindWindow("main.gc - GrndCntl",RunMacro("beep.gc"))
```

And this is beep.gc:

```
If(FileExist("Q:\FLAG.DAT"),Beep())
Delay(1000)
```

You can test these by running main.gc, and using a DOS box to create and delete the signal file.

So far you have looked at loops. Here we introduce the If and GoTo branch combination to you. These are actually the easiest pair to use and understand, but they do have a drawback: If you don't plan your labels and jumps properly, your macro will hop all over the place and you will be hard pressed to follow it around. GoTo statement have earned a nasty reputation as being responsible for the proliferation of "spaghetti code". Actually, tracing GoTo statements is what led to the rise of C++ Programming, the spread of cholera, and the Second Punic War. Anyway, here's how to write the beep code for our example:

```
:Start
// wait a minute then check the file ad infinitum
Delay(60000)
If(FileExist("Q:\FLAG.DAT"),Beep())
GoTo(":Start")
```

This doesn't look too bad. Suppose instead of beeping, we wanted to beep, play a sound file recording of "The flag file is here again, m'sieur", and log a message recording the fact for posterity to C:\FLAG.LOG. If the flag file isn't there, we just want to play a different sound clip and log a different message. Furthermore, if a file called Q:\KEEPGOING.DAT showed up on the network, we want to quit hanging around waiting for the first file, and get on with the rest of the macro. We can use If and GoTo like this:

```
:Start
If(FileExist("Q:\KEEPGOING.DAT"),GoTo(":Done"))
Delay(60000)
If(FileExist("Q:\FLAG.DAT"),GoTo(":FileHere"),GoTo(":NoFile"))
GoTo(":Start")

:FileHere
Beep()
PlaySound("C:\FILEHERE.WAV")
LogMessage("C:\FLAG.LOG","File showed up")
GoTo(":Start")

:NoFile
PlaySound("C:\NOFILE.WAV")
LogMessage("C:\FLAG.LOG","File isn't there")
```

```
GoTo(":Start")

:Done
// more functions after this
...
```

You'll notice that the line GoTo(":Start") occurs three times. The :Start label acts as the top of the loop. The :FileHere and :NoFile labels mark the start of the block of functions to execute depending on which was chosen, and each returns to the top of the loop (:Start). Which block to jump to is decided by the second If statement. The first If statement is what decides when to quit the loop. You will want to avoid programming infinite loops that never exit! However, if you do, you won't be the first, and the next section explains some possible scenarios where you might accidentally write some, and what you should do if you deliberately want to write one.

{button ,JI(`GC.HLP',`IDH_Infinite_Looping')} **Infinite Looping**

Infinite Looping

In this section, we point out some pitfalls of coding that you may encounter. What do you think the following examples do?

1. Simple loop:

```
:Start
// do stuff
GoTo(":Start")
```

2. Another simple loop:

```
:Start
// do stuff
Repeat(GoTo(":Start"),1)
```

3. Two-point loop:

```
:PointA
// do stuff
GoTo(":PointB")
:PointB
// do more stuff
GoTo(":PointA")
```

These are common examples of infinite loops. GroundControl will trap these and break out after 5000 executions. You can use these constructs in your code, but if your macros seem to quit working after a certain time for no reason, you may have accidentally written one of the above loop methods. Now, how about this one:

4. In macro file "EXAMPLE.gc":

```
Beep()
RunMacro("EXAMPLE.gc")
```

By running our macro file over and over, we are bypassing the loop limit and have created a true infinite loop. The end result of this is a stack fault, because we have called our own macro without ever truly exiting it first. This kind of problem is called a re-entrancy error. Make sure you don't call yourself! Here's another example using two macro files:

5. In macro file "EXAMPLE1.gc":

```
LogToFile("f:\gc\x.txt","hello from example1.gc")
RunMacro("EXAMPLE2.gc")
```

but in macro file "EXAMPLE2.gc" the following exists:

```
LogToFile("f:\gc\x.txt","hello from example2.gc")
RunMacro("EXAMPLE1.gc")
```

This code generated a log file 50 pages long before it finally ran out of stack space! What's sad about this is that the code looks OK, and even runs for a while, but it will crash eventually. You should easily be able to imagine a scenario where you have accidentally written a loop consisting of three or four files, but none of the files looks wrong. Your best bet here is to print out all your macros, close the door, unplug the phone, and start following the GoTo and RunMacro statements. If you end up where you started, you have found an infinite loop!

As a last example, look at the following little clip. You might think that it looks OK, but there is a problem:

6. In macro file main.gc:

```
While(FindWindow("main.gc - GrndCntl"),RunMacro("f:\gc\beep.gc"))
```

In macro file beep.gc

```
If(FileExist("Q:\FLAG.DAT"),Beep())
//Delay(1000)
```

This is just like the earlier example, except with the Delay statement commented out. Everything looks OK, we have a While loop that executes as long as GroundControl is running, it avoids re-entrancy and stack faults by using a subroutine, and it won't bail after 5000 iterations. The problem is that since there is no delay, the loop is tight, and there is no chance for anything else to happen. In computer jargon, the GroundControl thread or process never has a chance to respond to any other messages. You can make the delay as small as 1 ms if you want and the program will run. If you make it zero, or take the delay statement out, you will have to use the Task Manager to kill the process.

So to summarize, if you want an infinite loop, you should meet the following conditions:

1. Use a While statement to look for the GroundControl window (which will always be there)
2. Call RunMacro to execute the body of the While loop.
3. Add some small delay inside the loop to keep the thread from locking up.

Good luck!

Interactive Dialogs

What's more polite than a program that asks you nicely for stuff, and listens to what you said? OK, besides a Knight of the Bath after a good dubbing? Nothing, right? Right. How do you write one of those?

The best way is to use the MessageBox, MessageBoxOkCancel, and InputBox functions. The first simply throws up a box with your favorite message on it, like "File didn't exist. Try again!". The user hits OK and the macro continues. The second box shows a message the same way, but will return which button the user pushed to clear it, like "File didn't exist, run program anyway?". You can use them together as shown below:

```
MessageBox("Going to look for account file")
If( FileExist("ACCOUNT.TXT"),GoTo(":Found"))
If( MessageBox("Couldn't find the account file. Give up?"),GoTo(":OtherStuff"))
RunProgram("NOTEPAD")
GoTo(":OtherStuff")
:Found
RunProgram("NOTEPAD ACCOUNT.TXT")

:OtherStuff
// keep going
```

As a different example, lets ask the user for a file name, and open that in notepad. The comments explain what happens as we go:

```
InputBox("Gimme the name of the account file")
// At this point the name is stored in the internal buffer used by GroundControl.
// there is enough room in the buffer for one answer at a time. We start notepad:
RunProgram("NOTEPAD")
// then send it the ALT-F combination to manipulate the menus for File-Open.
SendKeys("%F")
SendKeys("O");
// we now have the File Open dialog up with focus in the file name box.
// this is where we send it the contents of the buffer
SendInput();
SendKeys("^M")
// ^M is like hitting Enter, or OK on the dialog. We should now have the file open
```

The InputBox function asks for one piece of text, and saves it in the input buffer until you send it someplace via SendKeys. This is good spot to examine the next feature, the SendKeys function in a little more detail.

{button ,JI(`GC.HLP',`IDH_Tip_SendKeys')} SendKeys

SendKeys

SendKeys allows you to control another program as though you were actually running it. In Windows technical terms, the information you send via the SendKeys functions is placed in the Keyboard Input Queue for that program one keystroke at a time. If you wanted to send an address, for example "101, Pine St." this would actually be broken down and sent as:

"1"	
"0"	
"1"	
","	comma character
" "	
"P"	
"i"	
"n"	
"e"	
" "	space character
"S"	
"t"	
"."	period character

The capital P and S are a combination of the Shift and p and s keys respectively and are combined inside Windows, but you don't have to worry about them. Any string you send is dis-assembled into individual keystrokes. What's cool is that you just type anything as normal into the string you want to send, including upper-lower case and punctuation ("My @#!53$% computer!!!") and it will be sent.

This functionality is enhanced by being able to send special keys, like CTRL-M, ALT-T, and the function keys F1 - F12, and even some funky system keys (Home, Num Lock). Now, why do you want to send ALT-F anyway? The answer is that these let you blast your way through menus and dialog boxes like there's no tomorrow. For example, fire up NotePad, and do the following:

```
ALT-F          // activate file menu
N              // choose new
H              // letter H
e              // letter e
l              // letter l
l              // letter l
o              // letter o
!              // exclamation
ENTER          // new line
ALT-F          // activate file menu again
a              // letter a
.              // period
t              // letter t
x              // letter x
t              // letter t
ENTER          // hit OK button to save file
```

You have created a text file called "a.txt" with the contents "Hello!". You sent the ALT-F combination twice to access the File menu, and some of the other keystrokes went either to the notepad document itself or to an entry in the dialog box.

This should illustrate the point of "focus". Where the keystrokes go is a matter of which window is selected, then where in the window, which menu, which field in the dialog box, etc. If you send keys to the wrong control the results will be unpredictable. In most cases, the system will probably just beep at you in error, but you could do more serious damage if the circumstances were right.

GroundControl has a shortcut for the special keys like ALT- and CTRL-. ALT- is symbolized by a % and CTRL- by a caret (^), as in:

ALT-F	is %F

CTRL-M	is ^M, can be used for ENTER

To send "Hello" followed by ENTER, use:

SendKeys("Hello")
SendKeys("^M")

To send ALT-F followed by X for an Exit command, use:

SendKeys("%F")
SendKeys("X")

You could send all the keystrokes in sequence like:

SendKeys("Hello^M") and SendKeys("%FX").

The other SendKeys functions allow you to send other data. They just format it as text and break it down into keystrokes before it gets sent. Use SendEnvironment to send an environment variable. If you're not sure what an environment variable is, it's one of those things the network manager told you that you had to have "set properly" in the "config.sys" file for your computer to work. You can see all your environment variables by opening a Command Window and typing "SET":

SendEnvironment("WINDOW") sends "C:\WINDOWS"
SendEnvironment("TEMP") sends "C:\TEMP"

You can use SendNow to send the current time, or parts of it:

SendNow("%c")

If you are running several windows at once, the following allow you to specify which window to send the keys or input buffer to:

SendKeysToWindow	*send a line of text to the specified window*
SendInputToWindow	*send the input buffer to the specified window*

The last SendKeys command is SendKeysFromFile , which lets you put all your keystrokes into a file, and send the file. It also allows you to change windows mid-stream, so not everything has to go to the same window.

If you want to see a more elaborate example of how you can use GroundControl to communicate with another program using SendKeys, follow on:

{button ,JI(`GC.HLP',`IDH_Key_Codes')} **Key Codes**

{button ,JI(`GC.HLP',`IDH_Ground_Control_does_math_too')} **Advanced SendKeys**

Start another program?

This is accomplished with the RunProgram function. Use CloseWindow or QuitProgram to end when you're done.

Example:

```
RunProgram("Notepad.exe")
```

Check for disk space and stuff like that?

You can use the IfMoreThan or IfLessThan functions to check any of GroundControl's predefined variables, and make a decision based on the TRUE or FALSE result. These variables are defined for different aspects of memory and disk space. For example, to see if your PC has at least 32MB of physical memory, do the following:

IfMoreThan(MEMORY_TOTALPHYSICAL,32000000,MessageBox("My PC rocks!")

Windows usually creates a swapfile to make room in physical memory as needed. There are variables to check the swapfile:

IfLessThan(MEMORY_AVAILPAGEFILE,5000000,MessageBox("Less than 5 MB left in the swapfile!")

The sum of physical memory and the swapfile is called virtual memory, and you can check that too:

IfLessThan(MEMORY_TOTALVIRTUAL,8000000,MessageBox("Better buy some chips!'')
IfMoreThan(MEMORY_PERCENTUSED,90,LogError("Almost no room for growth - close some apps!")

You can check the size and space of any hard drive too:

IfMoreThan(DISK_Q_FREE,10000000,LogError("10MB of space on network disk Q:!"))
IfMoreThan(DISK_C_SIZE,2000000000,LogError("You've got a two gigabytes hard drive!"))

Finally, there are a few string constants, like your computer name, the user name, etc. These can't be used by IfMoreThan or IfLessThan, but you can log them to a file with LogSystem. LogSystem can log any of the disk or memory variables as well. See LogSystem and IfMoreThan, IfLessThan for details.

Move windows around?

You can minimize, maximize, restore, and close Windows using the simple commands below. As long as you know the exact title of the window, you can use these commands, plus a bunch of others:

```
MinimizeWindow ("Calculator")
MaximizeWindow ("File Manager")
```

Write stuff to a file?

You can use LogToFile , LogSystem , and LogError to create time-stamped system logs. These look like this:

```
01/24/97 19:50:20 User said no to my dialog box
01/24/97 19:50:25 The Calculator program wasn't running.
```

Alternately, you can use WriteToFile and WriteLineToFile to create non-timestamped text files with and without carriage returns appended for you automatically.

Interactively fill out a dialog box?

If you know the fields of the dialog box in advance, you can use the SendKeys functions to transmit keystrokes to the box controls. Generally, you will send ALT- combinations to set focus to a control, then send the keys you want to enter in. You can send date, time, and environment variables via different SendKeys functions, but the really cool function is by using the InputBox to ask for something from the user, and then send what they typed in using SendInput.

For more information, check out the tutorial on SendKeys.

See Also:

{button ,JI(`GC.HLP',`IDH_SendKeys')} **SendKeys Function**
{button ,JI(`GC.HLP',`IDH_Tip_SendKeys')} **SendKeys Tutorial**
{button ,JI(`GC.HLP',`IDH_SendKeys_Overview')} **SendKeys Overview**

Make a choice based on the result of a function?

The If statement lets you specify what to do if a function succeeds or fails. A very common example is to check for a file or program, and use GoTo to jump to a different part of the program:

```
If(FileExists("myfile.txt"),GoTo(":Yes"),GoTo(":No"))
```

You can check resources and branch using the IfMoreThan and IfLessThan functions. You don't necessarily have to branch someplace:

```
IfMoreThan(MEMORY_TOTALPHYSICAL,64000000,MessageBox("This PC has way more than enough memory!"))
IfLessThan(DISK_C_FREE,10000000,MessageBox("This PC has less than than 10MB of disk space left!"))
```

Ask the user for something, and make a decision on it?

You can use the If statement to check the results of OK/Cancel message boxes:

```
If(MessageBoxOkCancel("Should we continue?"),GoTo(":RunProg"),GoTo(":GiveUpGhost"))
```

You can use the ResponseOk or ResponseCancel functions too:

```
MessageBoxOkCancel("Should we continue?")
If(ResponseOk(),GoTo(":RunProg"),GoTo(":GiveUpGhost"))
```

Write a For-Next loop?

You can use a simple Repeat function to execute one statement a specified number of times:

```
// annoying noise
Repeat(Beep(),50)
```

If you want to repeat a group of statements, you are better off putting those statements in a separate file, and using the RunMacro with the repeat:

```
//for-next loop
Repeat(RunMacro("chores.gc"),50)
```

If you have a need for this, you should review the tutorial on loops and branches.

Also See:

{button ,JI(`GC.HLP',`IDH_Loop_and_Branches')} **Loops and Branches**

Write a Do-While loop?

GroundControl has a While statement that accepts a function to evaluate every pass, and a function to execute if true. If false, the loop exits:

```
// annoying noise
Repeat(Beep(),50)
```

If you want to repeat a group of statements, you are better off putting those statements in a separate file, and using the RunMacro with the repeat:

```
//do-while loop
While(FileExists("signal.txt"),RunMacro("chores.gc"),50)
```

If you have a need for this, you should review the tutorial on loops and branches.

Also See:

{button ,JI(`GC.HLP',`IDH_Loop_and_Branches')}   Loops and Branches

IfMoreThan

IfMoreThan(*<A>,<B>,< Action if A more than B >***)**

IfMoreThan compares A against B and performs the specified action if A is more than B.

Parameters:	*A,B - Numbers or numerical constants*
	Action - The action if A is more than B

This function is useful for monitoring system resources like disk space, memory, etc. You will generally enter one of the numerical constants listed below for parameter A and a number for B. Constants exist for memory and disk space monitoring. The last two examples show how you can change the order of parameters and reverse the condition:

MEMORY_TOTALPHYSICAL	bytes of physical memory installed
MEMORY_AVAILPHYSICAL	bytes of physical memory available
MEMORY_TOTALPAGEFILE	bytes allocated for the pagefile
MEMORY_AVAILPAGEFILE	bytes left in the pagefile
MEMORY_TOTALVIRTUAL	bytes installed in physical memory + allocated for pagefile
MEMORY_AVAILVIRTUAL	bytes left in physical + pagefile
MEMORY_PERCENTUSED	actual % (0-100)
DISK_x_SIZE	total size of specified disk in bytes (x can be A through Z)
DISK_x_FREE	actual bytes left on specified disk (x can be A through Z)

Examples:

```
IfMoreThan(MEMORY_PERCENTUSED,50,GoTo(":LotsOfRoom"))
MessageBox("May run out of memory!")
GoTo(":Done")
:LotsOfRoom
MessageBox("Keep loading programs!")
:Done
```

You can put the constants in either parameter:

```
IfMoreThan(10000000,DISK_Q_FREE,MessageBox("AlmostOut"))
IfMoreThan(DISK_Q_FREE,10000000,MessageBox("PlentyOfSpace"))
```

See Also:

{button ,JI(`GC.HLP',`IDH_IfLessThan')} **IfLessThan**

IfLessThan

IfLessThan(*<A>,<B>,<Action if A less than B>***)**

IfLessThan compares A against B and performs the specified action if A is less than B.

Parameters: *A,B - Numbers or numerical constants*
Action - The action if A is less than B

This function is useful for monitoring system resources like disk space, memory, etc. You will generally enter one of the numerical constants listed below for parameter A and a number for B. Constants exist for memory and disk space monitoring (See IfMoreThan for the full list). The last two examples show how you can change the order of parameters and reverse the condition:

Examples:

```
IfLessThan(MEMORY_PERCENTUSED,50,GoTo(":RunningLow"))
MessageBox("Keep loading programs!")
GoTo(":Done")
:RunningLow
MessageBox("May run out of memory!")
:Done
```

You can put the constants in either parameter:

```
IfLessThan(10000000,DISK_Q_FREE,MessageBox("PlentyOfSpace"))
IfLessThan(DISK_Q_FREE,10000000,MessageBox("AlmostOut"))
```

Also See:

{button ,JI(`GC.HLP',`IDH_IfMoreThan')} IfMoreThan

DdeCommand

DdeCommand(*<application>, <topic>, <command>***)**

Parameters: *application - DDE server application*
topic - DDE server topic
command - A single DDE command

DdeCommand will send a single DDE command to the specified application and topic. The application must support DDE commands, and the command must be in the syntax supported by the server. This will vary from server to server.

GroundControl will open a conversation to the application and topic, send the command, and close the conversation. This tends to create overhead if you have a series of commands to send, so if you have two or more, you might consider using the DdeScript function instead.

Examples:

DdeCommand("launchpad","commands","FileClose()")
DdeCommand("launchpad","commands","Exit()")

See Also:
{button ,JI(`GC.HLP',`IDH_DdeScript')} **DdeScript**
{button ,JI(`GC.HLP',`IDH_DDE_Overview')} **DDE Overview**

DdeScript

DdeScript(*<application>, <topic>, <command script file>***)**

Parameters: *application - DDE server application*
topic - DDE server topic
command - script file filename with DDE commands

This function allows you to open a single conversation, and send a large number of DDE execute commands before closing the link. The commands are stored in a single text file, and look exactly like the single commands in DdeCommand:

Example:

Suppose we have a hypothetical server and topic:

DdeScript("launchpad","commands","longlist.txt")

If longlist.txt has the following contents, the commands would be executed one after the other:

FileClose()
Exit()

Also See:

{button ,JI(`GC.HLP',`IDH_DdeCommand')} DdeCommand
{button ,JI(`GC.HLP',`IDH_DDE_Overview')} DDE Overview
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} Stopping a running script

DDE Overview

Dynamic Data Exchange (DDE) is a protocol that permits two applications to communicate by exchanging data through a DDE "connection."   Through this connection you can send commands that open, close, and save files, etc. Applications that can receive DDE commands will often list the available DDE commands in the help file. Not all programs support DDE, however.

To establish a connection, you need a server application and a topic name. The server is usually the name of the program you are connecting to, but the topic is a little more specific. If you were connecting to a program that supplied times for different countries, the application might be “timeclock”, and the topics might be ”USA”, “Canada”, “Brazil”, “Vanuatu”, etc.

If the program is running on a different PC on a network , the sever name is prefixed with a \\NODENAME\ specifying the PC name. Thus, if there were two PCs running “timeclock”, each for a different hemisphere, you might use the following server and topics:

“\\NORTHERN\timeclock”, “USA”
“\\SOUTHERN\timeclock”, “Brazil”

Once a connection has been established, you typically make a request for or send an item of data, or send a command string for the server to execute. This is very specific to each application, and you must have the syntax for that application in order to establish a conversation with and/or make requests of it.

See Also:

{button ,JI(`GC.HLP',`IDH_How_Do_I_Use_DDE_Effectively')}   **DDE Tutorial**
{button ,JI(`GC.HLP',`IDH_DdeCommand')}   **DdeCommand**
{button ,JI(`GC.HLP',`IDH_DdeScript')}   **DdeScript**

GoTo

GoTo(*<label>***)**

GoTo jumps program execution to the label

Parameters: *label - Specifies the label to go to*

Returns: TRUE if successful; otherwise FALSE.

The function is used to move program execution to the line after the label. You would use this command as a way to branch execution to another part of your program.

Examples:

```
MessageBoxOkCancel("Do you want to continue?")
If(ResponseOk(),GoTo(":process"),ExitMacro())

:process
LogToFile("c:\test.log","Processing....")
```

Also See:

{button ,JI(`GC.HLP',`IDH_Labels')} Labels
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} Stopping a running script

Labels

Labels used to identify a location that is accessed using GoTo().

Labels are used to mark a location in a program. You can then route the execution of the program to the label using the GoTo() function. Label are defined as any text string beginning with a colon (":"). For example the following are all labels

:end
:process files
:error

Examples:

```
MessageBoxOkCancel("Do you want to continue?")
If(ResponseOk(),GoTo(":process"),ExitMacro())

:process
LogToFile("c:\test.log","Processing....")
```

Also See:

{button ,JI(`GC.HLP',`IDH_GoTo')} **GoTo**

Repeat

Repeat(*<command>, <number of times>***)**

Repeat executes a command a specified number of times

Parameters: *command - The command to execute*
times - The number of times to repeat the command

The Repeat command is used to repeat a command a specified number of times. If you want to repeat more than one command, you need to use a Repeat(RunMacro(), x) command where the commands you want to repeat are in a separate macro file. For more information on this, see the Tutorial on Loops and Branches.

Examples:

RunProgram("c:\winnt\notepad.exe")
Delay(2000)
Repeat(SendKeys("This is a test~"),10)

Also See:

{button ,JI(`GC.HLP',`IDH_While')} <u>While</u>
{button ,JI(`GC.HLP',`IDH_GoTo')} <u>GoTo</u>
{button ,JI(`GC.HLP',`IDH_RunMacro')} <u>RunMacro</u>
{button ,JI(`GC.HLP',`IDH_Loop_and_Branches')} <u>SendKeysFromFile</u>
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} <u>Stopping a running script</u>

While

While(*<condition>, <action>***)**

While accepts a condition function to evaluate every pass, and an action function to execute if true. If false, the loop exits:

Parameters:	*condition - function to evaluate*
	action - function to execute if the condition is found to be true

Examples:

```
// create an annoying noise
Repeat( Beep(), 50 )
```

If you want to repeat a group of statements, you are better off putting those statements in a separate file, and using the RunMacro with the repeat:

```
//do-while loop
While(FileExists("signal.txt"), RunMacro("chores.gc"))
```

See Also:

{button ,JI(`GC.HLP',`IDH_Loop_and_Branches')} **Loops and Branches**
{button ,JI(`GC.HLP',`IDH_RunMacro')} **RunMacro**
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} **Stopping a running script**

WaitForProgram

WaitForProgam(<*milliseconds to wait*>**)**

WaitForProgram halts program execution until the launched program completes.

Parameters: *milliseconds to wait - determines how long the program will wait*

WaitForProgram is used along with the RunProgram command to suspend macro execution until the launched program completes. You pass in the number of milliseconds to wait or -1 to wait indefinitely.

Examples:

RunProgram("c:\winnt\notepad.exe")
WaitForProgram(-1)

Also See:

{button ,JI(`GC.HLP',`IDH_RunProgram')} **RunProgram**

NormalizeWindow

NormalizeWindow()

NormalizeWindow restores the active window to its normal size

Parameters: *None.*

Returns: TRUE if successful; otherwise FALSE.

NormalizeWindow restores the currently active window to its original size. You can use this command to restore a window that has been minimized or maximized.

Examples:

NormalizeWindow()

Also See:

{button ,JI(`GC.HLP',`IDH_MinimizeWindow')} <u>**MinimizeWindow**</u>
{button ,JI(`GC.HLP',`IDH_MaximizeWindow')} <u>**MaximizeWindow**</u>

MinimizeWindow

MinimizeWindow()

MinimizeWindow minimizes the current active window

Parameters: *None.*

Returns: TRUE if successful; otherwise FALSE.

MinimizeWindow minimizes the currently active window. You can use this command to restore the active window to its original size. You can make sure the window is active first by using ActivateWindow.

Examples:

ActivateWindow("Untitled - Notepad")
MinimizeWindow()

Also See:

{button ,JI(`GC.HLP',`IDH_NormalizeWindow')} NormalizeWindow
{button ,JI(`GC.HLP',`IDH_MaximizeWindow')} MaximizeWindow
{button ,JI(`GC.HLP',`IDH_ActivateWindow')} ActivateWindow

MaximizeWindow

MaximizeWindow()

MaximizeWindow maximizes the current active window

Parameters: *None.*

Returns: TRUE if successful; otherwise FALSE.

MaximizeWindow maximizes the currently active window. You can use this command to restore the active window to its original size. You can make sure the window is active first by using ActivateWindow.

Examples:

ActivateWindow("Untitled - Notepad")
MaximizeWindow()

Also See:

{button ,JI(`GC.HLP',`IDH_NormalizeWindow')} **NormalizeWindow**
{button ,JI(`GC.HLP',`IDH_MinimizeWindow')} **MinimizeWindow**
{button ,JI(`GC.HLP',`IDH_ActivateWindow')} **ActivateWindow**

OnError

OnError(<label>**)**

OnError defines the label that will be called when an error occurs

Parameters: *label - Where to jump to if an error occurs.*

OnError defines a special label that program flow will go to whenever an error in the program occurs. You can use this to write your own error handling routine. LogError is handy for recording errors and events to a file.

Examples:

```
OnError(":error")
// your macro commands
LogToFile("mylog.txt","Macro completed successfully")
ExitMacro()

:error
LogError("Macro failed")
```

Also See:

{button ,JI(`GC.HLP',`IDH_GoTo')} GoTo

{button ,JI(`GC.HLP',`IDH_LogError')} LogError

SendNow

SendNow(*<format string>***)**

SendNow sends the current date and time as keystrokes

Parameters: *format string - The string that defines the date time format*

SendNow sends the current date and time as keystrokes to the current active window. The format of the string is defined by the format string parameter.

Date Formats for SendNow() command:

%a	Abbreviated weekday name = Mon
%A	Full weekday name = Monday
%b	Abbreviated month name = Feb
%B	Full month name = February
%c	Date and time representation appropriate for locale = 02/10/97 19:58:35
%d	Day of month as decimal number (01 – 31) = 10
%H	Hour in 24-hour format (00 – 23) = 19
%I	Hour in 12-hour format (01 – 12) = 07
%j	Day of year as decimal number (001 – 366) = 041
%m	Month as decimal number (01 – 12) = 02
%M	Minute as decimal number (00 – 59) = 58
%p	Current locale's A.M./P.M. indicator for 12-hour clock = PM
%S	Second as decimal number (00 – 59) = 35
%U	Week of year as decimal number, with Sunday as first day of week (00 – 51) = 06
%w	Weekday as decimal number (0 – 6; Sunday is 0) = 1
%W	Week of year as decimal number, with Monday as first day of week (00 – 51) = 06
%x	Date representation for current locale = 02/10/97
%X	Time representation for current locale = 19:58:36
%y	Year without century, as decimal number (00 – 99) = 97
%Y	Year with century, as decimal number = 1997
%z, **%Z**	Time-zone name or abbreviation; no characters if time zone is unknown = Pacific

Examples:

```
// the tildes (~) in the SendNow lines are just linefeeds so that the notepad display is easy to read
SendKeysToWindow("Untitled - Notepad",”The time is: ”)
SendNow("%X~”)
SendKeysToWindow("Untitled - Notepad",”The date is: ”)
SendNow("%x~”)
SendKeysToWindow("Untitled - Notepad",”The time zone is: ”)
SendNow("%z~”)
SendKeysToWindow("Untitled - Notepad",”Today is: ”)
SendNow("%A~”)
```

Also See:

{button ,JI(`GC.HLP',`IDH_SendKeys')} SendKeys
{button ,JI(`GC.HLP',`IDH_SendKeysToWindow')} SendKeysToWindow
{button ,JI(`GC.HLP',`IDH_SendEnvironment')} SendEnvironment
{button ,JI(`GC.HLP',`IDH_SendKeysFromFile')} SendKeysFromFile
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} Stopping a running script

SendEnvironment

SendEnvironment(<*message string*>**)**

SendEnvironment sends an environment variable as text to the active window.

Parameters: *message string - string to send, including environment variables.*

SendEnvironment expands strings containing environment variables and sends them as keystrokes to the currently active window. You can see all the environment variables on your PC by opening a command window (DOS Box) and type "SET".

You can specify a delay between characters in the System Options dialog.

Examples:

SendEnvironment("The temp path is %TEMP%")

Also See:

{button ,JI(`GC.HLP',`IDH_SendNow')} **SendNow**
{button ,JI(`GC.HLP',`IDH_SystemOptions')} **System Options**
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} **Stopping a running script**

SendInput

SendInput()

Parameters: *None.*

SendInput is used to send the text that was input from the user in the InputBox command. The text is sent as keystrokes to the active window.

You can specify a delay between characters in the System Options dialog.

Examples:

InputBox("Enter your name:")
ActivateWindow("Untitled - Notepad")
Delay(3000)
SendInput()

Also See:

{button ,JI(`GC.HLP',`IDH_InputBox')} InputBox
{button ,JI(`GC.HLP',`IDH_SendInputToWindow')} SendInputToWindow
{button ,JI(`GC.HLP',`IDH_SystemOptions')} System Options
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} Stopping a running script

PowerOff

PowerOff()

PowerOff is used to power down computers equipped with ability to turn off power automatically, like notebooks.

Parameters: *None.*

Example:

PowerOff()

Also See:

{button ,JI(`GC.HLP',`IDH_Logout')} Logout
{button ,JI(`GC.HLP',`IDH_RebootSystem')} Logout
{button ShutdownSystem,JI(`GC.HLP',`IDH_ShutdownSystem')}

LogSystem

LogSystem(*<File Name>,<Resource>***)**

LogSystem Log system information to a specified file.

Parameters: *File Name - Specifies the name of the file to write system information to.*
Resource - Specifies the type of resource information.

You can use LogSystem to track information such as free disk space and memory usage.
Available resource information includes:

Disk Information:	DISK_C_FREE
	DISK_C_SIZE
	DISK_X_FREE
	DISK_X_SIZE

You can replace the 'X' in the above commands with a different drive letter i.e. DISK_D_FREE

Memory Information:	MEMORY_AVAILPAGEFILE
	MEMORY_AVAILPHYSICAL
	MEMORY_AVAILVIRTUAL
	MEMORY_PERCENTUSED
	MEMORY_TOTALPAGEFILE
	MEMORY_TOTALPHYSICAL
	MEMORY_TOTALVIRTUAL
Environment Information:	SYSTEM_DIR
	WINDOWS_DIR
	USER_NAME

Examples:

LogSystem("C:\Test.log",DISK_C_FREE)

Also See:

{button ,JI(`GC.HLP',`IDH_LogToFile')} LogToFile
{button ,JI(`GC.HLP',`IDH_LogError')} LogError
{button ,JI(`GC.HLP',`IDH_WriteLineToFile')} WriteLineToFile
{button WriteToFile,JI(`GC.HLP',`IDH_WriteToFile')}

How Do I Use DDE Effectively?

DDE is a powerful tool if used correctly. It stands for Dynamic Data Exchange and it allows two or more Windows programs to make requests of each other via a well-structured syntax. Typically, it is used to request or send items of data, but it also allows for the transmission of commands to another program. DDE can be configured to work between computers on a network, so you can actually control programs on other PCs.

You can use DDE through GroundControl to send DDE commands to a server application if that server supports DDE commands. If it does, you will need to know the syntax it supports, and any peculiarities in the command language that may exist. A well-written DDE server will have a well-documented set of commands, representing at least most of the commands you would need to use.

To send a DDE Command, you need to establish a conversation with a server and topic. You then send the command as a carefully formatted string, and close the conversation. GroundControl does all of these for you with a simple function:

DDECommand("servername", "topicname", "carefully_formatted_command_string")

The server and topic for an imaginary editor might be "slicktyper" and "commands". The command string depends on what you want to do, but it must be supported by the server. To tell the slicktyper server to quit might require an "[Exit()]" command. To tell it to open a file might need a "[FileOpen('C:\myfile.txt)']".

If you are following closely, you might be thinking it is inefficient for GroundControl to open and close a conversation for every command, and you would be correct! If you have a long sequence of commands to send, then it is much better for you to put them into a file, and run DDEScript instead. This command opens the conversation once, then executes all the commands in the file before closing the link. It is better to use DDEScript if you have more than three or four commands to send.

Beware: some servers support more than one topic, and not all commands may go to the same topic. If you are using DDEScript, all commands go to the same topic, and you will have to separate them out.

{button ,JI(`GC.HLP',`IDH_What_s_the_advantage_of_DDE_over_SendKeys')} **DDE vs SendKeys**
{button ,JI(`GC.HLP',`IDH_DDE_Overview')} **DDE Overview**
{button ,JI(`GC.HLP',`IDH_DdeCommand')} **DdeCommand**
{button ,JI(`GC.HLP',`IDH_DdeScript')} **DdeScript**

What's the advantage of DDE over SendKeys?

You might think SendKeys is easier than DDE, and you would be right. SendKeys is does not require a conversation link, where DDE does. However, SendKeys can be confusing because you have to keep very close track of where your keys are going, which can get tricky if your windows lose focus or get hidden. DDE on the other hand, is like a solid pipe through which your commands can flow, and you don't have to worry about where it goes once you send it. The server does that work for you.

DDE has one other advantage. If your PC is part of a network, and NetDDE has been configured on another PC, you can send commands to a program on that PC! The network is invisible, and as long as you know the name of the machine, only the server name has to be modified. The example command above to open a slicktyper file on your machine becomes the following if you want to tell slicktyper on another machine to open a file:

```
DDECommand("\\remotepc\slicktyper", "commands", "[FileOpen('c:\myfile.txt')]" )
```

In this example, the remote PC would open a file called myfile.txt on its C drive, not yours. The topic and command are exactly the same, but the server name has been prefixed with "\\nodename\". SendKeys does not work across different PCs.

The only thing holding you back from sending DDE commands all over the network is any security the manager may have implemented using DDE Shares. These can be tricky to set up, and if you cannot get DDE commands working over the network, you may need to see your LAN manager and have them check that NetDDE is running on both PCs, and that the remote PC has a DDE Share for you.

GroundControl does math too?

Here's an example where GroundControl is an interactive front end for another program. Suppose we want to turn the Microsoft Calculator program into an interactive, MessageBox application. We will use GroundControl to ask for two number to multiply together, and write the answer to a file called answer.txt.

```
RunProgram("notepad")
RunProgram("calc")
Delay(1000)
InputBox( "Please enter the first number" )
SendInputToWindow("Calculator")
SendKeysToWindow("Calculator", "*" )
InputBox( "Please enter the second number" )
SendInputToWindow("Calculator")
SendKeysToWindow("Calculator","=")
// now get the number back, hmmmm...
SendKeysToWindow("Calculator","^C")
SendKeysToWindow("Untitled - Notepad","answer: ")
SendKeysToWindow("Untitled - Notepad","^V, and don't forget to check it.")
MessageBox("Done!")
```

This will send the two numbers to multiply to the Calculator, and uses the clipboard to get the answer back. GroundControl opens notepad, and types the answer into it. Easy, eh?

If you think about it, by using the clipboard, you have access to numbers from almost any application. Although clipboard doesn't work over a network, you could easily paste answers into a notepad text file, and save that on the network. Then use DDE or some file trigger on the remote PC to open that file. You can get complicated, but the opportunity is there.

Getting Started

Cool! You've just opened up a very powerful little program. You should probably take a look around some of the help topics listed below, and familiarize yourself with what's available. You'll probably realize that most of the commands are things you do every day without thinking about it, especially the basic windows commands, and these shouldn't present any problem. If you just can't wait to get started, select Tease Me! below.

{button ,JI(`GC.HLP',`IDH_Tease_Me_Now')} **Tease Me!**

If you're still here, good. If you are a novice programmer, don't worry, you're in good hands. There is a ton of help to assist you as you build programs, and not just the technical reference-type stuff either. (Check out the tutorial on Loops and Branches!). If you have had lots of programming experience, you'll find the If, For, Repeat, GoTo functions a cake-walk. You'll be more interested in the interactive dialogs, the SendKeys and DDE features. Some of these commands take a little more time to learn how to use properly, but they really let you to stretch out your control ability.

{button ,JI(`GC.HLP',`IDH_Application_Overview')} **Official High-Brow Program Overview**
{button ,JI(`GC.HLP',`IDH_Function_Descriptions_by_Category')} **Overview of Commands by Category**
{button ,JI(`GC.HLP',`IDH_Loop_and_Branches')} **Loops and Branches Tutorial**
{button ,JI(`GC.HLP',`IDH_How_Do_I_Use_DDE_Effectively')} **DDE Tutorial**
{button ,JI(`GC.HLP',`IDH_Tip_SendKeys')} **SendKeys Tutorial**
{button ,JI(`GC.HLP',`IDH_How_Do_I')} **How Do I ...**

If you made it past all these, there's just a few more words to say. If this is all new, don't worry, you'll be turning out DDE subroutines in no time. Learn how to use the toolbars - you may not need all of them but you'll wonder how you did without them. Read the help files whenever you have problems, and enjoy the product!

MessageBox

MessageBox (*<message>***)**

MessageBox creates a simple message box with text you specify, plus an OK button.

Parameters: *message - Specifies the text that will be displayed in the message box prompt.*

MessageBox enables you to display a message to the screen. The execution of the macro stops until the OK button is clicked.

Examples:

MessageBox("File not found. Press OK to continue.")

Also See:

{button ,JI(`GC.HLP',`IDH_ResponseOk')} ResponseOk
{button ,JI(`GC.HLP',`IDH_ResponseCancel')} ResponseCancel
{button ,JI(`GC.HLP',`IDH_MessageBoxOkCancel')} MessageBoxOkCancel
{button ,JI(`GC.HLP',`IDH_InputBox')} InputBox

RunDOSCommand

RunDOSCommand(<*DOS Command*>)

RunDOSCommand will execute the specified DOS command from a command prompt.

Parameters: *DOS Command - Specifies the name of the DOS command to execute.*

You can use RunDOSCommand to execute any command you can run from a command prompt (Copy, Del, Dir, etc.).

Examples:

RunDOSCommand("Copy C:\*.bat C:\Temp\*.")*

Also See:

{button ,JI(`GC.HLP',`IDH_RunProgram')} **RunProgram**

How Do I …

Here's some tips to help you get started:

Start another program
Check for disk space and stuff like that
Move windows around
Write stuff to a file
Interactively fill out a dialog box
Make a choice based on the result of a function
Ask the user for something, and make a decision on it
Write a Do-While Loop
How do I use DDE effectively

Using GroundControl with LaunchPad

GroundControl and LaunchPad are very complementary programs when used together. Since GroundControl lets you build your own Windows batch files, and LaunchPad lets you schedule unattended tasks, using them together really extends your power.

To run GroundControl macros from LaunchPad, follow these simple steps:

1. Create a macro using GroundControl and save it.
2. Run LaunchPad.
3. Click on the Clock icon to add an event.
4. Type in a description for your event ("Launch My Macro"), and then use the Open Folder icon at the end of the Command Line to browse through your directories until you find the Ground Control exe file (GC.EXE). In the ‘Command Line’ box   after the exe type a space then “/r” followed by the path to the GroundControl macro file you want.   Your command line might look something like this ‘gc.exe /r myfile.gc’
5. Press "Next" to select the "Schedule" tab sheet, and set the date and time you want to run the program.
6. Press the Checked Clock icon to enable this event, and the Frequency description should change from "Never" to "One Time".

Key Codes

Key	**Code**
Shift	+
Control	^
Alt	%
BackSpace	{BKSP}
Break	{BREAK}
Caps Lock	{CAPSLOCK}
Clear	{CLEAR}
Del	{DELETE}
Down Arrow	{DOWN}
End	{END}
Enter	{ENTER} or ~
Esc	{ESC}
Help	{HELP}
Home	{HOME}
Ins	{INSERT}
Left Arrow	{LEFT}
Num Lock	{NUMLOCK}
Page Down	{PGDN}
Page Up	{PGUP}
Print Screen	{PRTSC}
Right Arrow	{RIGHT}
Scroll Lock	{SCROLLLOCK}
Tab	{TAB}
Up Arrow	{UP}

F1	{F1}	F2	{F2}
F3	{F3}	F4	{F4}
F5	{F5}	F6	{F6}
F7	{F7}	F8	{F8}
F9	{F9}	F10	{F10}
F11	{F11}	F12	{F12}
F13	{F13}	F14	{F14}
F15	{F15}	F16	{F16}

{Activate}	{Delay}	{Date}	{Time}
{Day}	{Month}	{Year}	{Hour}
{Minute}	{Second}		

SendKeys Overview

You can define a series of keystrokes to automatically send as part of the macro using the SendKeys and SendKeysToWindow functions.

Each key is represented by one or more characters. To specify a single keyboard character, use the character itself. For example, to represent the letter A, use A for key text. If you want to represent more than one character, append each additional character to the one preceding it. To represent the letters A, B, and C, use ABC for key text.

The plus sign (+), caret (^), percent sign (%), tilde (~), and parentheses ( ) have special meanings to SendKeys. To specify one of these characters, enclose it inside braces. For example, to specify the plus sign, use {+}. Brackets ([ ]) have no special meaning to SendKeys, but you must enclose them in braces as well, because in other applications for Microsoft Windows, brackets do have special meaning that may be significant when dynamic data exchange (DDE) occurs. To send brace characters, use {{} and {}}. To specify characters that aren't displayed when you press a key (such as Enter or Tab) and keys that represent actions rather than characters, use the key codes defined in the SendKeys Codes.

Key Codes

To specify keys combined with any combination of Shift, Ctrl, and Alt keys, precede the regular key code with one or more of the following codes:

Key	**Code**
Shift	+
Control	^
Alt	%

To specify that Shift, Ctrl, and/or Alt should be held down while several other keys are pressed, enclose the keys' code in parentheses. For example, to have the Shift key held down while E and C are pressed, use "+(EC)". To have Shift held down while E is pressed, followed by C being pressed without Shift, use "+EC".

To specify repeating keys, use the form {key number}; you must put a space between key and number. For example, {LEFT 42} means press the Left Arrow key 42 times; {h 10} means press h 10 times.

Now, SendKeys will also recognize some special “sub-commands”. These are special instructions to implement delays, window commands, etc. They do the same thing as some of the other GroundControl macros, but they allow you to do it from within the SendKeys commands. This is very powerful if all your SendKeys commands are in a separate file, and you are executing the file with a single SendKeysFromFile command.

{Activate “<window title>”}

This sub-command will select the window named to be the active and top most window. The command is not case sensitive but the name must match the window exactly. For example, the following will bring notepad to the top-most window and all additional keys would be sent to this application:

{Activate "Untitled - Notepad"}

This has the same effect as the GroundControl command ActivateWindow(“Untitled - Notepad”).

{Delay <timeout>}

This sub-command will wait a determine time before sending the next key to the event. You specify the time in milliseconds following the command. For example, this will wait 1 second before execution of the next key:

{Delay 1000}

This has the same effect as the GroundControl command Delay(1000).

{Date [<offset>]}

This sub-command will send the current date as text. You can also define date math following the command. For example:

{Date 1}	will send tomorrow's date
{Date -1}	will send yesterday's date. You must leave a space between the command and the number.

{Time [<offset>]}

The Time sub-command sends the current time as text. You can also do math on this function by defining the amount of seconds after the time command. For example:

{Time 60}	would send the current time + one minute.
{Time -60}	would send the current time minus one minute.

{Day [<offset>]}

The Day sub-command will send the numeric value for the day of the current date. You can also do math on this function by defining the amount of days after the command. For example:

{Day 1}	would add one to the current day.
{Day -1}	would subtract 1 day.

{Month [<offset>]}

The Month sub-command will send the numeric value for the month of the current date. You can also do math on this function by defining the amount of months after the command. For example:

{Month 1}	would add one to the current month.
{Month -1}	would subtract one from the current month.

{Year [<offset>]}

The Year sub-command will send the numeric value for the Year of the current date. You can also do math on this function by defining the amount of Years after the command. For example:

{Year 1}	would add one to the current year.
{Year -1}	would subtract a year.

{Hour [<offset>]}

The Hour sub-command will send the numeric value for the current hour. You can also do math on this function by defining the amount of hours after the command. For example:

{Hour 1}	would add one hour.
{Hour -1}	would subtract a hour.

{Minute [<offset>]}

The Minute sub-command will send the numeric value for the current minutes. You can also do math on this function by defining the amount of minutes after the command. For example:

{Minute 1}	would add one minute.
{Minute -1}	would subtract a minutes.

{Second [<offset>]}

The Second sub-command will send the numeric value for the current second. You can also do math on this function by defining the amount of seconds after the command. For example:

{Second 1}	would add one second.
{Second -1}	would subtract a second.

You can also specify a delay between each character in the system Options dialog.

Also See:

{button ,JI(`GC.HLP',`IDH_Tip_SendKeys')} **<u>SendKeys Tutorial</u>**
{button ,JI(`GC.HLP',`IDH_SystemOptions')} **<u>System Options</u>**

SendKeysFromFile

SendKeysFromFile (*<filename>*)

SendKeysFromFile reads the specified file and sends the contents one character at a time to the active window.

Parameters: *filename - Specifies the file containing the text to send.*

The advantage of this command over the SendKeys command is that you can put everything you want to send in one file, and execute the entire file with one statement. You can change which keys go to which window within the file, so you don't have to have a file for each program.

You can break the keystrokes out onto different lines. For example, sending a file that looks like this:

```
H
e
l
l
o
!
~
```

is no different than sending a file that looks like this:

```
Hello!~
```

Examples:

Here's the contents of a small file called commands.txt:

```
%FN
Hello from inside the command file
~
%FA
saved.txt
~
```

Here's the macro file you can use to send it to notepad. This file is called macro.gc:

```
//send the contents of the autoexec.bat file to notepad, in effect, making a copy
ActivateWindow("Untitled - Notepad")
SendKeysFromFile("commands.txt")
```

The net result is to send a line of text to notepad, then send it the ALT-F, A combination, which translates into the File \ Save As commands, and then sends the file name to save the line as. Note how the commands.txt file can contain both the text to send and the special control codes to manipulate the menus. The remote application, in this case, notepad, will interpret the commands as though you were typing the contents of the file in yourself.

Also See:

{button ,JI(`GC.HLP',`IDH_Key_Codes')} Key Codes
{button ,JI(`GC.HLP',`IDH_Tip_SendKeys')} Using SendKeys
{button ,JI(`GC.HLP',`IDH_SystemOptions')} System Options
{button ,JI(`GC.HLP',`IDH_Stopping_a_Running_Script')} Stopping a running script

FileOlderThan

FileOlderThan(*<filename>, <age>***)**

FileOlderThan returns TRUE if the file is older than the allowed age.

Parameters: *filename - name of the file to examine*
age - maximum allowed age in seconds

Returns: TRUE if the file is older than the specified age in seconds.
FALSE if the file cannot be found or it is younger than the specified age.

Examples:

FileOlderThan("a.txt",86400)

Order Information

When you purchase a registered copy of GroundControl you will receive a registration key, that when installed, will remove the shareware reminder screens from the application startup. All orders for GroundControl are subject to the GroundControl licensing agreement.

You can also order a diskette copy of the application for additional cost.

There are several ways to order GroundControl. For information on these methods click on the items below

{button ,JI(`GC.HLP',`IDH_Pricing')}   Pricing
{button ,JI(`GC.HLP',`IDH_Order_Form')}   Order Form
{button ,JI(`GC.HLP',`IDH_Ordering_by_Check_or_Money_Order')}   Ordering by Check or Money Order
{button ,JI(`GC.HLP',`IDH_Ordering_with_Credit_Card')}   Ordering using Compuserve SWREG
{button ,JI(`GC.HLP',`IDH_Ordering_with_Purchase_Order')}   Ordering with Purchase Order

Comment

A Comment is not a command, but a macro line that will not be executed.

Comment places two forward slashes at the front of the macro line, and these line are not executed.   Comments can be used in two different ways.   One use for comments is to make notes in your macros to document what the macro is doing at that point in its execution. Another is to disable lines for testing or debugging.

Examples:

```
//My Backup Macro
//Now check to see if the file was created and run the delete temps macro
If(FileExist("c:\temp\backups.log","RunMacro("DelTemps.grn")","ExitMacro()")
//Now Shutdown   un-comment the next line after testing
//ShutdownSystem()
//if testing just exit macro
ExitMacro()
```

System Options

The System Options allow you a little extra flexibility in tuning GroundControl.

Send Keys Delay

This allows you to specify the millisecond spacing between each character as it is transmitted.

Macro Editor

This option allows you to specify an external editor to use to edit the macro. If you would rather use your favorite editor to build your macro instead of GroundControl, simply choose Edit\External Editor, and GroundControl will launch your editor and load your macro into it. When you exit your editor, make sure you save your macro, and GroundControl will load it back in, if you choose to do so.

Print Line Numbers

This is used when you are printing out your macro. If checked, it simply puts the line number in front of the macro text.

Event Log

This is where you define the Event Log path and name that you want GroundControl to use whenever the LogEvent function is called.

Error Log

This is where you define the Error Log path and name that you want GroundControl to use whenever the LogError function is called.

Stopping a Running Script

Whenever a GroundControl script is running, there will be a GroundControl Icon in the systray for Windows 95 and Windows NT 4.0   Clicking on this icon will stop the running script immediately without running to its completion.

LogEvent

LogEvent(*<message>***)**

LogEvent writes an entry in the error log file

Parameters: *message - Specifies the text to be written to the log file.*

Use LogEvent to write custom messages with a time and date stamp to the default log file, called macsys.log. Your macro can post messages to the system log based on conditions in the macro. This is useful for tracing automated jobs. The log file looks something like this:

```
01/24/97 19:50:20 User said no to my dialog box
01/24/97 19:50:25 The Calculator program wasn't running.
```

If you want to use LogEvent as a means to trace program execution, but you don't want to clutter the log with general messages, you can use LogToFile instead, which does exactly the same thing, except you specify which file to write the message to. This means you can have many different logfiles, each for different purposes.

Examples:

If(FileExist("C:\WIN95\win.ini"),LogEvent("File found"),Beep())

Also See:

{button ,JI(`GC.HLP',`IDH_WriteToFile')} **WriteToFile**
{button ,JI(`GC.HLP',`IDH_WriteLineToFile')} **WriteLineToFile**
{button ,JI(`GC.HLP',`IDH_LogToFile')} **LogToFile**

Order Form

```
Name...: __________________________________________________
Company: __________________________________________________
Address: __________________________________________________
City...: _____________________ State:____ Zip Code:________

Email Address: ____________________________
Phone Number.: ____________________________


Num. Copies_____   at _______ each          = ______________

CA Residents add local sales tax (7.25%)   + ______________

Please indicate whether or not a diskette copy
is being ordered. Add the additional cost
for shipping and handling to the total.

Diskettes Yes ___ No: __

United States........................$ 6.00
Mexico and Canada....................$ 8.00
All Others...........................$10.00 + _____________


                              Total Amount ________________



Send Completed Form and Payment To:

Cypress Technologies
4450 California Ave. Suite K-165
Bakersfield, CA 93309
```

Ordering by Check or Money Order

Print and Fill Out the order form and mail with check or money order to:

Cypress Technologies
4450 California Ave. Suite K-165
Bakersfield, Calif. 93309

To Print the order form click on Order Form from the content page and pull down the help file menu and select print topic.

Payments must be made in US dollars or drawn on a US bank.

Ordering using Compuserve SWREG

To register by credit card through CompuServe, you need to have a CompuServe account. Log on to Compuserve. At any prompt type "GO SWREG"   Select the registration Id# for the version you want to order.

REG ID#	DESCRIPTION
14709	GroundControl (No Disk)
14710	GroundControl (w/disk)
14711	GroundControl Site License 1 - 4 computers
14712	GroundControl Site License 5 - 24 computers
14713	GroundControl Site License 25 - 49 computers
14714	GroundControl Site License 50 - 99 computers

The registration fee will be billed to your CompuServe account.

Ordering with Credit Card

Public Software Library

You can order with MC, Visa, Amex, or Discover from Public Software Library by calling:

800-2424-PSL
713-524-6394
713-524-6398 (FAX)

When ordering use the following Id # "**15152**"

- Ask operator to Note on the order the version you desire. Windows 3.1 or Windows 95

- Also you will need to specify whether or not you want a diskette copy.

THE ABOVE NUMBERS ARE FOR ORDERS ONLY. Any questions about the status of the shipment of the order, registration options, product details, technical support, volume discounts, dealer pricing, site licenses, etc., must be directed to Cypress Technologies. at Email address support@cypressnet.com or the address below.

Cypress Technologies
4450 California Ave. K-165
Bakersfield, CA 93309

Ordering with Purchase Order

Purchase orders (net 30 days) are accepted only from government and accredited educational institutions and major corporations, provided that they are submitted on purchase order forms with a purchase order number. Please be sure to include the standard GroundControl order form with each purchase order.

Send Purchase Order /w Order Form to:

Cypress Technologies
4450 California Ave. Suite K-165
Bakersfield, CA. 93309

Pricing

Cost for GroundControl is $39.95 for a single copy. When ordered you will receive a registration key that will remove the shareware screen that appears each time GroundControl is started. For additional cost you can also receive a diskette with the latest copy of GroundControl. This cost is dependent on where the diskette will be shipped.

Diskette Cost:

United States	$ 6.00
Mexico and Canada	$ 8.00
All Others	$10.00

Site Licensing

Site licensing is available based on the pricing structure below. A site licensing agreement receives 1 copy of the application diskette and allows you to make up to the number of copies you order. For information on ordering a site license contact us by one of the Email address listed in the support section of the file,

Site License:

1	to	4	computers: $39.95 each
5	to	24	computers: $31.95 each
25	to	49	computers: $28.95 each
50	to	99	computers: $26.95 each
100+			computers: CALL