Welcome

Welcome to the AutoMate™ COM Objects Help.

The AutoMate™ COM Objects provide a programmatic interface to the power of AutoMate™. Use these objects to change task properties.

CurrentTask Object

Properties Methods

NOTE: The CurrentTask object has been deprecated. Use the CurrentTask2 object instead.

The CurrentTask object encapsulates the properties of a task, and methods that act upon that task.

Description

The CurrentTask object contains properties and methods to allow you to view and modify attributes of the currently executing task. Use this object to control specific parts of the executing task "on- the-fly."

When a script is started, AutoMate™ automatically instantiates an CurrentTask object and populates it with the properties of the currently running task. This object is referred to as "CurrentTask." Use CurrentTask to access and modify the properties of the currently running task, as well as performing actions on the task, such as rescheduling and saving it to a file.

CurrentTask Properties

CurrentTask Legend

- TaskName
- LaunchDate
- LaunchTime
- Frequency
- Interval
- IntervalType
- RunLate
- ScheduleLate
- Active
- Triggers
- Hotkey
- WindowName
- StepCount
- Details
- FrequencyString
- Events

TaskName Property

CurrentTask Example

Specifies the name of the currently executing task.

Declaration

Dim TaskName As Variant

Description

Use the TaskName property to retrieve or set the name of the currently running task. If you change the name of the task during execution of your script, be sure it is unique. Problems may arise in the AutoMate™ Configuration Manager if you attempt to set the task to a name that already exists in your tasklist.

Accessibility

- Read Only Property

LaunchDate Property

CurrentTask Example

Specifies the date the task is next scheduled to launch.

Declaration

Dim LaunchDate As Variant

Description

Use the LaunchDate property to retrieve or set the date the task is scheduled to launch. The property is set when the task begins execution. The date format must follow the format set in Regional Settings. The new LaunchDate is set when the task finishes executing.

To view the format required for your regional settings, view Regional Settings from the Control Panel. Click the ‘‘Date’’ tab and note the ‘‘Short Date Format.’’ This is the format that AutoMate™ expects.

NOTE: Be careful when changing the LaunchDate of a task that uses the Schedule trigger. Because the LaunchDate is set when the task *starts* executing, the schedule trigger may change the date on you while the task is executing. Therefore, use caution when adjusting the LaunchDate relative to its initial value.

LaunchTime Property

CurrentTask Example

Specifies the time the task is next scheduled to launch.

Declaration

Dim LaunchTime As Variant

Description

Use the LaunchTime property to retrieve or set the time the task is scheduled to launch. The property is set when the task begins execution. The time format must follow the format set in Regional Settings. The new LaunchTime is set when the task finishes executing.

To view the format required for your regional settings, view Regional Settings from the Control Panel. Click the "Time" tab and note the "Time Style." This is the format that AutoMate™ expects.

NOTE: Be careful when changing the LaunchTime of a task that uses the Schedule trigger. Because the LaunchTime is set when the task *starts* executing, the schedule trigger may change the time on you while the task is executing. Therefore, use caution when adjusting the LaunchTime relative to its initial value.

Frequency Property

CurrentTask Example

Specifies the how often the schedule task runs.

Declaration

Dim Frequency As Integer

Description

Use the Frequency property to specify how often the task will run.

The Frequency property is an integer value. Use one of the following three integer values to specify how the task reschedules.

Value	Meaning	Description
0	Once	The task will run once, and become inactive afterwards
1	Every	After running, the task will reschedule itself based on the values on IntervalType, Interval, RunLate and ScheduleLate properties)
2	Manual	Will run only when explicitly told to do so, ignoring the schedule.

IntervalType Property

CurrentTask Example

Specifies the interval at which the task will be rescheduled.

Declaration

Dim IntervalType As Integer

Description

Use the IntervalType property to specify the denomination at which the task is to be rescheduled. Use in conjunction with the Frequency property set to "Every" (integer value 1).

Set IntervalType to one of the following integer values to specify how the task is rescheduled:

Value	Meaning	Description
0	Minute	Schedules the task every x minutes
1	Hour	Schedules the task every x hours
2	Day	Schedules the task every x hours
3	Week	Schedules the task every x weeks
4	Weekday	Schedules the task every x weekdays
5	Weekend	Schedules the task every x weekend
6	Bi-week	Schedules the task every x bi-weeks (i.e. every 2(x) weeks)
7	Month	Schedules the task every x months
8	Quarter	Schedules the task every x quarters (i.e., every 4(x) months)
9	Year	Schedules the task every x years
10	Seconds	Schedules the task every x seconds

where x is the interval specified by the Interval property.

Interval Property

CurrentTask Example

Specifies the scalar at which to reschedule the task. Used in conjunction with the IntervalType.

Declaration

Dim Interval As Integer

Description

Set Interval to the integer value specifying the scalar amount to use with the IntervalType. For example, if IntervalType is set to “3” (for “Weeks”), setting the Interval to “2” will cause the task to reschedule every 2 weeks.

RunLate Property

CurrentTask Example

Controls run action when the task is overdue.

Declaration
Dim RunLate As Integer

Description
Set RunLate to one of the following integer values to specify how the task should react to being late:

Value	Meaning	Description
0	Immediately	The task is run immediately when the task becomes late.
1	Don't Run	The task is not to run at all.
2	Prompt	A dialog box appears, asking whether or not to run the task.

Use with ScheduleLate to specify how the task is reschedule when the task is late.

ScheduleLate Property

CurrentTask Example

Controls reschedule when the task is overdue.

Declaration

Dim ScheduleLate As Integer

Description

Set ScheduleLate to one of the following integer values to specify how the task should be reschedule when the task is late:

Value	Meaning	Description
0	Relative to original date/time	The task is rescheduled relative to the original date and time the task was set to launch. (For example. if the task was scheduled to launch at 10:51, and reschedule every 2 hours, but actually launches at 11:01, the task will reschedule to 12:51
1	Relative to launch date/time	The task is schedules relative to the time the task was launch. (For example, if the task was scheduled to launch at 9:51, and reschedule every 2 hours, but actually launches at 10:01, the task will reschedule to 12:01)
2	Don't reschedule	The task will not reschedule itself, and instead set itself to an inactive state.

Active Property

CurrentTask Example

Specifies whether or not the task is active.

Declaration

Dim Active As Integer

Description

An active task is one that responds to a trigger. A task set to launch on a schedule or hotkey or any other trigger will not trigger if it is set to an inactive state. Inactive tasks, however, can still be launched manually through the AutoMate™ Configuration Manager.

Setting the Active property of a task while the task is executing an AutoMate™ BASIC script will stop the task from running any steps after a SCRIPT step.

Set the Active property to one of the following integer values to specify its active state:

Value	Meaning	Description
0	Inactive	The task is inactive (or stops, if it is currently running)
1	Active	The task is active and responds to triggers

TaskTriggers Property

CurrentTask Example

Specifies the triggers that will launch the task.

Declaration

Dim TaskTriggers As Integer

Description

The TaskTriggers property is an integer value that specifies which triggers will launch the task. To set the triggers, add the following values together from the table below:

Value	Trigger	Associated Properties
0	No Trigger	None; the task is launched manually only
1	Scheduled	LaunchDate, LaunchTime, Frequency, IntervalType, Interval, RunLate, ScheduleLate
2	Windows Events	Events
4	Wait for a Window	WindowName
8	Hotkey	Hotkey

For example, to set the task to launch on a schedule and a hotkey, the value of the TaskTriggers property would be 9 (1 + 8 = 9).

Hotkey Property

CurrentTask Example

Specifies the hotkey that is used with the hotkey trigger to launch the task.

Declaration

Dim Hotkey As Variant

Description

Set the Hotkey property to the hotkey to be used to launch the task when one of the triggers is the Hotkey trigger.

The Hotkey text must be specified in AutoMate's "Send Keystrokes" format:

Modifier	Character to Use
ALT	^
CTRL	%
SHIFT	&

For example, to use Ctrl-Alt H for a hotkey, the property should be set to %^H.

Remarks

Please note that the characters used to specify a hotkey are different than the characters used to send the specific key. In other words, in AutoMate and VBA "^c" will **send** CTRL + c. If you set the **hotkey** to "^c" it will be the key combination ALT + c.

WindowName Property

CurrentTask Example

Specifies what window to wait for when the Wait For Window trigger is specified for the task.

Declaration
Dim WindowName As Variant

Description
Set the WindowName property to the full string of the window to wait for to launch the task.

StepCount Property

CurrentTask READ ONLY Example

The number of steps in the task.

Declaration
Dim StepCount As Integer

Description
Read the StepCount property to find out the number of steps in the current task.

Details Property

CurrentTask Example

Stores the details of the task in one continuous string.

Declaration

Dim Details As Variant

Description

The Details property stores the details of the task. The return value is a continuous string, with individual lines separated by a CR-LF combination.

FrequencyString Property

CurrentTask READ ONLY Example

Contains the frequency and scheduling information about the task in a readable format.

Declaration

Dim FrequencyString As Variant

Description

The FrequencyString property combines the scheduling properties of the task and formats it into a sentence form for easier reading.

Events Property

CurrentTask

Specifies the window event triggers that will launch the task. Use in conjunction with the "Windows Events" value added to the Triggers property of the task.

Declaration

Dim Events As Integer

Description

The Events property is an integer value that specifies which windows' system events will launch the task. To set the window event triggers, add the following values together from the table below:

Value	Trigger	Associated Properties
0	None	No Windows Events triggers
1	System Time	System time has changed (NOTE: Windows will trigger this event each time you click on the System Calendar, regardless of whether or not you click "OK", "Apply" or "Cancel" when setting the system time.
2	Low Memory	The system is getting low on memory.
4	System Device	A system device has been added, removed or its properties have been changed.
8	Display Mode	The display mode (resolution or driver) has changed.
16	Color Palette	The color palette has changed.
32	Spooler	A print job has been added or removed from a printer queue.

For example, to set the task to launch on a system time change and a display mode change, the value of the property would be 9 (1 + 8 = 9).

CurrentTask methods

CurrentTask

- GetTaskStep
- GetStepCommand
- GetStepParamCount
- SetTaskStep
- GetStepParameter
- AppendTaskStep
- InsertTaskStep
- DeleteTaskStep
- GetStepStatus
- SetStepStatus
- ClearSteps
- SaveToFile
- LoadFromFile

GetTaskStep Method

CurrentTask Example

Retrieves the string containing the specified task step.

Declaration

Function GetTaskStep (StepNo As Integer) As Variant

Description

The first step is numbered 0. The string contains the task step command and parameters in the format that appears in the AutoMate™ step windows.

GetStepCommand Method

CurrentTask Example

Retrieves the string containing the specified task step command.

Declaration

Function GetStepCommand (StepNo As Integer) As Variant

Description

The first step is numbered 0. The string contains the task step command in the format that appears in the AutoMate™ step windows.

GetStepParamCount Method

CurrentTask Example

Returns the number of parameters in the specified task step.

Declaration

Function GetStepParamCount (StepNo As Integer) As Variant

Description

The first step is numbered 0. Use this function to return the number of parameters in the task step number specified by StepNo.

SetTaskStep Method

CurrentTask Example

Changes the specified task step.

Declaration

Sub SetTaskStep (StepNo As Integer, NewAction As Variant)

Description

Changes the step number specified by StepNo to the contents of the NewAction parameter. The NewAction parameter must be passed in the correct format for the new step to be valid.

GetStepParameter Method

CurrentTask Example

Returns a string containing the specified task step parameter.

Declaration

Function GetStepParameter (StepNo As Integer, Parameter as Integer) As Variant

Description

The first step is numbered 0 and the first parameter is numbered 0.

AppendTaskStep Method

CurrentTask Example

Adds a new task step.

Declaration

Sub AppendTaskStep (Action As Variant)

Description

Adds the new task step specified by the Action parameter to the end of the task step. The Action parameter must be in the correct format for the new step to be valid.

InsertTaskStep Method

CurrentTask Example

Inserts a new task step before the specified step.

Declaration

Function GetStepParamCount (StepNo As Integer) As Variant

Description

Inserts the new task step specified by the Action parameter before the step specified by BeforeStep. The Action parameter must be in the correct format for the new step to be valid.

DeleteTaskStep Method

CurrentTask Example

Deletes the specified step.

Declaration
Sub DeleteTaskStep (StepNo As Integer)

Description
Deletes the step specified by the StepNo parameter.

GetStepStatus Method

CurrentTask Example

Retrieves the status flag for the specified step.

Declaration

Function GetStepStatus (StepNo As Integer) As Integer

Description

Returns a value whether or not the step specified by StepNo is active or not. If the status is “0”, the step is disabled (i.e., the step will be skipped when the task is executed); if the status is “1”, the step is enabled.

SetStepStatus Method

CurrentTask Example

Retrieves the status flag for the specified step.

Declaration

Function SetStepStatus (StepNo As Integer)

Description

Specifies whether or not the step specified by StepNo is active or not. Setting the status to “0” means the step is disabled (i.e., the step will be skipped when the task is executed); setting the status to “1” means the step is enabled.

ClearSteps Method

CurrentTask Example

Clears all the steps in the task.

Declaration
Sub ClearSteps

Description
Deletes all the steps from the task and sets the StepCount to 0.

SaveToFile Method

CurrentTask Example

Saves the task to a file.

Declaration

Sub SaveToFile (FileName As Variant)

Description

Use SaveToFile to export the task in an AMOLE readable format to the filename specified by FileName. (The AMOLE architecture currently does not support the new STFF file format.) The task can then be imported to another installation of AutoMate™ on another machine using the LoadFromFile method of the CurrentTask object.

LoadFromFile Method

CurrentTask Example

Loads the task data from a file.

Declaration

Sub LoadFromFile (FileName As Variant)

Description

Use LoadFromFile to import a task specified in the FileName parameter into the task. The task to be loaded must have previously been saved in an AMOLE compatible format using the SaveToFile method. The task's information is overwritten by the information from the task file.

LoadFromFile Example

The following example loads the STFF file “c:\sample_task.amt”, replacing the current contents of the CurrentTask object with the contents of the sample_tast.amt file.

```
Sub Main
    CurrentTask.LoadFromFile(“c:\sample_task.amt”)
End Sub
```

SaveToFile Example

The following example saves the task contents of the CurrentTask object to a STFF file named “c:\my_task.amt”. The task can then be imported to another AutoMate™ installation.

```
Sub Main
    CurrentTask.SaveToFile(“c:\my_task.amt”)
End Sub
```

SetStepStatus Example

The following example sets the second step of the task to a “disabled” step, meaning it will be skipped when the task is executed.

```
Sub Main
    CurrentTask.SetStepStatus (1, 0)
End Sub
```

GetStepStatus Example

The following example stores the current status of the second step of the task and displays a message box with the result. A “0” result means the step is inactive, while a “1” result means it is active.

```
Sub Main
    Dim Status As Integer

    Status = CurrentTask.GetStepStatus (1)
    MsgBox Status
End Sub
```

DeleteTaskStep Example

The following example will delete the fourth step of the task. (Note that step indexing starts at “0.” Therefore, a value of “3” in DeleteTaskStep will remove the fourth step in the task.)

```
Sub Main
    CurrentTask.DeleteTaskStep (3)
End Sub
```

InsertTaskStep Example

The following example will insert a “Send Keystrokes” step into the current task as the fourth step.

```
Sub Main
    CurrentTask.InsertTaskStep (3, “SEND: AutoMate™ Version 4”)
End Main
```

This example will copy the third step, and place the copy above itself.

```
Sub Main
    Dim StepToCopy As Variant

    StepToCopy = CurrentTask.GetTaskStep (2)
    CurrentTask.InsertTaskStep (2, StepToCopy)
End Sub
```

AutoMate Object

Properties Methods

NOTE: The AutoMate object has been deprecated. Use the AutoMate2 object instead.

The AutoMate object encapsulates the properties of the AutoMate™ Configuration Manager and provides methods to manipulate the execution of a task.

Description

The AutoMate object provides properties that provide information about the current state of the AutoMate™ Configuration manager and the current task list. It also provides methods that allow you to change the running and execution flow of the current task, such as waiting for a window to appear or running another task.

AppendTaskStep Example

The following example will append a ‘‘SEND: AutoMate’’ action to the end of the list of steps of the current task.

```
Sub Main
    CurrentTask.AppentTaskStep (‘‘SEND: AutoMate’’)
End Sub
```

GetStepParameter Example

The following example stores the first parameter of the second step into a variable called “Param” and then displays the parameter in a message box.

```
Sub Main
    Dim Param As Variant

    Param = CurrentTask.GetStepParameter (1, 0)
    MsgBox Param
End Sub
```

SetTaskStep Example

The following example changes the second step of the task to a ‘‘Send Keystrokes’’ action.

```
Sub Main
    CurrentTask.SetTaskStep (1, ‘‘SEND: AutoMate’’)
 End Sub
```

GetTaskStep Example

The following example stores the first step of the task into a variable called “FirstStep” and displays the result in a message box.

```
Sub Main
    Dim FirstStep As Variant

    FirstStep = CurrentTask.GetTaskStep (0)
    MsgBox FirstStep
End Sub
```

AutoMate properties

- LastRASErrorCode
- LastRASErrorText
- RASHandle
- SystemDir
  TaskCount
- TaskList
- WindowsDir

AutoMate methods

AutoMate

- FindWindowByTitle
- FindWindowByClass
- FindWindowContaining
- RASConnect
- RASDisconnectEx
- RunTask

LastRASErrorCode Property

AutoMate READ ONLY

Contains the error code of the last RAS command attempted.

Declaration

Dim LastRASErrorCode As Integer

Description

The LastRASErrorCode property contains the integer error code of the last RAS command attempted in the script. Use LastRASErrorCode to find out why a particular RAS command, such as RASConnect or RASDisconnect, failed.

LastRASErrorCode returns “0” if the last RAS command was successful; otherwise it returns a non-zero value.

LastRASErrorText Property

AutoMate READ ONLY

Contains the error text of the last RAS command attempted.

Declaration

Dim LastRASErrorText As Variant

Description

The LastRASErrorText property contains a textual explanation of the error code of the last RAS command attempted.

RASHandle Property

AutoMate READ ONLY

NOTE: This method has been deprecated. Use the DialUp method instead.

Contains the RAS handle to a previously established dial-up connection.

Declaration

Dim RASHandle As Integer

Description

The RASHandle property contains a handle to an active dial-up connection established using the RASConnect method. The property is “0” if no active dial-up connection is present for this task.

SystemDir Property

AutoMate READ ONLY

Contains the System directory of the system.

Declaration

Dim SystemDir As Variant

Description

The SystemDir property contains the fully qualified path to the System directory of the current installation.

TaskCount Property

AutoMate

Contains the number of tasks in the current task list.

Declaration
Dim TaskCount As Integer

Description
Read TaskCount to obtain the total number of tasks in the task list currently in use by AutoMate™.

TaskList Property

AutoMate READ ONLY

Contains the path to the task list in use by AutoMate™.

Declaration

Dim TaskList As Variant

Description

Use the TaskList property to obtain the fully qualified path to the task list file AutoMate™ is currently using. A task list is a collection of tasks.

WindowsDir Property

AutoMate READ ONLY

Contains the Windows directory of the system.

Declaration

Dim WindowsDir As Variant

Description

The WindowsDir property contains the fully qualified path to the Windows directory of the current installation.

FindWindowByTitle Method

AutoMate Example

Returns the handle of the window that matches a specified window title.

Declaration

Function FindWindowByTitle (WindowTitle As Variant) As Integer

Description

Use FindWindowByTitle to obtain a handle to the window with the title specified in WindowTitle. The window must be open for this method to be successful. The method returns “0” if the window is not found, or it returns a non-zero value (the window handle) if the window is located.

FindWindowByClass Method

Returns the handle of the window that matches a specified class name.

Declaration

Function FindWindowByClass (WindowClass As Variant) As Integer

Description

Use FindWindowByClass to obtain a handle to the window with the class name specified in WindowClass. The window must be open for this method to be successful. If multiple windows with the specified class name are open, FindWindowByClass will return the handle to the first window it enumerates. The method returns "0" if the window is not found, or it returns a non-zero value (the window handle) if the window is located.

FindWindowContaining Method

Returns the handle of the window that contains a specified text.

Declaration

Function FindWindowContaining (TextToFind As Variant) As Integer

Description

Use FindWindowContaining to obtain a handle to the window that contains the text specified by TextToFind. The window must be open for this method to be successful. If multiple windows containing the specified text are open, the method will return the handle to the first window it finds. The method returns "0" if there are no windows with the text, or it returns a non-zero value (the window handle) if the window is located.

RASConnect Method

NOTE: This method has been deprecated. Use the DialUp method instead.

Attempts a dial-up connection using a specified phonebook entry, and returns a handle to the connection on success.

Declaration

Function RASConnect (PhonebookEntry As Variant) As Integer

Description

Use RASConnect to attempt to establish a dial-up connection using the phonebook entry supplied in PhonebookEntry.

On success, RASConnect returns a handle to a RAS object, which can be used in subsequent FTP related calls such as FTPUpload. RASConnect returns "0" if the connection fails. In this case, use RASLastErrorCode and RASLastErrorText for information about why the method failed.

RASDisconnectEx Method

AutoMate Example

NOTE: This method has been deprecated. Use the DialUp method instead.

Attempts to disconnect a previously established dial-up connection.

Declaration

Function RASDisconnectEx (PhonebookEntry As Variant) As Integer

Description

Use RASDisconnectEx to disconnect an active dial-up connection. The connection does not have to have been established by AutoMate™. Pass the name of the connection to be disconnected in the PhonebookEntry variable.

RunTask Method

Runs another task.

Declaration

Sub RunTask (TaskName As Variant)

Description

Use RunTask to start another task. The task specified in TaskName must be in the current TaskList (use the TaskList property to obtain the path to the current task list in use by the AutoMate object). The task run by RunTask executes asynchronously with other tasks (i.e., the execution of the script using the RunTask method will not stop).

Action Object

Properties Methods

NOTE: The Action Task object has been deprecated. Use the Action2 object instead.

The Action object encapsulates the actions available from the Step Builder (i.e. the available actions in AutoMate™).

Description

The Action object provides an interface to the available actions in AutoMate™. This enables a task developer to utilize variables from the BASIC language and pass them into the AutoMate™ action parameters.

There are a few limitations to be aware of in the Action object:

1) 1) No return codes for success or failure.
2) 2) All methods run asynchronously (i.e., they return immediately). Therefore, you should not perform a loop on an action without adding a delay to account for the amount of time it may take for the action to complete.

Action Properties

Action Legend

No properties

Action Methods

Action

- ChDir
- CloseWind
- CopyFile
- DDE
- DeleteFile
- FindText
- Focus
- FocusEx
- FTPDownload
- FTPUploadEx
- Hide
- MakeDir
- Maximize
- Minimize
- MoveFile
- OpenFile
- OSCommand
- PlaySound
- PrintFile
- RemDir
- RenameFile
- Restore
- RunScript
- SendKeys
- SendMail
- Start
- Unhide

PlaySound Method

Action Example

Encapsulates the AutoMate™ action ‘‘Play a sound.’’ Plays a WAV file.

Declaration
Procedure PlaySound(FileName, Wait: OleVariant); safecall;

Description
Specify the path and filename in the FileName parameter of the sound to be played.

PlaySound Example

This example will play a .wav file named “2010_09.wav” from the AutoMate™ directory.

```
Sub Main
    'Play the sound file from the AutoMate™ directory
    'the %AMDIR% is an AutoMate™ system constant that
    'can only be used with the AutoMate™ task interpreter
    'we could have also used a literal path

    Action.PlaySound("%AMDIR%\2010_09.wav", "1")
End Sub
```

Focus Method

Action Example

Encapsulates the AutoMate™ action in the ‘‘Window’’ group, ‘‘Focus.’’ Brings a window in front of all the other windows on the system based on the window’s title caption.

Declaration

Procedure Focus(WindowTitle: OleVariant); safecall;

Description

Specify the title of the window to focus in the Window Parameter. This call does not support substring matches. To perform substring matches, use Action.FocusEx.

NOTE: To focus a window based on the handle of a launched EXE file, use the BASIC language command *AppActivate* in conjunction with the *Shell* command.

Focus Example

This example will focus the Notepad window using the AutoMate™ Focus Action.

```
Sub Main
    'This script will focus the Notepad window
    'using the AutoMate™ Focus Action
    Action.Focus("Untitled - Notepad")
End Sub
```

FocusEx Method

Action Example

Encapsulates the AutoMate™ action in the “Window” group, “Focus.” Same as Action.Focus, except contains support for partial title matches. Brings a window in front of all the other windows on the system based on the window’s title caption.

Declaration

Procedure FocusEx(Window_: OleVariant; ExactMatch: OleVariant; HandlePassed: OleVariant); safecall;

Description

Specify the title of the window to focus in the Window Parameter. Set “ExactMatch” to “0” for a substring search or “1” for an exact search.

NOTE: To focus a window based on the handle of a launched EXE file, use BASIC language command AppActivate in conjunction with the Shell command.

FocusEx Example

This example will focus a window that has a window title beginning with the text “Note”.

```
Sub Main
    'This script will focus the Notepad window
    'using the AutoMate™ Focus Action, note that
    'only a partial title was specified
    Action.FocusEx("Note", "0", "")
End Sub
```

Minimize Method

Action Example

Encapsulates the AutoMate™ action in the ‘‘Window’’ group, ‘‘Minimize.’’ Minimizes a window so that it is not displayed anywhere except the task tray.

Declaration

Procedure Minimize(Window_: OleVariant; ExactMatch: OleVariant; HandlePassed: OleVariant); safecall;

Description

Specify the title of the window to minimize in the Window Parameter. Set ‘‘ExactMatch’’ to ‘‘0’’ for a substring search or ‘‘1’’ for an exact search.

Minimize Example

This example will minimize a window with a window title beginning with the text “Note”.

```
Sub Main
    'This script will minimize the Notepad window
    'using the AutoMate™ Focus Action, note that
    'only a partial title was specified
    Action.Minimize("Note", "0", "")
End Sub
```

Maximize Method

Action Example

Encapsulates the AutoMate™ action in the “Window” group, “Maximize.” Maximizes a window so that it occupies the full screen.

Declaration

Procedure Maximize(Window_: OleVariant; ExactMatch: OleVariant; HandlePassed: OleVariant); safecall;

Description

Specify the title of the window to maximize in the Window Parameter. Set “ExactMatch” to “0” for a substring search or “1” for an exact search.

Maximize Example

This example will maximize a window with a window title beginning with the text “Note.”

```
Sub Main
    'This script will maximize the Notepad window
    'using the AutoMate™ Maximize Action, note that
    'only a partial title was specified
    Action.Maximize("Note", "0", "")
End Sub
```

Restore Method

Action Example

Encapsulates the AutoMate™ action in the “Window” group, “Restore.” Restores a window to its normal state so that it is sizable. When a window is restored, it is neither minimized nor maximized.

Declaration

Procedure Restore(Window_: OleVariant; ExactMatch: OleVariant; HandlePassed: OleVariant); safecall;

Description

Specify the title of the window to minimize in the Window Parameter. Set “ExactMatch” to “0” for a substring search or “1” for an exact search.

Restore Example

This example will restore a window with a window title beginning with the text “Note.”

```
Sub Main
    'This script will restore the Notepad window
    'using the AutoMate™ Restore Action, note that
    'only a partial title was specified
    Action.Restore("Note", "0", "")
End Sub
```

CloseWind Method

Action Example

Encapsulates the AutoMate™ action in the “Window” group, “Close.” Sends a close message to a window. This will usually result in the closing of the window, and in some instances, shutting down the application.

Declaration

Procedure CloseWind(Window_: OleVariant; ExactMatch: OleVariant; HandlePassed: OleVariant); safecall;

Description

Specify the title of the window to minimize in the Window Parameter. Set “ExactMatch” to “0” for a substring search or “1” for an exact search.

You can attempt to close an application by trying to close an application’s main window.

CloseWind Example

This example will close a window with a window title beginning with the word “Note.”

```
Sub Main
    'This script will close the Notepad window
    'using the AutoMate™ CloseWind Action, note that
    'only a partial title was specified
    Action.CloseWind("Note", "0", "")
End Sub
```

Hide Method

Action Example

Encapsulates the AutoMate™ action in the “Window” group, “Hide.” Hidden windows can be un-hidden by using the Action.UnHide method.

Declaration

Procedure Hide(Window_: OleVariant; ExactMatch: OleVariant; HandlePassed: OleVariant); safecall;

Description

Specify the title of the window to minimize in the Window Parameter. Set “ExactMatch” to “0” for a substring search or “1” for an exact search.

Hide Example

This example will hide a window with a window title beginning with the word “Note.”

```
Sub Main
    'This script will hide the Notepad window
    'using the AutoMate™ Hide Action, note that
    'only a partial title was specified
    Action.Hide("Note", "0", "")
End Sub
```

UnHide Method

Action Example

Encapsulates the AutoMate™ action in the ‘‘Window’’ group, ‘‘UnHide.’’ Makes a window previously hidden with the Action.Hide method visible again.

Declaration

Procedure Unhide(Window_: OleVariant; ExactMatch: OleVariant; HandlePassed: OleVariant); safecall;

Description

Specify the title of the window to minimize in the Window Parameter. Set ‘‘ExactMatch’’ to ‘‘0’’ for a substring search or ‘‘1’’ for an exact search.

UnHide Example

This example will unhide a window with a window title beginning with the word “Note.”

```
Sub Main
    'This script will unhide the Notepad window
    'using the AutoMate™ UnHide Action, note that
    'only a partial title was specified
    Action.UnHide("Note", "0", "")
End Sub
```

FindText Method

Action Example

Encapsulates the AutoMate™ action in the “Window” group, “Focus Window Containing.” Same as Action.FocusEx, except contains support for checking the text contents inside a window. If the text is found, the method brings the window in front of all the other windows on the system.

Declaration

procedure FindText(Window_: OleVariant; ExactMatch: OleVariant; HandlePassed: OleVariant; SearchText: OleVariant; SearchExact: OleVariant); safecall;

Description

Specify the title of the window to focus in the Window Parameter, the text to search for inside the dialog box in the SearchText parameter, and whether it should match exactly in the SearchExact parameter.

NOTE: To focus a window based on the handle of a launched EXE file, use BASIC language command AppActivate in conjunction with the Shell command.

FindText Example

This example will attempt to locate a window with a title with the word “note” in it, and containing the text “Do you want to save the changes?” If such a window exists, it is focused.

```
Sub Main
   Action.FindText("note", "0", "", "Do you want to save the changes?", "0")
End Sub
```

DDE Method

Action Example

Encapsulates the AutoMate™ action ‘‘DDE Command.’’ Performs a DDE request to another application.

Declaration

Procedure DDE(Service: OleVariant; Topic: OleVariant; Item: OleVariant); safecall;

Description

Use of this method is not recommended, instead use the BASIC language DDE commands such as DDEExecute and DDERequest.

DDE Example

The following example illustrates how to use AutoMate's DDE Action to send a DDE Command to the Windows Program Manager.

```
Sub Main
    Action.DDE("PROGMAN", "PROGMAN","[CreateGroup(XXX)]")
End Sub
```

The following example illustrates an alternate way to perform the above example using AutoMate™ Basic native commands.

```
Sub Main      ChanNum = DDEInitiate("PROGMAN","PROGMAN")      DDEExecute
ChanNum,"[CreateGroup(XXX)]"      DDETerminate ChanNumEnd Sub
```

Start Method

Action Example

Encapsulates the AutoMate™ action ‘‘Start an application.’’ Starts an application and sets associated window flags.

Declaration

```
procedure Start(FileName: OleVariant; Parameters: OleVariant; ShowFlag: OleVariant;
                DefaultDir: OleVariant); safecall;
```

Description

Starts an application using the AutoMate™ task interpreter engine. Use of this method is not recommended, instead use the BASIC language commands such as Shell() or the ShellExecute Windows API call.

Start Example

This example will attempt to open notepad and send a series of keystrokes to it.

```
Sub Main
    'This script will start Notepad and then type some text and then access the file menu
    Action.Start("Notepad","","0","")
    Wait 1
    Action.SendKeys("1","100","These are the keys to be sent~")
    Wait 1
    Action.SendKeys("1","500","You can also use AutoMate™ keystroke commands like ALT{+}F~")
    Wait 1
    Action.SendKeys("1","500","Watch as AutoMate™ accesses the File Menu and chooses save")
    Wait 2
    Action.SendKeys("1","500","%fs")
End Sub
```

OSCommand Method

Action Example

Encapsulates the AutoMate™ actions under the “System” group. Performs a system reboot, shutdown or logout. OSCommand can also be used to programmatically shutdown the AutoMate™ Task Service.

Declaration
procedure OSCommand(Command: OleVariant); safecall;

Description
Set Command to one of the following integer values to specify which system command to perform:

Value	Description
REBOOT	Reboots the system
SHUTDOWN	Shuts the system down
LOGOUT	Logs-out of the system
QUIT	Shuts down AutoMate™

OSCommand Example

This example will shut down the AutoMate™ Task Service.

```
Sub Main
   ‘Shuts the AutoMate™ task service down right now
        Action.OSCommand(“4”)

End Sub
```

SendKeys Method

Action Example

Encapsulates the AutoMate™ action ‘‘Send Keystrokes’’

Declaration

Procedure SendKeys(PauseEnabled: OleVariant; PauseTime: OleVariant; Keystrokes: OleVariant); safecall;

Description

Sends the keystrokes specified by the keystrokes parameter to the currently active window.

Although the BASIC language contains its own sendkeys command, you may wish to use AutoMate’s native sendkeys engine.

The PauseEnabled parameter controls whether AutoMate™ will respect the settings of the Pause time parameter. If Pause enabled is set to ‘‘0’’, AutoMate™ will execute the keystrokes immediately.

SendKeys Example

This example will open notepad, and send keystrokes to it after a 50 millisecond pause.

```
Sub Main
    Shell "notepad.exe", 1
    'Pause for 50 milliseconds before send
    Action.SendKeys("1", "50", "this is a test of the AutoMate™ send keystrokes engine")
End Sub
```

RunScript Method

Action Example

Encapsulates the AutoMate™ action ‘‘Run a Basic Script.’’

Declaration

procedure RunScript(FileName: OleVariant); safecall;

Description

Use of this encapsulated action is not advised as developers can easily include and call other BASIC subroutines using the BASIC commands *Call* or *MacroRun.*

RunScript Example

The following example will run an AutoMate™ BASIC Script.

```
Sub Main
    Action.RunScript(“c:\mytest.bas”)
End Sub
```

OpenFile Method

Action Example

Encapsulates the AutoMate™ action ‘‘Open a document.’’ Opens any documents type that is registered on the system (i.e. DOC file, TXT file, XLS, or Web Site address).

Declaration

procedure OpenFile(FileName: OleVariant); safecall;

Description

Allows opening of a registered document type by specifying the path and filename in the Filename parameter.

OpenFile Example

The following example will open the file “readme.txt” in the AutoMate™ directory using the viewer for text files (Windows sets this to Notepad.exe by default).

```
Sub Main

    'This script uses the AutoMate™ system constant %AMDIR%
    'but we could have used a literal path such as C:\test.txt

    Action.OpenFile("%AMDIR%\readme.txt")

End Sub
```

PrintFile Method

Action Example

Encapsulates the AutoMate™ action ‘‘Print a document.’’ Prints any registered document type using the application that is registered to process this type of request.

Declaration

procedure PrintFile(FileName: OleVariant); safecall;

Description

Allows printing of a registered document type by specifying the path and filename in the Filename parameter.

Note: Not all applications support PRINT, so this method may not work with your document type.

PrintFile Example

The following example prints the file “readme.txt” using the default editor for text files (usually Notepad).

```
Sub Main

    'This script uses the AutoMate™ system constant %AMDIR%
    'but we could have used a literal path such as C:\test.txt

    Action.PrintFile("%AMDIR%\readme.txt")

End Sub
```

MoveFile Method

Action Example

Encapsulates the AutoMate™ action under the “File” group, “Move file(s).”

Declaration

procedure MoveFile(Source: OleVariant; Destination: OleVariant); safecall;

Description

The action copies one or more files from the specified Source to the specified Destination. MoveFile supports wildcards. After copying, the source file(s) are deleted.

MoveFile Example

The following example moves the file “test1.txt” from the c: drive to the d: drive as “test2.txt”

```
Sub Main

    'you must have a file called c:\test1.txt created to use this
    Action.MoveFile("c:\test1.txt", "c:\test2.txt")

End Sub
```

This example moves all the files in the c:\onetemp directory to the c:\nexttemp directory.

```
Sub Main

    'you must have directories with files in them called c:\onetemp and c:\nexttemp
    Action.MoveFile("c:\onetemp\*.*", "c:\nexttemp\*.*")

End Sub
```

RenameFile Method

Action Example

Encapsulates the AutoMate™ action under the “File” group, “Rename files.”

Declaration

```
procedure RenameFile(OldName: OleVariant; NewName: OleVariant); safecall;
```

Description

The action renames the file specified in the Source parameter to the name specified in Destination. Rename does not currently support wildcards.

RenameFile Example

The following example renames the file “c:\test1.txt” to “test2.txt”

```
Sub Main

    'you must have a file called c:\test1.txt created to use this
          Action.RenameFile("c:\test1.txt", "c:\test2.txt")
End Sub
```

CopyFile Method

Action Example

Encapsulates the AutoMate™ action under the “File” group, “Copy file(s).”

Declaration

procedure CopyFile(Source: OleVariant; Destination: OleVariant); safecall;

Description

The action copies one or more files from the specified Source to the specified Destination. CopyFile supports wildcards.

CopyFile Example

The following example copies the file named “test1.txt” to “test2.txt.”

```
Sub Main

    'you must have a file called c:\test1.txt created to use this
    Action.CopyFile("c:\test1.txt", "c:\test2.txt")

End Sub
```

This example will copy all the files from the “onetemp” directory into the “nexttemp” directory. Note that the “nexttemp” directory must exist for the action to work correctly.

```
Sub Main

    'you must have directories with files in them called c:\onetemp and c:\nexttemp
    Action.CopyFile ("c:\onetemp\*.*", "c:\nexttemp\*.*")

End Sub
```

DeleteFile Method

Action Example

Encapsulates the AutoMate™ action under the “File” group, “Delete file(s).”

Declaration
procedure DeleteFile(FileName: OleVariant); safecall;

Description
The action deletes the filename specified in the Filename parameter. DeleteFile supports wildcards.

DeleteFile Example

The following example deletes the file named “test1.txt” from the c: drive.

```
Sub Main

    'you must have a file called c:\test1 with any extension created to use this
    Action.DeleteFile ("c:\test1.txt")

End Sub
```

This example will delete all the files named “test1”, regardless of their extension.

```
Sub Main

    'you must have a file called c:\test1 with any extension created to use this
    Action.DeleteFile ("c:\test\test1.*")

End Sub
```

MakeDir Method

Action Example

Encapsulates the AutoMate™ action under the ‘‘File’’ group, ‘‘Create Directory.’’

Declaration
procedure MakeDir(Directory: OleVariant); safecall;

Description
The action creates a folder on the file system named by the value specified in the Directory parameter.

MakeDir Example

The following example makes a directory called “tester” on the c: drive.

```
Sub Main

    'you must have a file called c:\test1 with any extension created to use this
    Action. MakeDir ("c:\tester\")

End Sub
```

RemDir Method

Action Example

Encapsulates the AutoMate™ action under the ‘‘File’’ group, ‘‘Remove Directory.’’

Declaration
procedure RemDir(Directory: OleVariant); safecall;

Description
The action deletes a folder on the file system named by the value specified in the Directory parameter.

RemDir Example

The following example removes the ‘‘tester’’ directory from the c: drive. Note that the directory ‘‘tester’’ must exist before this task is run to work properly.

```
Sub Main

    'you must have a file called c:\test1 with any extension created to use this
    Action. RemDir ("c:\tester\")

End Sub
```

ChDir Method

Action Example

Encapsulates the AutoMate™ action under the ‘‘File’’ group, ‘‘Change Directory.’’

Declaration

procedure ChDir(Directory: OleVariant); safecall;

Description

The action changes the current directory of the system to the folder named by the value specified in the Directory parameter.

ChDir Example

The following example changes the current AutoMate™ directory to the c:\windows directory.

```
Sub Main
    Action. ChDir ("c:\windows\")
End Sub
```

FTPDownload Method

Action Example

Encapsulates the AutoMate™ action under the “Internet | FTP” group, “Quick Retrieve.”

Declaration

procedure FTPDownload(SourceFile: OleVariant; DestinationFile: OleVariant; TransferType: OleVariant; CheckType: OleVariant; CheckTime: OleVariant); safecall;

Description

Specifies file(s) to download from an FTP Server. This command performs a three-step operation of logging into the server, downloading files, and logging out of the server.

FTPDownload Example

The following example downloads a file named “test.txt” from the FTP server onto the local c: drive as “test.txt.”

```
Sub Main
    'This script will download one file called test.txt.
    'Note that the full path to the file may have to be included in both source
    'and destination depending on your server
    Action.FTPDownload("your.ftp.server","username","password","21","/test.txt","c:\test.txt","","0")
End Sub
```

SendMail Method

Action Example

Encapsulates the AutoMate™ action under the “Internet | E-Mail” group, “Send Message.”

Declaration

procedure SendMail(Server: OleVariant; Sender: OleVariant; Recipient: OleVariant; CCRecipient: OleVariant; Subject: OleVariant; FileName: OleVariant); safecall;

Description

Sends a MIME encoded message using the Internet standard SMTP protocol. The body of the message to be sent should be located in a text file – the path to the text file should be specified in the parameter FileName.

NOTE: SMTP user authentication is not currently supported.

NOTE: File attachments are not currently supported.

SendMail Example

The following example sends the contents of the text file “c:\message.txt” to the recipient via your.smtp.server.

```
Sub Main

Action.SendMail("your.smtp.server","youraddress@domain.com","recipientsaddress@domain.com","","
this is a test","c:\message.txt")

End Sub
```

FTPUpload Method

Action Example

Encapsulates the AutoMate™ action under the “Internet | FTP” group, “Quick Send.”

Declaration

procedure SendMail(Server: OleVariant; Sender: OleVariant; Recipient: OleVariant; CCRecipient: OleVariant; Subject: OleVariant; FileName: OleVariant); safecall;

Description

Specifies file(s) to upload to an FTP Server. This command performs a three-step operation of logging into the server, downloading files, and logging out of the server.

FTPUpload Example

The following example uploads a file to the FTP server.

```
Sub Main
    'This script will upload one file called test.txt.
    'Note that the full path to the file may have to be included in both source
    'and destination depending on your server
    Action.FTPUpload("your.ftp.server","username","password","21","c:\test.txt","/test.txt","","0")
End Sub
```

TaskName Example

The following example displays the name of the currently executing task in a message box.

```
Sub Main
    Dim RunningTask As Variant

    RunningTask = CurrentTask.TaskName
    MsgBox RunningTask
End Sub
```

LaunchDate Example

The following example sets the launch date of the current task to September 14, 1999. Note that, for this example, we are assuming the Regional Settings to be “English (United States).”

```
Sub Main
    CurrentTask.LaunchDate = “09/14/1999”
End Sub
```

LaunchTime Example

The following example sets the launch time of the current task to 11:56 AM. Note that, for this example, we are assuming the Regional Settings to be ‘‘English (United States).’’

```
Sub Main
    CurrentTask.LaunchTime = ‘‘11:56 AM’’
End Sub
```

Frequency Example

The following example will set the current task to run only when manually launched.

```
Sub Main
    CurrentTask.Frequency = 2
End Sub
```

This example sets the current task to run itself every 3 days starting on September 15, 1999 at 11:29AM. When the task runs, it will reschedule itself relative to the time it was originally set to launch. If the task is late, it will not run at all.

```
Sub Main
    ‘ Set the task to “every”
    CurrentTask.Frequency = 1

    ‘ Start launching on September 15, 1999
    CurrentTask.LaunchDate = “09/15/1999”

    ‘ At 11:59 AM
    CurrentTask.LaunchTime = “11:29 AM”

    ‘ Set the interval type to “days”
    CurrentTask.IntervalType = 2

    ‘ Set the interval to 3, for every three days
    CurrentTask.Interval = 3

    ‘ If we are late, don’t run at all
    CurrentTask.RunLate = 1

    ‘ Reschedule the task relative to the launch date/time if we are late
    CurrentTask.ScheduleLate = 0
End Sub
```

Setting Task Properties

This example sets the current task to run itself every 3 days starting on September 15, 1999 at 11:29AM. When the task runs, it will reschedule itself relative to the time it was originally set to launch. If the task is late, it will not run at all.

```
Sub Main
    ‘ Set the task to trigger by schedule and hotkey
    CurrentTask.TaskTriggers = 9

    ‘ Set the task to “every”
    CurrentTask.Frequency = 1

    ‘ Start launching on September 15, 1999
    CurrentTask.LaunchDate = “09/15/1999”

    ‘ At 11:59 AM
    CurrentTask.LaunchTime = “11:29 AM”

    ‘ Set the interval type to “days”
    CurrentTask.IntervalType = 2

    ‘ Set the interval to 3, for every three days
    CurrentTask.Interval = 3

    ‘ If we are late, don’t run at all
    CurrentTask.RunLate = 1

    ‘ Reschedule the task relative to the launch date/time if we are late
    CurrentTask.ScheduleLate = 0

    ‘ Set the hotkey trigger to Ctrl-Shift H
    CurrentTask.Hotkey = “%&H”
End Sub
```

Active Example

The following example sets the task to inactive. This will prevent the task from executing the rest of the task.

```
Sub Main
    CurrentTask.Active = 0
End Sub
```

WindowName Example

The following example sets the current task to trigger only when the window with the title of “Untitled – Notepad” appears.

```
Sub Main
    CurrentTask.Triggers = 4
    CurrentTask.WindowName = “Untitled – Notepad”
End Sub
```

StepCount Example

The following example will display the current number of steps in a message box.

```
Sub Main
    MsgBox CurrentTask.StepCount
End Sub
```

Details Example

The following example sets the task's details to "This task is wonderful."

```
Sub Main
    CurrentTask.Details = "This task is wonderful."
End Sub
```

FrequencyString Example

The following example displays the current task's scheduling parameters in a friendly, sentence form.

```
Sub Main
    MsgBox CurrentTask.FrequencyString
End Sub
```

Events Example

The following example sets the current task to trigger only when the spooler changes (i.e. when a print job is added, removed or completed from the printer queue) or the system date/time changes

```
Sub Main
    ‘ Set the task to trigger on Windows Events
    CurrentTask.Triggers = 2

    ‘ Set to trigger on spooler or time change
    CurrentTask.Events = 33
End Sub
```

GetStepCommand Example

The following example displays the step command of step #2 in a message box.

```
Sub Main
    MsgBox CurrentTask.GetStepCommand (1)
End Sub
```

GetStepParamCount Example

The following example displays the total number of parameters of step #2 in a message box.

```
Sub Main
    MsgBox CurrentTask.GetStepParamCount (2)
End Sub
```

ClearSteps Example

The following example clears the current task of all its steps.

```
Sub Main
    CurrentTask.ClearSteps
End Sub
```

PlayRecording Method

Action Example

Encapsulates the AutoMate™ action ‘‘Record Events’’ under the ‘‘Recorder’’ branch.

Declaration

procedure PlayRecording (FileName: OleVariant); safecall;

Description

Plays the mouse movements recorded in the file specified by Filename. Use the Step Builder to record a series of mouse movements to a file.

PlayRecording Example

The following example plays back the recorded mouse movements from the file “mousey.dat.”

```
Sub Main
    CurrentTask.PlayRecording (“c:\mousey.dat”)
End Sub
```

Action2

Properties Methods

The Action2 object encapsulates the actions available in AutoMate™. The Action2 object contains updated methods of the Action object, as well as extended methods and actions made available since AutoMate™ 4.06d.

Description

The CurrentTask object contains methods that invoke AutoMate™ actions through the AutoMate™ Task Service. Use this object to execute any of the actions that AutoMate™ supports through the AutoMate™ Step Builder.

When a script is started, AutoMate™ automatically instantiates an Action2 object as ‘‘Action2’’.

The methods of Action2 take the parameters of the method and wrap them into an AutoMate™ compatible command. This command is then marshaled to the AutoMate™ Configuration Manager to be run in the context of the same task that started the script. Every method returns an integer value as a result of the step’s execution in AutoMate™:

Return Value	Meaning
0	The action failed. Use the LastErrorMessage Property to get the error message.
1	The action completed successfully.
2	The action told AutoMate™ to stop the task.

Because every method in the Action2 uses the same return codes, you can easily create more elaborate and complex tasks using AutoMate™ steps.

Action2 Methods

Action2 Legend Alphabetical Listing

General

- StartApplication
- OpenDocument
- SendKeystrokes
- PasteKeys
- PrintDocument
- Message
- Reminder
- PlaySound
- StopSound
- DDECommand

Window

- WindowFocus
- WindowMaximize
- WindowMinimize
- WindowRestore
- FocusWindowContaining
- WindowClose
- WindowHide
- WindowUnhide
- GetFocusedWindowName

Internet

- DialUp
- OpenWebpage
- PingMachine

FTP

- FTPQuickSend
- FTPQuickRetrieve
- FTPLogin
- FTPRename
- FTPUpload
- FTPDownload
- FTPDelete
- FTPMakeDirectory
- FTPRemoveDirectory
- FTPLogout
- FTPChangeDirectory

E-Mail

- CheckForMail

- SendEmail

Mouse

- MoveMouse
- LeftClick
- LeftDblClick
- LeftMouseDown
- LeftMouseUp
- RightClick
- RightDblClick
- RightMouseDown
- RightMouseUp
- MiddleClick
- MiddleDblClick
- MiddleMouseDown
- MiddleMouseUp

File System

- AMCopyFile
- AMMoveFile
- AMRenameFile
- AMDeleteFile
- AMMakeDir
- AMRemoveDir
- AMChangeDir

Zip

- Zip
- Unzip

Clipboard

- ClipboardCut
- ClipboardCopy
- ClipboardPaste
- ClipboardClear

System

- Login
- RebootMachine
- ShutdownMachine
- QuitAutoMate™
- LogoffMachine
- UpdateAutoMate™

Control Panel

- **Services**
  - StartService

- StopService
- PauseService
- ContService
- InstallService
- RemoveService

Flow Control

- Wait
- PromptUser
- WaitForWindow
- WaitForWindowToDisappear
- StartTask
- DisableThisTask

Security

- LockKeyboard
- LockMouse
- UnlockKeyboard
- UnlockMouse
- Password

Recorder

- PlayMouse

BASIC Scripting

- RunScript

Others

- SendRawStep
- GetFocusedWindowName

WindowFocus Method

Action2 Example

Encapsulates the AutoMate™ action in the “Window” group, “Focus.” Finds the window with the title text indicated, focuses the window, and attempts to bring it to the foreground.

Declaration

function WindowFocus(varWindowTitle As Variant,
    intExactMatch As Integer) As Integer

Parameters

varWindowTitle

The text of the window title to be focused.

intExactMatch

Whether or not the title bar text should match *varWindowTitle* exactly, or partially. Use “0” for a partial match or “1” for an exact match.

Description

The WindowFocus method will attempt to locate a window with the title specified in the *varWindowTitle* parameter. If a window exists, AutoMate™ will attempt to focus the window and bring it to the foreground. If a window cannot be found, the method returns “1”, indicating a failed step.

If multiple windows are found matching *varWindowTitle*, AutoMate™ will attempt to focus the first one it eternally enumerates.

WindowMinimize Method

Action2 Example

Encapsulates the AutoMate™ action in the “Window” group, “Minimize.” Finds the window with the title text indicated and minimizes the window.

Declaration

function WindowMinimize(varWindowTitle As Variant,
intExactMatch As Integer) As Integer

Parameters

varWindowTitle

The text of the window title to be minimized.

intExactMatch

Whether or not the title bar text should match *varWindowTitle* exactly or partially. Use “0” for a partial match or “1” for an exact match.

Description

The WindowMinimize method will attempt to locate a window with the title specified in the *varWindowTitle* parameter. If a window exists, AutoMate™ will attempt to minimize the window as if a user clicked the “minimize” button from the titlebar pane of the window. If a window cannot be found, the method returns “1”, indicating a failed step.

If multiple windows are found matching *varWindowTitle*, AutoMate™ will attempt to minimize the first window it eternally enumerates.

WindowMaximize Method

Action2 Example

Encapsulates the AutoMate™ action in the “Window” group, “Maximize.” Finds the window with the title text indicated and maximizes the window.

Declaration

function WindowMaximize(varWindowTitle As Variant,
intExactMatch As Integer) As Integer

Parameters

varWindowTitle

The text of the window title to be maximized.

intExactMatch

Whether or not the title bar text should match *varWindowTitle* exactly or partially. Use “0” for a partial match or “1” for an exact match.

Description

The WindowMaximize method will attempt to locate a window with the title specified in the *varWindowTitle* parameter. If a window exists, AutoMate™ will attempt to maximize the window as if a user clicked the “maximize” button from the titlebar pane of the window. If a window cannot be found, the method returns “1”, indicating a failed step.

If multiple windows are found matching *varWindowTitle*, AutoMate™ will attempt to minimize the first window it eternally enumerates.

WindowHide Method

Action2 Example

Encapsulates the AutoMate™ action in the “Window” group, “Hide.” Finds the window with the title text indicated and hides the window.

Declaration

function WindowHide(varWindowTitle As Variant,
intExactMatch As Integer) As Integer

Parameters

varWindowTitle

The text of the window title to be hidden.

intExactMatch

Whether or not the title bar text should match *varWindowTitle* exactly or partially. Use “0” for a partial match or “1” for an exact match.

Description

The WindowHide method will attempt to locate a window with the title specified in the *varWindowTitle* parameter. If a window exists, AutoMate™ will attempt to hide the window. The window will not appear in the taskbar if hidden. If a window cannot be found, the method returns “1”, indicating a failed step.

If multiple windows are found matching *varWindowTitle*, AutoMate™ will attempt to hide the first window it eternally enumerates.

WindowUnhide Method

Action2 Example

Encapsulates the AutoMate™ action in the “Window” group, “Unhide.” Finds the window with the title text indicated and unhides the window.

Declaration

function WindowUnhide(varWindowTitle As Variant,
intExactMatch As Integer) As Integer

Parameters

varWindowTitle

The text of the window title to be revealed.

intExactMatch

Whether or not the title bar text should match *varWindowTitle* exactly or partially. Use “0” for a partial match “1” for an exact match.

Description

The WindowUnhide method will attempt to locate a *hidden* window with the title specified in the *varWindowTitle* parameter. If a window exists, AutoMate™ will attempt to unhide the window. If a window cannot be found, the method returns “1”, indicating a failed step.

If multiple windows are found matching *varWindowTitle*, AutoMate™ will attempt to unhide the first window it eternally enumerates.

WindowRestore Method

Action2 Example

Encapsulates the AutoMate™ action in the “Window” group, “Restore.” Finds the window with the title text indicated and restores the window.

Declaration

function WindowRestore(varWindowTitle As Variant,
intExactMatch As Integer) As Integer

Parameters

varWindowTitle

The text of the window title to restore.

intExactMatch

Whether or not the title bar text should match *varWindowTitle* exactly or partially. Use “0” for a partial match or “1” for an exact match.

Description

The WindowRestore method will attempt to locate a window with the title specified in the *varWindowTitle* parameter. If a window exists, AutoMate™ will attempt to restore the window to its default show state. If a window cannot be found, the method returns “1”, indicating a failed step.

If multiple windows are found matching *varWindowTitle*, AutoMate™ will attempt to restore the first window it eternally enumerates.

WindowClose Method

Action2 Example

Encapsulates the AutoMate™ action in the “Window” group, “Close.” Finds the window with the title text indicated and closes the window.

Declaration

function WindowClose(varWindowTitle As Variant,
intExactMatch As Integer) As Integer

Parameters

varWindowTitle

The text of the window title to be hidden.

intExactMatch

Whether or not the title bar text should match *varWindowTitle* exactly or partially. Use “0” for a partial match or “1” for an exact match.

Description

The WindowClose method will attempt to locate a window with the title specified in the *varWindowTitle* parameter. If a window exists, AutoMate™ will attempt to close the window. The window will not appear in the taskbar if hidden. If a window cannot be found, the method returns “1”, indicating a failed step.

If multiple windows are found matching *varWindowTitle*, AutoMate™ will attempt to close the first window it eternally enumerates.

NOTE: Most applications will terminate once their main window is closed. Therefore, try using the WindowClose command to terminate applications through the AutoMate™ scripting language.

Message Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘General’’ group, ‘‘Message.’’ The method displays a standard message box with the desired text, and optionally waits for a response before continuing.

Declaration

function Message(varMessageText As Variant,
intModal As Integer) As Integer

Parameters

varMessageText

The text to be displayed inside the message box.

intModal

Whether the message box is displayed modally or not. If the message box is modal, AutoMate™ will wait until the user clicks the OK button before the function returns. Set *intModal* to ‘‘1’’ for a modal message box or ‘‘0’’ otherwise.

Description

The Message method will display a standard Windows message box with an OK button, and *varMessageText* as the text within the message box. Use Message to display a message to a user while the task is running.

StartApplication Method

Action2 Example

Encapsulates the AutoMate™ action in the "General" group, "Start an application." The method attempts to start the given application using the specified commandline.

Declaration

function StartApplication (varCommandLine As Variant,
varParameters As Variant,
varDefaultDir As Variant,
intWindowState As Integer,
intWaitUntilReady As Integer,
intWaitUntilDone As Integer) As Integer

Parameters

varCommandLine

The commandline AutoMate™ should use to start the application (e.g. C:\Program Files\Unisyn\AutoMate4\Settings.exe). Note that the commandline is not case sensitive.

varParameters

Any additional parameters that may be needed to run the program.

varDefaultDir

Path to the directory where AutoMate™ should start the program.

intWindowState

An integer specifying how AutoMate™ should attempt to display the window:

Value	Meaning
0	Normal
1	Maximized
2	Minimized
3	Hidden

intWaitUntilReady

Specifies whether or not AutoMate™ should wait until the application is finished its startup procedures and signals it is ready to accept input before continuing. Set to "1" to have AutoMate™ wait or "0" otherwise.

intWaitUntilDone

Specifies whether or not AutoMate™ should wait until the started application closes before returning. Set to "1" to make the method wait until the application has completed before returning or "0" otherwise.

Description

The StartApplication method attempts to start a standard Windows or DOS-based program at the path specified by *varCommandLine*. AutoMate™ can be told to wait until the application is ready for input or

wait until the application is done, before continuing with the next line in the script. Setting both *intWaitUntilReady* and *intWaitUntilDone* to ‘‘1’’ is undefined.

If the program pointed to by *varCommandLine* cannot be started for whatever reason, the method will return ‘‘1.’’ Use the GetLastError property for more information about why the method failed.

PrintDocument Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘General’’ group, ‘‘Print a document.’’ The method attempts to print the specified document using the document’s registered data and program type.

Declaration

function PrintDocument (varDocument As Variant) As Integer

Parameters

varDocument

The full path to the document to be printed.

Description

The PrintDocument method attempts to print out the document at the specified path using the file’s associated file type. For example, pointing to a file with a .txt extension will print the document using Notepad.

OpenDocument Method

Action2 Example

Encapsulates the AutoMate™ action in the “General” group, “Open a document.” The method attempts to open the specified document using the document’s registered data and program type.

Declaration

function OpenDocument (varDocument As Variant) As Integer

Parameters

varDocument

The full path to the document to be opened.

Description

The OpenDocument method attempts to open the document at the specified path using the file’s associated file type. For example, pointing to a file with a .txt extension will open the document using Notepad, and opening a website address will use Internet Explorer (on systems where IE is the default browser).

If the function fails, check to ensure the file you are pointing to has an associated file type.

SendKeystrokes Method

Action2 Example

Encapsulates the AutoMate™ action in the "General" group, "Send keystrokes." The method sends keystrokes to the currently focused window as if a user where typing them on the keyboard.

Declaration

function SendKeystrokes (varKeystrokes As Variant,
    intPauseFirst As Integer,
    intPauseLength As Integer) As Integer

Parameters

varKeystrokes

The keystrokes to send.

intPauseFirst

Specifies whether or not AutoMate™ should pause before sending the keystrokes. If set to "1", AutoMate™ will pause for the number of milliseconds specified by *PauseLength*. If set to "0", the *PauseLength* parameter is ignored.

intPauseLength

The number of milliseconds to wait before AutoMate™ attempts to send the keystrokes specified by *varKeystrokes*. This parameter is used only if *intPauseFirst* is "1."

Description

The SendKeystrokes method will send keys to the currently focused application or window, mimicking a user at the keyboard. Use *intPauseFirst* and *intPauseLength* to specify if AutoMate™ should pause before attempting to send the keystrokes and for how long.

AutoMate™ does not do any checks before sending the keystrokes – it will blindly write to the keyboard buffer regardless of the current system state. Because of this, ensure that the proper window to receive the keyboard focus is selected (using the WindowFocus method) before using the SendKeystrokes method.

PasteKeys Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘General’’ group, ‘‘Paste Keys.’’ The method first copies the desired key sequence to the system clipboard, then pastes them to the focused window.

Declaration

function PasteKeys (varKeysToPaste As Variant) As Integer

Parameters

varKeysToPaste

The keystrokes (or text) to be pasted to the focused application.

Description

The PasteKeys method first copies the text specified in *varKeysToPaste* to the system clipboard, then copies it to the currently focused window.

PasteKeys is faster than SendKeystrokes because it copies the contents from the system clipboard in one step, whereas SendKeystrokes emulates a user typing each key individually on the keyboard. The disadvantage to this is that PasteKeys does *not* support special keys (such as the arrow keys or HOME key). If you need to send anything other than straight text, use SendKeystrokes instead.

Reminder Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘General’’ group, ‘‘Reminder.’’ The method displays an AutoMate™ Reminder dialog box with the desired text and allows the user to optionally specify a ‘‘sleep’’ time.

Declaration

function Reminder (varReminderText As Variant,
    intReschedule As Integer) As Integer

Parameters

varReminderText

The text to be displayed inside the reminder dialog box when it appears.

intReschedule

The number of minutes to display as the default reschedule time.

Description

The Reminder Method will show an interactive reminder dialog box with the text in *varReminderText* and populate the ‘‘reschedule time’’ with *intReschedule*. The user can then choose Ok, which resets the reminder’s timer to the time the user entered in the dialog and reshows the reminder when the timer expires. The function does not return until the user clicks Cancel from the reminder dialog.

PlaySound Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘General’’ group, ‘‘Play A Sound.’’ The method plays the wav sound specified by the filename.

Declaration

function PlaySound (varSoundToPlay As Variant,
intWaitUntilFinished As Integer) As Integer

Parameters

varSoundToPlay

The full path to the sound to be played.

intWaitUntilFinished

Specifies whether or not the method should return before the sound is finished playing. If *intWaitUntilFinished* is ‘‘1’’, the method returns after the sound is finished playing. Otherwise, the method returns immediately.

Description

The PlaySound method will play the sound pointed to by *varSoundToPlay*. Use the StopSound method to stop a playing sound.

StopSound Method

Action2 Example

Immediately stops a playing sound.

Declaration

function StopSound () As Integer

Parameters

none

Description

The StopSound method stops any and all playing sounds on the system, not just one started by AutoMate™.

OpenWebpage Method

Action2 Example

Encapsulates the AutoMate™ action in the "General" group, "Open webpage." The method opens a webpage in the default browser.

Declaration

function OpenWebpage (varWebpageAddress As Variant) As Integer

Parameters

varWebpageAddress

The URL of the webpage to open.

Description

The OpenWebpage method attempts to open the webpage pointed to by *varWebpageAddress* in the system's default browser.

If the default web browser cannot be opened, the method returns "1." Otherwise, it returns "0." Note that the function will therefore return a success regardless of whether or not the page was successfully opened.

FocusWindowContaining Method

Action2 Example

Encapsulates the AutoMate™ action in the “Window” group, “Focus Window Containing.” The method attempts to find and focus a window that contains the specified text.

Declaration

function FocusWindowContaining(varWindowTitle As Variant,
intExactTitleMatch As Integer,
varTextToFind As Variant,
intExactTextMatch As Integer) As Integer

Parameters

varWindowTitle

The title bar text of the window to search for the specified text. Use “*” here to search all open windows in the system.

intExactTitleMatch

Specifies whether or not the *varWindowTitle* must be matched exactly. Set to “1” if yes or “0” if a substring match is preferred.

varTextToFind

The text within the window specified by *varWindowTitle* to be searched for.

varExactTextMatch

Specified whether or not the text inside the window must match *varTextToFind* exactly. Set to “1” for an exact search or “0” for a substring search.

Description

The FocusWindowContaining method will search the window with the titlebar text specified by *varWindowTitle* for the text specified by *varTextToFind*. If the text is located, AutoMate™ will attempt to focus the window and bring it to the foreground.

To search all the open windows in the system for *varTextToFind*, set *varWindowTitle* to “*”.

The method returns”1” if a window containing the specified text cannot be found and “0” if the window is found and focused successfully.

PingMachine Method

Action2 Example

Encapsulates the AutoMate™ action in the "Internet" group, "Ping Machine." The method attempts to ping the specified host and places the roundtrip time into an AutoMate™ variable.

Declaration

```
function PingMachine (varMachineAddress As Variant,
    intPingTimeout As Integer,
    varVarToSet As Variant,
    intPingAction As Integer,
    varPingTaskToStart As Variant,
    intNoPingAction As Integer,
    varNoPingTaskToStart As Variant) As Integer
```

Parameters

varMachineAddress

The hostname or IP address of the machine to be pinged.

intPingTimeout

The amount of time (in milliseconds) to wait for a ping response before the ping times out.

varVarToSet

The *variable name* AutoMate™ is to put the ping response time.

intPingAction

The course of action AutoMate™ should take if the ping is responded to successfully (i.e. the ping is returned before the timeout expires):

Value	Meaning
0	Start the task specified by varPingTaskToStart
1	Stop the current task (the method returns 2)
2	Continue the current task (the method returns 0)

varPingTaskToStart

The name of the task AutoMate™ should start if the ping is successful. This parameter is ignored unless *intPingAction* is set to "0."

intNoPingAction

The course of action AutoMate™ should take if the ping fails (i.e., there is no response from the destination before the timeout expires)

Value	Meaning
0	Start the task specified by varNoPingTaskToStart
1	Stop the current task (the method returns 2)
2	Continue the current task (the method returns 0)

varNoPingTaskToStart

The name of the task AutoMate™ should start if the ping is unsuccessful. This parameter is ignored unless *intNoPingAction* is set to "0."

Description

The PingMachine method will send an ICMP ping to the machine specified by *varMachineName* and wait for a response for the time indicated by *intPingTimeout*. The method returns different integer values based on the settings of *intPingAction* and *intNoPingAction* and if the machine responds to the ping in time.

FTPQuickSend Method

Action2 Example

Encapsulates the AutoMate™ action in the "FTP" group, "FTP Quick Send." The method attempts to login to an FTP server, upload a file, and disconnect in one step.

Declaration

function FTPQuickSend (varHost As Variant,
varUsername As Variant,
varPassword As Variant,
intPort As Integer,
varLocalFile As Variant,
varRemoteFile As Variant,
varFTPLogFile As Variant,
intOverwriteLog As Integer) As Integer

Parameters

varHost

The hostname or IP address of the FTP server to upload the file to.

varUsername

The username used to log in to the server. For anonymous login, try "anonymous"

varPassword

The clear text password to be used to login to the server. For anonymous logins, try your email address (e.g. user@rockon.com)

intPort

The port number to connect to on the FTP server. Most FTP servers use port 23.

varLocalFile

The full path of the file on the local system to be uploaded to the FTP server.

varRemoteFile

The full path to place the file on the FTP server.

varFTPLogFile

The full path to a logfile on your local system that AutoMate™ can use to write session information to. If the file does not exist, AutoMate™ will create it first.

intOverwriteLog

Specifies whether or not AutoMate™ should overwrite the file specified by *varFTPLogFile* if the file already exists. Set to "1" to overwrite the file or set to "0" if you wish to append logging to the end of the file.

Description

The FTPQuickSend method attempts to logon to an FTP server on the specified port, login using the

specified username and password, upload the desired file, and disconnect from the server, all in one function call.

FTPQuickRetrieve Method

Action2 Example

Encapsulates the AutoMate™ action in the "FTP" group, "FTP Quick Retrieve." The method attempts to login to an FTP server, download a file, and disconnect in one step.

Declaration

function FTPQuickSend (varHost As Variant,
varUsername As Variant,
varPassword As Variant,
intPort As Integer,
varLocalFile As Variant,
varRemoteFile As Variant,
varFTPLogFile As Variant,
intOverwriteLog As Integer) As Integer

Parameters

varHost

The hostname or IP address of the FTP server to download the file from.

varUsername

The username used to log in to the server. For anonymous login, try "anonymous"

varPassword

The clear text password to be used to login to the server. For anonymous logins, try your email address (e.g. user@rockon.com)

intPort

The port number to connect to on the FTP server. Most FTP servers use port 23.

varLocalFile

The full path (including filename) of where to place the downloaded file on the local system.

varRemoteFile

The full path of the file on the FTP server to download.

varFTPLogFile

The full path to a logfile on your local system that AutoMate™ can use to write session information to. If the file does not exist, AutoMate™ will create it first.

intOverwriteLog

Specifies whether or not AutoMate™ should overwrite the file specified by *varFTPLogFile* if the file already exists. Set to "1" to overwrite the file or set to "0" if you wish to append logging to the end of the file.

Description

The FTPQuickRetrieve method attempts to logon to an FTP server on the specified port, login using

the specified username and password, download the desired file to the local machine, and disconnect from the server, all in one function call.

FTPLogin Method

Action2 Example

Encapsulates the AutoMate™ action in the "FTP" group, "FTP Login." The method attempts to connect AutoMate™ to an FTP server.

Declaration

function FTPLogin (varHost As Variant,
varUsername As Variant,
varPassword As Variant,
intPort As Integer,
varFTPLogFile As Variant,
intOverwriteLog As Integer,
intPassiveMode As Integer) As Integer

Parameters

varHost

The hostname or IP address of the FTP server to connect to.

varUsername

The username used to log in to the server. For anonymous login, try "anonymous"

varPassword

The clear text password to be used to login to the server. For anonymous logins, try your email address (e.g. user@rockon.com)

intPort

The port number to connect to on the FTP server. Most FTP servers use port 23.

varFTPLogFile

The full path to a logfile on your local system that AutoMate™ can use to write session information to. If the file does not exist, AutoMate™ will create it first.

intOverwriteLog

Specifies whether or not AutoMate™ should overwrite the file specified by *varFTPLogFile* if the file already exists. Set to "1" to overwrite the file or set to "0" if you wish to append logging to the end of the file.

intPassiveMode

If the local machine or remote machine is behind a firewall, it may be necessary to use the FTP commands in passive mode. To do this, set *intPassiveMode* to "1." Otherwise, set to "0."

Description

The FTPLogin method attempts to connect to the FTP server specified by *varHost* on the port *intPort*, using the username specified by *varUsername* and the password specified by *varPassword*.

Use the FTPLogin method to establish a persistent connection to an FTP server. Once a connection is

made, you can use the other FTP commands to upload, download, rename, and delete files on the FTP server. This differs from the FTPQuickUpload and FTPQuickDownload methods, which connect to a server, perform one action, and then disconnect.

When finished with the FTP server, call the FTPLogout method to cleanly disconnect from the server.

NOTE: The Action2 FTP methods (with the exception of FTPQuickSend and FTPQuickRetrieve) may not function properly in the AutoMate™ BASIC IDE. To more accurately use these functions, run the script from within an AutoMate™ task using the ‘‘Run a BASIC Script’’ action.

FTPLogout Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘FTP’’ group, ‘‘FTP Logout.’’ The method disconnects a connection to an FTP server previously established using the FTPLogin method.

Declaration

function FTPLogout () As Integer

Parameters

None

Description

The FTPLogout method attempts to disconnect from an FTP server previously connected to using the FTPLogin method.

Use this command when you are finished with a previously established FTP connection to cleanly disconnect from the server.

NOTE: The Action2 FTP methods (with the exception of FTPQuickSend and FTPQuickRetrieve) may not function properly in the AutoMate™ BASIC IDE. To more accurately use these functions, run the script from within an AutoMate™ task using the ‘‘Run a BASIC Script’’ action.

FTPRename Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘FTP’’ group, ‘‘FTP Rename.’’ The method attempts to rename a file on the FTP server.

Declaration

function FTPRename (varFTPOldName As Variant,
varFTPNewName As Variant,
intIncludeSubDirs As Integer) As Integer

Parameters

varFTPOldName

The full path of the file on the FTP server to be renamed.

varFTPNewName

The new name (including path, if relevant) of the file on the FTP server.

intIncludeSubDirs

If set to ‘‘0’’, the rename command applies to only the current directory. If set to ‘‘1’’, the rename command will go through all subdirectories as well.

Description

The FTPRename method attempts to rename a file on the FTP server specified by *varFTPOldName* to the filename *varFTPNewName*.

A connection to the FTP server must have been previously established by a successful called to FTPLogin.

NOTE: The Action2 FTP methods (with the exception of FTPQuickSend and FTPQuickRetrieve) may not function properly in the AutoMate™ BASIC IDE. To more accurately use these functions, run the script from within an AutoMate™ task using the ‘‘Run a BASIC Script’’ action.

FTPDelete Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘FTP’’ group, ‘‘FTP Delete.’’ The method attempts to delete a file on the FTP server.

Declaration

function FTPDelete (varFTPFilename As Variant,
intIncludeSubDirs As Integer) As Integer

Parameters

varFTPFilename

The full path of the file on the FTP server to be deleted.

intIncludeSubDirs

If set to ‘‘0’’, the delete command applies to only the current directory. If set to ‘‘1’’, the delete command will go through all subdirectories as well.

Description

The FTPDelete method attempts to delete a file on the FTP server specified by *varFTPFilename*.

A connection to the FTP server must have been previously established by a successful called to FTPLogin.

NOTE: The Action2 FTP methods (with the exception of FTPQuickSend and FTPQuickRetrieve) may not function properly in the AutoMate™ BASIC IDE. To more accurately use these functions, run the script from within an AutoMate™ task using the ‘‘Run a BASIC Script’’ action.

FTPRemoveDirectory Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘FTP’’ group, ‘‘FTP Remove Directory.’’ The method attempts to delete a directory on the FTP server.

Declaration

function FTPRemoveDirectory (varDirectory As Variant,
intIncludeSubDirs As Integer) As Integer

Parameters

varDirectory

The full path of the directory on the FTP server to be deleted.

intIncludeSubDirs

If set to ‘‘0’’, the remove directory command applies to only the current directory. If set to ‘‘1’’, the remove directory command will go through all subdirectories as well.

Description

The FTPRemoveDirectory method attempts to delete a directory on the FTP server specified by *varDirectory*.

The directory must be empty for the function to succeed.

A connection to the FTP server must have been previously established by a successful called to FTPLogin.

NOTE: The Action2 FTP methods (with the exception of FTPQuickSend and FTPQuickRetrieve) may not function properly in the AutoMate™ BASIC IDE. To more accurately use these functions, run the script from within an AutoMate™ task using the ‘‘Run a BASIC Script’’ action.

FTPMakeDirectory Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘FTP’’ group, ‘‘FTP Make Directory.’’ The method attempts to create a directory on the FTP server.

Declaration

function FTPMakeDirectory (varDirectory As Variant) As Integer

Parameters

varDirectory

The full path of the directory on the FTP server to be created.

Description

The FTPMakeDirectory method attempts to create a directory on the FTP server specified by *varDirectory*.

A connection to the FTP server must have been previously established by a successful called to FTPLogin.

NOTE: The Action2 FTP methods (with the exception of FTPQuickSend and FTPQuickRetrieve) may not function properly in the AutoMate™ BASIC IDE. To more accurately use these functions, run the script from within an AutoMate™ task using the ‘‘Run a BASIC Script’’ action.

FTPChangeDirectory Method

Action2 Example

Encapsulates the AutoMate™ action in the “FTP” group, “FTP Change Directory.” The method attempts to change the current default directory of AutoMate™ actions performed on the FTP server.

Declaration

function FTPChangeDirectory (varNewDirectory As Variant) As Integer

Parameters

varNewDirectory

The full path of the directory on the FTP server to be used as the default directory.

Description

The FTPChangeDirectory changes the default directory to the directory specified by *varDirectory*. Once this method is successfully executed, all following AutoMate™ FTP commands that work with paths work relative to this directory.

A connection to the FTP server must have been previously established by a successful called to FTPLogin.

NOTE: The Action2 FTP methods (with the exception of FTPQuickSend and FTPQuickRetrieve) may not function properly in the AutoMate™ BASIC IDE. To more accurately use these functions, run the script from within an AutoMate™ task using the “Run a BASIC Script” action.

FTPDownload Method

Action2 Example

Encapsulates the AutoMate™ action in the "FTP" group, "FTP Download." The method attempts to download the specified file from the FTP server previously connected to using the FTPLogin method.

Declaration

function FTPDownload (varFTPSource As Variant,
varFTPDest As Variant,
intTransType As Integer,
intIncludeSubDirs As Integer) As Integer

Parameters

varFTPSource

The full path of the file on the FTP server to be downloaded.

varFTPDest

The full path on the local machine of where the file should be downloaded to.

intTransType

The transfer type AutoMate™ should use to download the file:

Value	Meaning
1	Default (Auto-detect)
2	ASCII/Text
3	Binary

intIncludeSubDirs

If set to "0", the download command applies to only the current directory. If set to "1", the download command will go through all subdirectories as well.

Description

The FTPDownload method will attempt to download the file specified by *varFTPSource* to the local machine.

A connection to the FTP server must have been previously established by a successful called to FTPLogin.

NOTE: The Action2 FTP methods (with the exception of FTPQuickSend and FTPQuickRetrieve) may not function properly in the AutoMate™ BASIC IDE. To more accurately use these functions, run the script from within an AutoMate™ task using the "Run a BASIC Script" action.

FTPUpload Method

Action2 Example

Encapsulates the AutoMate™ action in the "FTP" group, "FTP Upload." The method attempts to upload the specified file to the FTP server previously connected to using the FTPLogin method.

Declaration

function FTPUpload (varFTPSource As Variant,
varFTPDest As Variant,
intTransType As Integer,
intIncludeSubDirs As Integer) As Integer

Parameters

varFTPSource

The full path of the local file to be uploaded to the FTP server.

varFTPDest

The full path on the FTP server of where the file should be placed.

intTransType

The transfer type AutoMate™ should use to download the file:

Value	Meaning
1	Default (Auto-detect)
2	ASCII/Text
3	Binary

intIncludeSubDirs

If set to "0", the upload command applies to only the current directory. If set to "1", the upload command will go through all subdirectories as well.

Description

The FTPUpload method will attempt to upload the file specified by *varFTPSource* to the FTP server using the filename *varFTPDest.*

A connection to the FTP server must have been previously established by a successful called to FTPLogin .

NOTE: The Action2 FTP methods (with the exception of FTPQuickSend and FTPQuickRetrieve) may not function properly in the AutoMate™ BASIC IDE. To more accurately use these functions, run the script from within an AutoMate™ task using the "Run a BASIC Script" action.

CheckForMail Method

Action2 Example

Encapsulates the AutoMate™ action in the "Email" group, "Check For Mail." The method attempts to contact a POP3 server, login as the requested user, and check if any messages are waiting for retrieval.

Declaration

function CheckForMail (varServer As Variant,
varUsername As Variant,
varPassword As Variant,
intWaitingMailAction As Integer,
intNoWaitingMailAction As Integer) As Integer

Parameters

varServer

The hostname or IP address of the POP3 server to connect to.

varUsername

The username to use when connecting to the POP3 server.

varPassword

The clear text password to use for the username connecting to the POP3 server.

intWaitingMailAction

The action for AutoMate™ to take if there are any messages waiting for retrieval on the POP3 server:

Value	Meaning
0	Stop this task (method returns 2)
1	Continue this task (method returns 0)

intNoWaitingMailAction

The action for AutoMate™ to take if there are no messages waiting for retrieval on the POP3 server.

Value	Meaning
0	Stop this task (method returns 2)
1	Continue this task (method returns 0)

Description

The CheckForMail method will attempt to connect to the POP3 mail server specified by *varServer* as the user *varUsername*. It will then ask the POP3 server how many messages are in the user's mailbox. If there are any messages waiting, AutoMate™ carries out the action specified by *intWaitingMailAction*. If there are no messages waiting, AutoMate™ carries out the action specified by *intNoWaitingMailAction*.

The CheckForMail method performs the *intWaitingMailAction* parameter if there are *any* messages waiting on the server, not necessarily if there are any unread messages.

The method returns different values based on the state of the POP3 mailbox and the parameters of *intWaitingMailAction* and *intNoWaitingMailAction* (see above). It return “1” if the POP3 server cannot be contacted.

SendEmail Method

Action2 Example

Encapsulates the AutoMate™ action in the "Email" group, "Send Message." The method attempts to send an email message through the specified SMTP server.

Declaration

function SendEmail (varServer As Variant,
varUsername As Variant,
varTo As Variant,
varCC As Variant,
varSubject As Variant,
varBodyFilename As Variant,
varAttachmentFilename As Variant) As Integer

Parameters

varServer

The hostname or IP address of the SMTP server to use to send the email message.

varUsername

The username of the user with forwarding privileges on the SMTP server.

varTo

The email address (e.g. cowpoke@cowboy.com) of the primary recipient for the email message.

varCC

The email address or addresses of CC recipients of the message. Multiple email addresses can be supplied by separating each one with a semi-colon (e.g. ranchmaster@cowboy.com;masterchef@ranchhouse.com)

varSubject

The subject of the email message. This will appear in the **Subj** section of the sent message. This can be blank.

varBodyFilename

The full path and filename of the file which contains the contents of the message to be sent.

varAttachmentFilename

The full path to the binary attachment to send with the message. AutoMate™ currently supports only one attachment.

Description

The SendEmail method attempts to send the contents of text file *varBodyFilename* as an email message using the SMTP server supplied by *varServer*.

To send a message, first create a text file using your favorite text editor. Inside the text file, type out the message body of the email message. The body can contain AutoMate™ variables and constants; the

body will be parsed and processed by AutoMate™ before it is sent. Save the file and make note of its location.

Next, use the SendEMail method, supplying the primary recipient in the *varTo* address, and any carbon-copy recipients in the *varCC* parameter, separating their address with a semi-colon. Optionally, you can supply a subject heading for the message using the *varSubject* parameter. Use the path to the text file you previously created with the message contents in *varBodyFilename*, and a path to any attachment you wish to send in *varAttachmentFilename*.

The method returns ‘‘0’’ if the message was submitted to the SMTP server properly; otherwise it returns ‘‘1.’’ Note that the function succeeds if the message is successfully *submitted*, not whether or not the message necessarily received by the recipient(s).

MoveMouse Method

Action2 Example

Encapsulates the AutoMate™ action in the “Mouse” group, “Move To Location.” The method attempts to move the mouse pointer to a specific point on the desktop.

Declaration

function MoveMouse (intXPos As Integer,
    intYPos As Integer) As Integer

Parameters

intXPos

The X-coordinate to place the mouse pointer.

varYPos

The Y-coordinate to place the mount pointer.

Description

The MoveMouse method moves the mouse to the x, y point specified by the *intXPos* and *intYPos* parameters.

LeftClick Method

Action2 Example

Encapsulates the AutoMate™ action in the “Mouse” group, “Left Click.” The method simulates a single click of the left mouse button.

Declaration

function LeftClick () As Integer

Parameters

None

Description

The LeftClick method simulates a user pressing the left mouse button once at the current location of the mouse pointer.

LeftDblClick Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘Mouse’’ group, ‘‘Left Double Click.’’ The method simulates a double click of the left mouse button.

Declaration

function LeftDblClick () As Integer

Parameters

None

Description

The LeftDblClick method simulates a user double-clicking the left mouse button at the current location of the mouse pointer.

RightClick Method

Action2 Example

Encapsulates the AutoMate™ action in the “Mouse” group, “Right Click.” The method simulates a single click of the right mouse button.

Declaration

function RightClick () As Integer

Parameters

None

Description

The RightClick method simulates a user pressing the right mouse button once at the current location of the mouse pointer.

MiddleClick Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘Mouse’’ group, ‘‘Middle Click.’’ The method simulates a single click of the middle mouse button.

Declaration

function MiddleClick () As Integer

Parameters

None

Description

The MiddleClick method simulates a user pressing the middle mouse button once at the current location of the mouse pointer.

RightDblClick Method

Action2 Example

Encapsulates the AutoMate™ action in the “Mouse” group, “Right Double Click.” The method simulates a double click of the right mouse button.

Declaration

function RightDblClick () As Integer

Parameters

None

Description

The RightDblClick method simulates a user double-clicking the right mouse button at the current location of the mouse pointer.

MiddleDblClick Method

Action2 Example

Encapsulates the AutoMate™ action in the “Mouse” group, “Middle Double Click.” The method simulates a double click of the middle mouse button.

Declaration

function MiddleDblClick () As Integer

Parameters

None

Description

The MiddleDblClick method simulates a user double-clicking the middle mouse button at the current location of the mouse pointer.

AMCopyFile Method

Action2 Example

Encapsulates the AutoMate™ action in the “File I/O” group, “Copy a file.” The method attempts to copy the specified file from one location to another.

Declaration

function AMCopyFile (varSourceFile As Variant,
varDestFile As Variant) As Integer

Parameters

varSourceFile

The full path to the file to be copied.

varDestFile

The full path to the destination.

Description

The AMCopyFile method attempts to copy the source file *varSourceFile* to the location and name specified by *varDestFile*.

If the file is successfully copied, AMCopyFile returns “0.” If an error occurs, AMCopyFile returns “1.” In this case, use the GetLastError property to retrieve the error message AutoMate™ generated when trying to copy the file.

The AMCopyFile method supports wildcards.

AMMoveFile Method

Action2 Example

Encapsulates the AutoMate™ action in the "File I/O" group, "Move a file." The method attempts to move the specified file from one location to another.

Declaration

function AMMoveFile (varSourceFile As Variant,
varDestFile As Variant) As Integer

Parameters

varSourceFile

The full path to the file to be moved.

varDestFile

The full path to the destination.

Description

The AMMoveFile method attempts to move the source file *varSourceFile* to the location and name specified by *varDestFile*.

If the file is successfully moved, AMMoveFile returns "0." If an error occurs, AMMoveFile returns "1." In this case, use the GetLastError property to retrieve the error message AutoMate™ generated when trying to move the file.

The AMMoveFile method supports wildcards and can move files between drives.

The AMMoveFile method is identical to the AMRenameFile method.

AMRenameFile Method

Action2 Example

Encapsulates the AutoMate™ action in the "File I/O" group, "Rename a file." The method attempts to rename the specified file from one filename to another.

Declaration

function AMRenameFile (varOldName As Variant,
varNewName As Variant) As Integer

Parameters

varOldName

The full path to the file to be renamed.

varNewName

The full path and new name of the file.

Description

The AMRenameFile method attempts to rename the source file *varOldName* to *varNewName.*

If the file is successfully renamed, AMRenameFile returns "0." If an error occurs, AMRenameFile returns "1." In this case, use the GetLastError property to retrieve the error message AutoMate™ generated when trying to rename the file.

The AMRenameFile method supports wildcards.

The AMRenameFile method is identical to the AMMoveFile method.

AMDeleteFile Method

Action2 Example

Encapsulates the AutoMate™ action in the “File I/O” group, “Delete a file.” The method attempts to delete the specified file.

Declaration

function AMDeleteFile (varFilename As Variant) As Integer

Parameters

varFileName

The full path to the file to be deleted.

Description

The AMDeleteFile method attempts to delete the file specified by *varFilename*.

Use the AMDeleteFile method with caution. The method permanently deletes the file from the system. It does *not* move the file to the recycle bin.

The AMDeleteFile method supports wildcards.

If the file is successfully deleted, AMDeleteFile returns “0.” If an error occurs, AMDeleteFile returns “1.” In this case, use the GetLastError property to retrieve the error message AutoMate™ generated when trying to delete the file.

AMMakeDir Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘File I/O’’ group, ‘‘Create a directory.’’ The method attempts to create the specified directory.

Declaration

function AMMakeDir (varDirectory As Variant) As Integer

Parameters

varDirectory

The name of the directory to be created (including path, if necessary)

Description

The AMMakeDir method attempts to create the directory specified by *varDirectory*.

If the directory is successfully created, AMMakeDir returns ‘‘0.’’ If an error occurs, AMMakeDir returns ‘‘1.’’ In this case, use the GetLastError property to retrieve the error message AutoMate™ generated when creating the directory.

AMRemoveDir Method

Action2 Example

Encapsulates the AutoMate™ action in the "File I/O" group, "Remove a directory." The method attempts to remove the specified directory.

Declaration

function AMRemoveDir (varDirectory As Variant) As Integer

Parameters

varDirectory

The name of the directory to be removed (including path, if necessary)

Description

The AMMakeDir method attempts to remove the directory specified by *varDirectory*.

If the directory is successfully removed, AMRemoveDir returns "0." If an error occurs, AMRemoveDir returns "1." In this case, use the GetLastError property to retrieve the error message AutoMate™ generated when creating the directory.

The directory to be removed must be empty before calling AMRemoveDir.

AMChangeDir Method

Action2 Example

Encapsulates the AutoMate™ action in the “File I/O” group, “Change directory.” The method attempts to change the current default directory to the specified directory.

Declaration

function AMChangeDir (varDirectory As Variant) As Integer

Parameters

varDirectory

The name of the directory to be changed to.

Description

The AMChangeDir method attempts to change the current default directory to *varDirectory.* If successful, all subsequent file I/O commands use this directory as their relative path.

DDECommand Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘General’’ group, ‘‘DDE Command.’’ The method attempts to send a DDE Command.

Declaration

function DDECommand (varText As Variant,
varTopic As Variant,
varItem As Variant) As Integer

Parameters

varText

The text property of the DDE Command structure.

varTopic

The topic property of the DDE Command.

varItem

The item property of the DDE Command.

Description

The DDECommand method will issue a DDE command to the requested application.

See the Win32 SDK for more information on DDE Commands and their use.

ClipboardCut Method

Action2 Example

Encapsulates the AutoMate™ action in the “Clipboard” group, “Cut.” The method cuts the current user selection to the system clipboard.

Declaration

function ClipboardCut () As Integer

Parameters

None

Description

The ClipboardCut method cuts the current user selection to the system clipboard. AutoMate™ does this by simulating a “Ctrl-X” keyboard sequence. It is up to the user to first select the appropriate text or object to be cut to the clipboard before calling the ClipboardCut method, and to ensure that the focused application supports “Ctrl-X” as the cut command.

ClipboardCopy Method

Action2 Example

Encapsulates the AutoMate™ action in the “Clipboard” group, “Copy.” The method copies the current user selection to the system clipboard.

Declaration

function ClipboardCopy () As Integer

Parameters

None

Description

The ClipboardCopy method copies the current user selection to the system clipboard. AutoMate™ does this by simulating a “Ctrl-C” keyboard sequence. It is up to the user to first select the appropriate text or object to be copies to the clipboard before calling the ClipboardCopy method, and to ensure that the focused application supports “Ctrl-C” as the cut command.

ClipboardPaste Method

Action2 Example

Encapsulates the AutoMate™ action in the “Clipboard” group, “Paste.” The method pastes the current contents of the system clipboard at the user-selected location.

Declaration

function ClipboardPaste () As Integer

Parameters

None

Description

The ClipboardPaste method pastes the contents of the system clipboard to the current user selection. AutoMate™ does this by simulating a “Ctrl-V” keyboard sequence. It is up to the user to ensure the desired contents are on the clipboard and the selection (e.g., cursor or pointer) is in the appropriate place before calling the ClipboardPaste method.

ClipboardClear Method

Action2 Example

Encapsulates the AutoMate™ action in the “Clipboard” group, “Clear.” The method clears the system clipboard.

Declaration

function ClipboardClear () As Integer

Parameters

None

Description

The ClipboardClear method clears the contents of the system clipboard.

RebootMachine Method

Action2 Example

Encapsulates the AutoMate™ action in the “System” group, “Reboot.” The method forcibly reboots the machine.

Declaration

function RebootMachine () As Integer

Parameters

None

Description

The RebootMachine method forcibly and immediately reboots the machine. The system does not prompt to save contents of open applications.

NOTE: Use the RebootMachine method with caution. Because it is a forcible reboot, the user may lose any unsaved data without warning once the function is executed.

ShutdownMachine Method

Action2 Example

Encapsulates the AutoMate™ action in the “System” group, “Shutdown.” The method forcibly shuts down the machine.

Declaration

function ShutdownMachine () As Integer

Parameters

None

Description

The ShutdownMachine method forcibly and immediately shuts down the machine. The system does not prompt to save contents of open applications.

NOTE: Use the ShutdownMachine method with caution. Because it is a forcible shutdown, the user may lose any unsaved data without warning once the function is executed.

LogoffMachine Method

Action2 Example

Encapsulates the AutoMate™ action in the “System” group, “Logout.” The method forcibly logs off the current user.

Declaration

function LogoffMachine () As Integer

Parameters

None

Description

The LogoffMachine method forcibly and immediately logs the current user off. The system does not prompt to save contents of open applications.

NOTE: Use the LogoffMachine method with caution. Because it is a forcible log off, the user may lose any unsaved data without warning once the function is executed.

UpdateAutoMate™ Method

Action2 Example

Encapsulates the AutoMate™ action in the "System" group, "Update AutoMate™." The method attempts to check for a newer version of AutoMate™, and if one exists, updates the executables to the newest version.

Declaration

function UpdateAutoMate™ (intUpdateLocation As Integer,
varFTPServer As Variant,
varNetworkPath As Variant,
intShowProgress As Integer,
intForceReboot As Integer) As Integer

Parameters

intUpdateLocation

Indicates whether AutoMate™ should check an FTP site for an updated version, or use an installation that already exists on a local path:

Value	Meaning
0	Check the Unisyn site for a new version
1	Use an installation on a local path

varFTPServer

Checks the Unisyn FTP site for an updated version of AutoMate™. If left blank, AutoMate™ will attempt to contact ftp.unisyn.com for an update. This is only used if the *intUpdateLocation* parameter is set to "0."

varNetworkPath

The path to the installation to use. This is parameter is ignored unless *intUpdateLocation* is set to "1."

intShowProgress

Specifies whether or not to display a progress meter indicating the status of the download and update. Set to "1" to show a progress meter or "0" for silent upgrade.

intForceReboot

Some updates of AutoMate™ require that the system be rebooted before the changes take effect. If *intForceReboot* is set to "1", AutoMate™ will forcibly reboot the machine when the update is finished. If *intForceReboot* is set to "0", AutoMate™ will wait until the next reboot by the user before the changes take effect.

Description

The UpdateAutoMate™ method is used to automatically update AutoMate™ to the latest version available from the Unisyn FTP site. Setting *intUpdateLocation* to "0" and the *varFTPServer* to ftp.unisyn.com (or "" for default) will cause AutoMate™ to contact the Unisyn site, check for a newer version, and if one is present, download it and automatically install it.

The UpdateAutoMate™ method can also forcibly install a pre-existing or older installation over itself. To do this, set *intUpdateLocation* to “1” and set *varNetworkPath* to the location of the AutoMate™ setup file.

Note that the upgrade process requires that AutoMate™ shut itself down, run the setup executable, and then restart. Therefore, running the UpdateAutoMate™ method will cause the script and any running tasks to stop.

RemoveService Method

Action2 Example

Windows NT Machines Only

Encapsulates the AutoMate™ action in the "Services" group, "Remove a service." The method attempts to remove the specified service from the Windows NT Service Control Manager (SCM).

Declaration

function RemoveService (varServiceName As Variant,
intStopFirst As Integer) As Integer

Parameters

varServiceName

The name of the service (as it appears in the SCM) to be removed.

intStopFirst

Specifies whether or not AutoMate™ should attempt to stop the service before removing it. Set to "1" to attempt to stop the service first or set to "0" to remove the service.

Description

The RemoveService method attempts to locate the service *varServiceName* in the Service Control Manager and remove it.

Services should be stopped before attempting to remove them from the SCM. AutoMate™ may, however, take a longer period of time to remove the service if it attempts to stop a service that is already stopped. Therefore, you should set *intStopFirst* to "1" only if you know the service is running to avoid a performance hit.

RemoveService returns "0" if the service was successfully removed; otherwise, it returns "1." Use the GetLastError property to retrieve the error message generated by a failed removal attempt.

The function is applicable only to AutoMate™ installations running on Windows NT and Windows 2000 systems.

StartService Method

Action2 Example

Windows NT Machines Only

Encapsulates the AutoMate™ action in the "Services" group, "Start a service." The method attempts to start the specified service.

Declaration

function StartService (varServiceName As Variant) As Integer

Parameters

varServiceName

The name of the service (as it appears in the SCM) to be started.

Description

The StartService method attempts to start the service named *varServiceName*.

The method returns "0" if the service was started successfully; otherwise, it returns "1." Use the GetLastError property to retrieve the error message generated by a failed start attempt.

PauseService Method

Action2 Example

Windows NT Machines Only

Encapsulates the AutoMate™ action in the ‘‘Services’’ group, ‘‘Pause a service.’’ The method attempts to pause the specified service.

Declaration

function PauseService (varServiceName As Variant) As Integer

Parameters

varServiceName

The name of the service (as it appears in the SCM) to be paused.

Description

The PauseService method attempts to pause the service named *varServiceName*. The service must have the ability to be paused and unpaused. Check the documentation for the service if you are unsure whether or not it supports pausing.

Use the ContService method to continue a service that has been paused.

The method returns ’’0’’ if the service was paused successfully; otherwise it returns ‘‘1.’’ Use the GetLastError property to retrieve the error message generated by a failed pause attempt.

ContService Method

Action2 Example

Windows NT Machines Only

Encapsulates the AutoMate™ action in the "Services" group, "Continue a service." The method attempts to continue a paused service.

Declaration

function ContService (varServiceName As Variant) As Integer

Parameters

varServiceName

The name of the service (as it appears in the SCM) to be continued.

Description

The ContService method attempts to continue the service named *varServiceName*. The service must have been previously paused, either through the SCM or by the PauseService method.

The method returns "0" if the service was continued successfully; otherwise it returns "1." Use the GetLastError property to retrieve the error message generated by a failed continue attempt.

InstallService Method

Action2 Example

Windows NT Machines Only

Encapsulates the AutoMate™ action in the "Services" group, "Install a service." The method attempts to install a service application into the Service Control Manager (SCM).

Declaration

function InstallService (varExecutable As Variant,
varDisplayName As Variant,
varUsername As Variant,
varPassword As Variant,
intErrorType As Integer,
intServiceType As Integer,
intRunType As Integer,
intInteractive As Integer) As Integer

Parameters

varExecutable

The full path, including any command line parameters, to the service executable associated with this service entry.

varDisplayName

The name of the service as it is to appear in the SCM.

varUsername

If the service is to be run in the context of a specific user, supply the Username here.

varPassword

If the service is to be run in the context of a specific user, supply the password to *varUsername* here.

intErrorType

Specified the type of error control to associate with this service:

Value	Meaning
0	Ignore Error
1	Normal Error
2	Critical Error
3	Severe Error

intServiceType

The type of service to install:

Value	Meaning
0	Runs in own process

1	Shared process
2	Kernel driver
3	File system driver

intRunType

The run type of the service:

Value	Meaning
0	Automatic
1	Manual
2	Disabled
3	Boot
4	System

intInteractive

Specifies whether or not the service should be run as interactive. If it is an interactive process, set *intInteractive* to "1"; otherwise, set to "0." Note that if the process is interactive, the *varUsername* and *varPassword* variables are ignored.

Description

The InstallService method attempts to install a service application into the SCM in one simple to use step.

The use of services and the Service Control Manager, however, are beyond the scope of this help file. Please consult the Microsoft Win32 SDK under "Services" for more complete information about services and their use in the Windows NT platform.

The method returns "0" if the service was installed successfully; otherwise, it returns "1." Use the GetLastError property to retrieve the error message generated by a failed start attempt.

Wait Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘Flow Control’’ group, ‘‘Pause.’’ The method causes AutoMate™ to suspend task execution for the specified number of seconds.

Declaration

function Wait (intTimeToWait As Integer) As Integer

Parameters

intTimeToWait

The number of seconds AutoMate™ should wait before executing the next step or line of code.

Description

The Wait method pauses the execution of AutoMate™ for the number of seconds specified by *intTimeToWait*.

Login Method

Action2 Example

AutoMate™ NT Service Edition Only

Encapsulates the AutoMate™ action in the "System" group, "Login." The method attempts to set the security context of the currently running task to that of another user.

Declaration

function Login (varUsername As Variant,
varPassword As Variant,
varDomainName As Variant,
intNewDesktop As Integer,
intInteractive As Integer,
intLaunchExplorer As Integer,
intLockMouse As Integer,
intLockKeyboard As Integer) As Integer

Parameters

varUsername

The name of the user the AutoMate™ task should act as.

varPassword

The password of the user the AutoMate™ task should act as.

varDomainName

The domain that *varUsername* is a part of.

intNewDesktop

Set to "1" if AutoMate™ should login the user in a new interactive workspace and setup their default desktop. Set to "0" if no new interactive session is required.

intInteractive

Specified whether or not the new login session should allow the user to interactively work with the desktop. If set to "0", the session logins in the background, refusing user input and carrying out the task securely. Set to "1" if the user should access the new desktop.

intLaunchExplorer

Set to "1" if explorer.exe should be run on the new interactive desktop This establishes a new desktop. Set to "0" if no desktop is required.

intLockMouse

Set to "1" to stop the mouse from being accessed during the login session. Set to "0" to allow access to the mouse.

intLockKeyboard

Set to "1" to lock the keyboard during the login session. Set to "0" to allow access to the keyboard.

Description

The AutoMate™ NT Service edition starts each individual task in the space of a specific user. Usually, this user is the one set by the Default Login, set in Preferences. Sometimes it is necessary to override the default user and have the task operate in the security context of another user, for example the administrator.

In these cases, use the Login method to change the security context of the task to that of a different user. AutoMate™ will log the user on in the background, and each subsequent step will be executed as if the new user were at the computer.

See the AutoMate™ Help File under "NT Service Edition Notes" for more information about user logins and security contexts.

PromptUser Method

Action2 Example

Encapsulates the AutoMate™ action in the "Flow Control" group, "Prompt." The method shows a dialog box specified text and a selection of buttons and performs variable defined actions based on their selection.

Declaration

function PromptUser (varText As Variant,
intType As Integer,
intActionOnOK As Integer,
intFailOnCancel As Integer,
intFailOnNo As Integer,
varTaskToStart As Variant) As Integer

Parameters

varText

The text to display inside the dialog box.

intType

Specifies which buttons are to be available on the dialog box:

Value	Meaning
0	OK only
1	OK and Cancel
2	Yes and No
3	Yes, No, and Cancel

intActionOnOK

Specifies the action AutoMate™ should take if the user clicks the OK or Yes button:

Value	Meaning
0	Start another task (specified by varTaskToStart)
1	Stop the task (method returns 2)
2	Continue the task (method returns 0)

intFailOnCancel

Specifies whether the method should return a "1", indicating a step failure, if the user clicks the Cancel button (applicable only if the *intType* parameter is 1 or 3)

intFailOnNo

Specifies whether the method should return a "1", indicating a step failure, if the user clicks the No button (applicable only if the *intType* parameter is 2 or 3)

varTaskToStart

The name of the task AutoMate™ should start if the user clicks the OK or Yes button, and *intActionOnOK* is set to "0."

Description

The PromptUser method displays a user interactive dialog bog with the text *varText* and waits for the user to make a selection. The return value of the method depends on the *intActionOnOK, intFailOnCancel* and *intFailOnNo* parameters.

Use the PromptUser method to allow a user to interactively select the flow of a running task.

WaitForWindow Method

Action2 Example

Encapsulates the AutoMate™ action in the "Flow Control" group, "Wait for a window." The method pauses AutoMate™ for a specified period of time or until a window with the specified text appears.

Declaration

```
function WaitForWindow (varWindowText As Variant,
    intExactMatch As Integer,
    intIncludeChildren As Integer,
    intWaitTime As Integer,
    intCheckInterval As Integer,
    intNoWindowAction As Integer,
    varTaskToStart As Variant) As Integer
```

Parameters

varWindowText

The title bar text of the window to wait for.

intExactMatch

Specifies whether or not *varWindowText* must be matched exactly by any appearing window. Set to "1" if *varWindowText* must be matched exactly or "0" for a substring or partial match.

intIncludeChildren

Set to "1" to check all windows in the system for *varWindowText*. Set to "0" to just check top level or main application windows.

intWaitTime

The number of seconds AutoMate™ should wait for the window to appear. If the window does not appear, AutoMate™ checks the *intNoWindowAction* parameter for what to do. To wait an infinite time, set to –1.

intCheckInterval

The number of seconds between times AutoMate™ will enumerate all system windows in search for the wanted window. Setting this to a low value (e.g., 1-second) slightly increases CPU utilization.

intNoWindowAction

Specifies what action AutoMate™ should take if *intWaitTime* expires and the specified window still has not appeared:

Value	Meaning
0	Start another task (specified by varTaskToStart)
1	Stop the task (method returns 2)
2	Continue the task (method returns 0)

varTaskToStart

The task AutoMate™ should start if the window does not appear within *intWaitTime*, and

intNoWindowAction is set to "0."

Description
The WaitForWindow method pauses AutoMate™ until a window with the specified text appears.

Use WaitForWindow to wait for the system to enter a specific state (for example, an error message box appears) before continuing to the next step or next line of code.

WaitForWindowToDisappear Method

Action2 Example

Encapsulates the AutoMate™ action in the "Flow Control" group, "Wait for a window to disappear." The method pauses AutoMate™ for a specified period of time or until a window with the specified text disappears.

Declaration

function WaitForWindowToDisappear (varWindowTitle As Variant,
intExactMatch As Integer,
intIncludeChildren As Integer,
intWaitInfinite As Integer,
intInterval As Integer,
intWaitTime As Integer,
intStillPresentAction As Integer,
varTaskToStart As Variant) As Integer

Parameters

varWindowText

The title bar text of the window to wait for.

intExactMatch

Specifies whether or not *varWindowText* must be matched exactly by any appearing window. Set to "1" if *varWindowText* must be matched exactly or "0" for a substring or partial match.

intIncludeChildren

Set to "1" to check all windows in the system for *varWindowText*. Set to "0" to just check top level or main application windows.

intWaitInfinite

Specifies whether AutoMate™ should wait an infinite period of time for the window to disappear. Set to "0" to wait an infinite period of time and ignore the *intInterval* and *intWaitTime* parameters. Set to "1" to wait for a specific period of time.

intInterval

The number of seconds between times AutoMate™ will enumerate all system windows in search for the wanted window. Setting this to a low value (e.g., 1-second) slightly increases CPU utilization.

intWaitTime

The number of seconds AutoMate™ should wait for the window to appear. If the window does not appear, AutoMate™ checks the *intNoWindowAction* parameter for what to do.

intStillPresentAction

Specifies what action AutoMate™ should take if *intWaitTime* expires and the specified window is still present:

Value	Meaning

0	Start another task (specified by varTaskToStart)
1	Stop the task (method returns 2)
2	Continue the task (method returns 0)

varTaskToStart

The task AutoMate™ should start if the window does not appear within *intWaitTime*, and *intNoWindowAction* is set to "0."

Description

The WaitForWindowToDisappear method pauses AutoMate™ until a window with the specified text disappears.

Use WaitForWindowToDisappear to wait for the system to enter a specific state (for example, an application's main window closes, indicating the end of a program's run) before continuing to the next step or next line of code.

StartTask Method

Action2 Example

Encapsulates the AutoMate™ action in the "Flow Control" group, "Start Another Task." The method causes AutoMate™ to start another task.

Declaration

function StartTask (varTaskName As Variant,
    intWaitUntilDone As Integer,
    intFailThis As Integer) As Integer

Parameters

varTaskName

The name of the task to be started. The task must be in the currently loaded task list.

intWaitUntilDone

Specifies whether this script should wait for the new task to complete before continuing. Set to "1" to wait for the new task to finish or "0" to continue with this task after the new one starts.

intFailThis

Specifies whether this task should fail as a result of the failure of *varTaskName*. If *intFailThis* is set to "1", the method returns "1" if *varTaskName* fails; otherwise, the method returns "0." This parameter is ignored unless *intWaitUntilDone* is set to "1."

Description

Use the StartTask method to start executing another task alongside the currently running task. StartTask can wait until the new task is finished before continuing by setting *intWaitUntilDone* to "1."

StartTask returns "0" if the task started successfully, unless *intWaitUntilDone* is set to "1" and *intFailThis* is set to "1." In this case, StartTask returns "0" if the new task started and ended successfully, or it returns "1" if the new task either did not start or failed.

DisableThisTask Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘Flow Control’’ group, ‘‘Disable This Task.’’ The method disables the currently running task.

Declaration

function DisableThisTask () As Integer

Parameters

None

Description

Use the DisableThisTask method to disable the currently running task. The task becomes disabled once the script and any remaining steps are completed.

To stop the task immediately after the script is completed and abort any remaining steps, set CurrentTask2.Active to ‘‘0.’’

LockKeyboard Method

Action2 Example

Encapsulates the AutoMate™ action in the “Security” group, “Lock Keyboard.” The method locks the keyboard.

Declaration

function LockKeyboard () As Integer

Parameters

None

Description

Use the LockKeyboard method to lock the keyboard from user input. Once the method is executed, any keypress caught by the keyboard are discarded and not echoed to the system, effectively blocking any user interaction with the system.

To re-enable the keyboard, use the UnlockKeyboard method.

The keyboard automatically unlocks itself at the end of the task to prevent accidentally locking the keyboard until the next reboot.

LockMouse Method

Action2 Example

Encapsulates the AutoMate™ action in the “Security” group, “Lock Mouse.” The method locks the mouse.

Declaration

function LockMouse () As Integer

Parameters

None

Description

Use the LockMouse method to lock the mouse from user input. Once the method is executed, any mouse movements and clicks are discarded and not echoed to the system, effectively blocking any user interaction with the system.

To reenable the mouse, use the UnlockMouse method.

The mouse automatically unlocks itself at the end of the task to prevent accidentally locking the mouse until the next reboot.

UnlockKeyboard Method

Action2 Example

Encapsulates the AutoMate™ action in the “Security” group, “Unlock Keyboard.” The method unlocks the keyboard.

Declaration

function UnlockKeyboard () As Integer

Parameters

None

Description

Use the UnlockKeyboard method to unlock the keyboard after it was previously locked using the LockKeyboard method.

The keyboard automatically unlocks itself at the end of the task to prevent accidentally locking the keyboard until the next reboot.

UnlockMouse Method

Action2 Example

Encapsulates the AutoMate™ action in the “Security” group, “Unlock Mouse.” The method unlocks the mouse.

Declaration

function UnlockMouse () As Integer

Parameters

None

Description

Use the UnlockMouse method to unlock the mouse after it was previously locked using the LockMouse method.

The mouse automatically unlocks itself at the end of the task to prevent accidentally locking the mouse until the next reboot.

Password Method

Action2 Example

Encapsulates the AutoMate™ action in the “Security” group, “Password.” The method displays a dialog box prompting for a password.

Declaration

function Password (varMessageText As Variant,
    varPassword As Variant,
    intMaxRetries As Integer As Integer

Parameters

varMessageText

The text to be displayed inside the password dialog box.

varPassword

The clear text password to wait for. This is the password the user must enter to successfully proceed.

intMaxRetries

The number of times the user is allowed to retry entering the password before the dialog gives up and fails the method.

Description

Use the Password method to display a user dialog requested a password. The text entered by the user is ghosted using asterisks. The user can attempt to reenter the password *intMaxRetries* times before the method fails.

The Password method returns “0” if the user enters the password correctly; otherwise, it returns “1.”

PlayMouse Method

Action2 Example

Encapsulates the AutoMate™ action in the "Recorder" group, "Record events." The method plays the mouse and keyboard events previously recorded with AutoMate's mouse recorder.

Declaration

function PlayMouse (varFilename As Variant) As Integer

Parameters

varFilename

The filename of the events recording to play.

Description

The PlayMouse method plays back the system events recorded by the AutoMate™ Configuration Manager.

To make the series of events to play back, first open the AutoMate™ Configuration Manager. Make a new task. From the Step Builder, select "Record Events" from under the "Recorder" group. Enter a filename for the recording, and click "Record." This will begin recording the movements of your mouse and the keystrokes on your keyboard. When finished, press Ctrl-F12. Then, you can use the file you just created in a call to PlayMouse to playback the events.

SetVar Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘Variables’’ group, ‘‘Set variable.’’ The method sets a previously created AutoMate™ variable to the value specified.

Declaration

function SetVar (varVarName As Variant,
varNewValue As Variant) As Integer

Parameters

varVarName

The *name* of the AutoMate™ variable whose value to change.

varNewValue

The new value of the variable.

Description

Use the SetVar method to explicitly set the value of a variable created within AutoMate™ using the Create Variable step.

When the script starts, AutoMate™ will automatically allocate the variables from your task into the script using the same names from the task. For example, if before the SCRIPT: step in your task you have created a variable called MYVAR, the variable MYVAR will be accessible in your script like any other regular variable. You can safely modify and work with this variable like another other VBA object. When the script ends, AutoMate™ will automatically write any changes you made to the variable back to the task.

If, however, you need to create a NEW variable to be passed back to AutoMate™ when the script is finished, you will need to create the variable *before* entering the script by using a *Create Variable* action. This is because AutoMate™ can only track those variables that were created *before* the script started.

CreateVar Method

Action2 Example

Encapsulates the AutoMate™ action in the "Variables" group, "Create variable." The method creates a new variable for use in an AutoMate™ task.

Declaration

function CreateVar (varVarName As Variant,
    varInitialValue As Variant) As Integer

Parameters

varVarName

The *name* of the AutoMate™ variable to create.

varInitialValue

The new variable's initial value.

Description

Use the CreateVar method to create a new AutoMate™ variable that can be accessed and modified within the currently executing AutoMate™ task.

When the script starts, AutoMate™ will automatically allocate the variables from your task into the script using the same names from the task. For example, if before the SCRIPT: step in your task you have created a variable called MYVAR, the variable MYVAR will be accessible in your script like any other regular variable. You can safely modify and work with this variable like another other VBA object. When the script ends, AutoMate™ will automatically write any changes you made to the variable back to the task.

If, however, you need to create a NEW variable to be passed back to AutoMate™ when the script is finished, you will need to use the CreateVar method to first create the variable, and explicitly set any changes to the variable using the SetVar method. This is because AutoMate™ can only track those variables that were created *before* the script started.

SendVar Method

Action2 Example

Encapsulates the AutoMate™ action in the “Variables” group, “Send variable.” The method sends the contents of an AutoMate™ variable as keystrokes.

Declaration

function SendVar (varVarName As Variant) As Integer

Parameters

varVarName

The *name* of the AutoMate™ variable to send.

Description

Use the SendVar method to send the contents of an AutoMate™ variable as keystrokes.

This method call is the same as using the SendKeystrokes method and setting the *varKeysToSend* to an AutoMate™ variable.

InputToVar Method

Action2 Example

Encapsulates the AutoMate™ action in the ‘‘Variables’’ group, ‘‘Input variable.’’ The method displays a dialog box for the user to enter a value, which is then stored into the specified AutoMate™ variable.

Declaration

function InputToVar (varVarName As Variant,
    varPromptTitle As Variant,
    varPromptText As Variant) As Integer

Parameters

varVarName

The *name* of the AutoMate™ variable to input the user response into.

varPromptTitle

The text to display in the title bar of the dialog box.

varPromptText

The text to display within the dialog box.

Description

Use the InputToVar method to allow a user to interactively input a value for a variable.

This method call is the same as using the SendKeystrokes method and setting the *varKeysToSend* to an AutoMate™ variable.

RunScript Method

Action2 Example

Encapsulates the AutoMate™ action in the “BASIC Scripting” group, “Run a BASIC Script.” The method runs another AutoMate™ BASIC Script.

Declaration

function RunScript (varFilename As Variant) As Integer

Parameters

varFilename

The filename of the script to execute.

Description

Use the RunScript method to run another AutoMate™ BASIC script from within the currently executing script.

If possible, try to avoid running nesting scripts using the RunScript command. Each call to RunScript allocates memory for a AutoMate™ BASIC interpreter, which uses significant system resources. Calling RunScript multiple times without allowing the interpreter to clean up its resources could a rapid loss of memory.

AutoMate2 Object

Action2 Legend Grouped Listing

- AMChangeDir
- AMCopyFile
- AMDeleteFile
- AMMakeDir
- AMMoveFile
- AMRemoveDir
- AMRenameFile
- CheckForMail
- ClipboardClear
- ClipboardCopy
- ClipboardCut
- ClipboardPaste
- ContService
- DDECommand
- DialUp
- DisableThisTask
- FocusWindowContaining
- FTPChangeDirectory
- FTPDelete
- FTPDownload
- FTPLogin
- FTPLogout
- FTPMakeDirectory
- FTPQuickRetrieve
- FTPQuickSend
- FTPRemoveDirectory
- FTPRename
- FTPUpload
- GetFocusedWindowName
- InstallService
- LeftClick
- LeftDblClick
- LeftMouseDown
- LeftMouseUp
- LockKeyboard
- LockMouse
- Login
- LogoffMachine
- Message
- MiddleClick
- MiddleDblClick
- MiddleMouseDown
- MiddleMouseUp

MoveMouse
OpenDocument
OpenWebpage
Password
PasteKeys
PauseService
PingMachine
PlayMouse
PlaySound
PrintDocument
PromptUser
QuitAutoMate™
RebootMachine
Reminder
RemoveService
RightClick
RightDblClick
RightMouseDown
RightMouseUp
RunScript
SendEmail
SendKeystrokes
SendRawStep
SendVar
ShutdownMachine
StartApplication
StartService
StartTask
StopService
StopSound
UnlockKeyboard
UnlockMouse
Unzip
UpdateAutoMate™
Wait
WaitForWindow
WaitForWindowToDisappear
WindowClose
WindowFocus
WindowHide
WindowMaximize
WindowMinimize
WindowRestore
WindowUnhide
Zip

StopService Method

Action2 Example

Windows NT Machines Only

Encapsulates the AutoMate™ action in the “Services” group, “Stop a service.” The method attempts to stop the specified service.

Declaration

function StopService (varServiceName As Variant) As Integer

Parameters

varServiceName

The name of the service (as it appears in the SCM) to stop.

Description

The PauseService method attempts to stop the service named *varServiceName*. The service must have the ability to be stopped. Check the documentation for the service if you are unsure whether or not it supports pausing.

The method returns “0” if the service was stopped successfully; otherwise, it returns “1.” Use the GetLastError property to retrieve the error message generated by a failed stop attempt.

QuitAutoMate™ Method

Action2 Example

Encapsulates the AutoMate™ action in the “System” group, “Quit AutoMate™.” The method attempts to stop and shutdown the AutoMate™ Task Service.

Declaration

function QuitAutoMate™ () As Integer

Parameters

None

Description

Use the QuitAutoMate™ method to stop and immediately shutdown the AutoMate™ Task Service. This automatically stops all running tasks and removes the AutoMate™ Task Service from the system tray.

SendRawStep Method

Action2 Example

Sends a step to AutoMate™ in raw text format.

Declaration
function SendRawStep (varStepText As Variant) As Integer

Parameters
varStepText
The step to be executed in AutoMate™ in raw text format.

Description
Use the SendRawStep format to send a step in raw text format to AutoMate™.

The method returns "0" if the step passed was successfully executed; otherwise, it returns "1."

GetLastError Property

Action2 Read Only Example

Retrieve the text of the last error generated by AutoMate™.

Declaration

property GetLastError As Variant

Description

When an Action method fails in AutoMate™, AutoMate™ populates the GetLastError property with the text of the error. Use this property to retrieve the error text.

Action2 properties

- GetLastError

CurrentTask2 Properties

CurrentTask2 Legend

- Active
- Details
- Events
- Frequency
- FrequencyString
- Hotkey
- Interval
- IntervalType
- LaunchDate
- LaunchTime
- RunLate
- ScheduleLate
- StepCount
- TaskTriggers
- TaskName
- WindowName

CurrentTask2 Object

Properties Methods

The CurrentTask2 object encapsulates the properties of a task, and methods that act upon that task.

Description

The CurrentTask2 object contains properties and methods to allow you to view and modify attributes of the currently executing task. Use this object to control specific parts of the executing task “on the fly.”

The CurrentTask2 object provides all the functionality of the CurrentTask interface provided with previous versions of AutoMate™, but also offers extended capabilities that encapsulate the new features available in newer releases of AutoMate™. The CurrentTask2 interface should be used instead of the CurrentTask interface in newer scripts. The CurrentTask2 interface is still available for backward compatibility.

When a script is started, AutoMate™ automatically instantiates an CurrentTask2 object and populates it with the properties of the currently running task. This object is referred to as “CurrentTask2.” Use CurrentTask2 to access and modify the properties of the currently running task, as well perform actions on the task such as rescheduling or saving it to a file.

Remarks

When creating or altering steps for the currently running task the format must be the same as steps inside the Step Builder.You must make sure that all string paramaters are inside double quotes correctly. Here is an example of the correct way to format the string for AutoMate™.

```
varNewStep = "START: ""C:\Windows\Notepad.exe"","""",0,""c:\"",0,1,0,"""""

CurrentTask2.AppendTaskStep(varNewStep)
```

Notice the use of double quotes around strings. All methods in CurrentTask2 will require this in order for AutoMate™ to be able to use the new step.

CurrentTask2 Methods

CurrentTask2

- AppendTaskStep
- ClearSteps
- DeleteTaskStep
- GetStepCommand
- GetStepParameter
- GetStepParamCount
- GetStepStatus
- GetTaskStep
- InsertTaskStep
- LoadFromFileEx
- SaveToFileEx
- SetStepStatus
- SetTaskStep

TaskName Property

CurrentTask2 Example

Specifies the name of the currently executing task.

Declaration

Dim TaskName As Variant

Description

Use the TaskName property to retrieve or set the name of the currently running task. If you change the name of the task during execution of your script, be sure it is unique. Problems may arise in the AutoMate™ Configuration Manager if you attempt to set the task to a name that already exists in your tasklist.

LaunchDate Property

CurrentTask2 Example

Specifies the date the task is next scheduled to launch.

Declaration

Dim LaunchDate As Variant

Description

Use the LaunchDate property to retrieve or set the date the task is scheduled to launch. The property is set when the task begins execution. The date format must follow the format set in Regional Settings. The new LaunchDate is set when the task finishes executing.

To view the format required for your regional settings, view Regional Settings from the Control Panel. Click the "Date" tab, and note the "Short Date Format". This is the format AutoMate™ will expect.

NOTE: Be careful when changing the LaunchDate of a task that uses the Schedule trigger. Because the LaunchDate is set when the task *starts* executing, the schedule trigger may change the date on you while the task is executing. Therefore, use caution when adjusting the LaunchDate relative to its initial value.

LaunchTime Property

CurrentTask2 Example

Specifies the time the task is next scheduled to launch.

Declaration

Dim LaunchTime As Variant

Description

Use the LaunchTime property to retrieve or set the time the task is scheduled to launch. The property is set when the task begins execution. The time format must follow the format set in Regional Settings. The new LaunchTime is set when the task finishes executing.

To view the format required for your regional settings, view Regional Settings from the Control Panel. Click the "Time" tab, and note the "Time Style". This is the format AutoMate™ expects.

NOTE: Be careful when changing the LaunchTime of a task that uses the Schedule trigger. Because the LaunchTime is set when the task *starts* executing, the schedule trigger may change the time on you while the task is executing. Therefore, use caution when adjusting the LaunchTime relative to its initial value.

Frequency Property

CurrentTask2 Example

Specifies the how often the schedule task runs.

Declaration

Dim Frequency As Integer

Description

Use the Frequency property to specify how often the task will run.

The Frequency property is an integer value. Use one of the following three integer values to specify how the task reschedules.

Value	Meaning	Description
0	Once	The task will run once, and become inactive afterwards.
1	Every	After running, the task will reschedule itself based on the values on IntervalType, Interval, RunLate and ScheduleLate properties).
2	Manual	Will run only when explicitly told to do so, ignoring the schedule.

Interval Property

CurrentTask2 Example

Specifies the scalar at which to reschedule the task. Used in conjunction with the IntervalType.

Declaration

Dim Interval As Integer

Description

Set Interval to the integer value specifying the scalar amount to use with the IntervalType. For example, if IntervalType is set to “3” (for “Weeks”), setting the Interval to “2” will cause the task to reschedule every 2 weeks.

IntervalType Property

CurrentTask2 Example

Specifies the interval at which the task will be rescheduled.

Declaration

Dim IntervalType As Integer

Description

Use the IntervalType property to specify the denomination at which the task is to be rescheduled. Use in conjunction with the Frequency property set to "Every" (integer value 1).

Set IntervalType to one of the following integer values to specify how the task is rescheduled:

Value	Meaning	Description
0	Minute	Schedules the task every x minutes
1	Hour	Schedules the task every x hours
2	Day	Schedules the task every x hours
3	Week	Schedules the task every x weeks
4	Weekday	Schedules the task every x weekdays
5	Weekend	Schedules the task every x weekend
6	Bi-week	Schedules the task every x bi-weeks (i.e. every 2(x) weeks)
7	Month	Schedules the task every x months
8	Quarter	Schedules the task every x quarters (i.e., every 4(x) months)
9	Year	Schedules the task every x years
10	Seconds	Schedules the task every x seconds

where x is the interval specified by the Interval property.

RunLate Property

CurrentTask2 Example

Controls run action when the task is overdue.

Declaration
Dim RunLate As Integer

Description
Set RunLate to one of the following integer values to specify how the task should react to being late:

Value	Meaning	Description
0	Immediately	The task is run immediately when the task becomes late.
1	Don't Run	The task is not run at all.
2	Prompt	A dialog box appears, asking whether or not to run the task.

Use with ScheduleLate to specify how the task is reschedule when the task is late.

ScheduleLate Property

CurrentTask2 Example

Controls reschedule when the task is overdue.

Declaration

Dim ScheduleLate As Integer

Description

Set ScheduleLate to one of the following integer values to specify how the task should be reschedule when the task is late:

Value	Meaning	Description
0	Relative to original date/time	The task is rescheduled relative to the date and time the task was set to launch. (For example, if the task was scheduled to launch at 10:51, and reschedule every 2 hours, but actually launches at 11:01, the task will reschedule to 12:51.)
1	Relative to launch date/time	The task is scheduled relative to when the task was launched. (For example, if the task was scheduled to launch at 9:51, and reschedule every 2 hours, but actually launches at 10:01, the task will reschedule to 12:01.)
2	Don't reschedule	The task will not reschedule itself, and instead set itself to an inactive state.

Active Property

CurrentTask2 Example

Specifies whether or not the task is active.

Declaration

Dim Active As Integer

Description

An active task is one that responds to a trigger. A task set to launch on a schedule or hotkey or any other trigger will not trigger if it is set to an inactive state. Inactive tasks, however, can still be launched manually through the AutoMate™ Configuration Manager.

Setting the Active property of a task while the task is executing an AutoMate™ BASIC script will stop the task from running any steps after a SCRIPT step.

Set the Active property to one of the following integer values to specify its active state:

Value	Meaning	Description
0	Inactive	The task is inactive (or stops, if it is currently running).
1	Active	The task is active and responds to triggers.

TaskTriggers Property

CurrentTask2 Example

Specifies the triggers that will launch the task.

Declaration

Dim TaskTriggers As Integer

Description

The TaskTriggers property is an integer value which specifies which triggers will launch the task. To set the triggers, add the following values together from the table below:

Value	Trigger	Associated Properties
0	No Trigger	None; the task is launched manually only
1	Scheduled	LaunchDate, LaunchTime, Frequency, IntervalType, Interval, RunLate, ScheduleLate
2	Windows Events	Events
4	Wait for a Window	WindowName
8	Hotkey	Hotkey

For example, to set the task to launch on a schedule and a hotkey, the value of the TaskTriggers property would be 9 (1 + 8 = 9).

Hotkey Property

CurrentTask2 Example

Specifies the hotkey that is used with the hotkey trigger to launch the task.

Declaration

Dim Hotkey As Variant

Description

Set the Hotkey property to the hotkey to be used to launch the task when one of the triggers is the Hotkey trigger.

The Hotkey text must be specified in AutoMate's "Send Keystrokes" format:

Modifier	Character to Use
ALT	^
CTRL	%
SHIFT	&

For example, to use Ctrl-Alt H for a hotkey, the property should be set to %^H.

Remarks

Please note that the characters used to specify a hotkey are different than the characters used to send the specific key. In other words, in AutoMate and VBA "^c" will **send** CTRL + c. If you set the **hotkey** to "^c" it will be the key combination ALT + c.

WindowName Property

CurrentTask2 Example

Specifies what window to wait for when the Wait For Window trigger is specified for the task.

Declaration
Dim WindowName As Variant

Description
Set the WindowName property to the full string of the window to wait for to launch the task.

StepCount Property

CurrentTask2 READ ONLY Example

The number of steps in the task.

Declaration

Dim StepCount As Integer

Description

Read the StepCount property to find out the number of steps in the current task.

Details Property

CurrentTask2 Example

Stores the details of the task in one continuous string.

Declaration

Dim Details As Variant

Description

The Details property stores the details of the task. The return value is a continuous string, with individual lines separated by a CR-LF combination.

FrequencyString Property

CurrentTask2 READ ONLY Example

Contains the frequency and scheduling information about the task in a readable format.

Declaration

Dim FrequencyString As Variant

Description

The FrequencyString property combines the scheduling properties of the task and formats it into a sentence form for easier reading.

Events Property

CurrentTask2 Example

Specifies the window event trigger that will launch the task. Use in conjunction with the "Windows Events" value added to the TaskTriggers property of the task.

Declaration

Dim Events As Integer

Description

The Events property is an integer value that specifies which windows' system events will launch the task. To set the window event triggers, add the following values together from the table below:

Value	Trigger	Associated Properties
0	None	No Windows Events triggers.
1	System Time	System time has changed (NOTE: Windows will trigger this event each time you click on the System Calendar, regardless of whether or not you click "OK", "Apply" or "Cancel" when setting the system time.
2	Low Memory	The system is getting low on memory.
4	System Device	A system device has been added, removed or its properties has changed.
8	Display Mode	The display mode (resolution or driver) has changed.
16	Color Palette	The color palette has changed.
32	Spooler	A print job has been added or removed from a printer queue.
64	Logout	A user is about to log off the workstation
128	Screensaver	Occurs just before the screensaver starts

For example, to set the task to launch on a system time change and a display mode change, the value of the property would be 9 (1 + 8 = 9).

GetTaskStep Method

CurrentTask2 Example

Retrieves the entire step from the text at the step number specified.

Declaration

function GetTaskStep(StepNo As Integer) As Variant

Parameters

StepNo

The step number, as an integer, to be retrieved.

Return Values

The function returns a string variant with the text of the step at step number *StepNo*.

Description

The GetTaskStep method returns the entire step string at the step number *StepNo* as it appears in the step builder. Use the return value is calls to GetStepCommand and GetStepParameter to parse through the step and extract the step's command and parameters.

GetStepCommand Method

CurrentTask2 Example

Retrieves the command portion of the step at the step number specified.

Declaration

function GetStepCommand(StepNo As Integer) As Variant

Parameters

StepNo

The step number, as an integer, of the command to be retrieved.

Return Values

The function returns a string variant with the text of the command at step number *StepNo*.

Description

The GetStepCommand method returns the command portion of the step at step number *StepNo*.

GetStepParamCount Method

CurrentTask2 Example

Retrieves the number of parameters at the step number specified.

Declaration

function GetStepParamCount(StepNo As Integer) As Integer

Parameters

StepNo

The step number, as an integer, of the parameter total to be calculated.

Return Values

The function returns the number of parameters at the step number.

Description

The GetStepParamCount method returns the number of parameters present or required at the step number specified. Depending on the step command, the number of parameters may change. Use GetStepParamCount to calculate how many parameters are required to satisfy the current command at that step number.

SetTaskStep Method

CurrentTask2 Example

Modifies a step to the specified text.

Declaration

procedure SetTaskStep(StepNo As Integer,
  Value As Variant)

Parameters

StepNo

The step number to be modified.

Value

The step's new text.

Description

The SetTaskStep is used to change the current value of a step in a task. Use SetTaskStep to change the step at *StepNo* to a different command.

It is the caller's responsibility to ensure the text passed to SetTaskStep is properly formatted and contains a valid command and parameters for that command. Setting a task step to an improper value could cause the task to fail in subsequent runs.

GetStepParameter Method

CurrentTask2 Example

Retrieves the parameter at the step and position specified.

Declaration

function GetStepParameter(StepNo As Integer,
Parameter As Integer) As Variant

Parameters

StepNo

The step number, as an integer, of where the parameters are to come from.

Parameter

The parameter number, starting at "0", of the parameter to retrieve.

Return Values

The function returns the parameter of *StepNo* at position *Parameter.*

Description

The GetStepParameter method is used to retrieve a specific parameter from an existing step of a task. For example, consider the following step:

MESSAGE: "This is a message box", 1

Calling GetStepParameter (0,1) would return the value "1", since it is the second parameter of step 0 (the only existing step in our example).

AppendTaskStep Method

CurrentTask2 Example

Adds a new step to the end of existing task.

Declaration

procedure AppendTaskStep(Action As Variant)

Parameters

Action

The step text to be added to the end of the existing task.

Description

The AppendTaskStep adds the text *Action* to the end of the currently executing task. The *Action* parameter must be a properly formatted step as it would appear in the Step Builder, and consist of a command and its associated parameters.

It is the responsibility of the method caller to ensure the *Action* parameter is properly formatted. Adding an invalid step to an existing task could cause it to fail on subsequent calls.

New steps added to the task do not take effect until next time the task is run.

InsertTaskStep Method

CurrentTask2 Example

Inserts a new step before the position specified.

Declaration

procedure InsertTaskStep(Before As Integer,
  Action As Variant)

Parameters

Before

The step number where the step is to be inserted.

Action

The step text to be inserted.

Description

The InsertTaskStep method inserts the text *Action* at the position specified by *Before*. The *Action* parameter must be a properly formatted step as it would appear in the Step Builder, and consist of a command and its associated parameters.

It is the responsibility of the method caller to ensure the *Action* parameter is properly formatted. Adding an invalid step to an existing task could cause it to fail on subsequent calls.

New steps added to the task do not take effect until next time the task is run.

DeleteTaskStep Method

CurrentTask2 Example

Deletes the step at the step number specified.

Declaration
procedure DeleteTaskStep(StepNo As Integer)

Parameters
StepNo
The step number to be deleted.

Description
The DeleteTaskStep method deletes the step at step number *StepNo*.

Use DeleteTaskStep to remove unwanted steps from the currently executing task.

Changes to the task do not take effect until the next time the task is run.

GetStepStatus Method

CurrentTask2 Example

Returns if the step is active or not.

Declaration

function GetStepStatus(StepNo As Integer) As Integer

Parameters

StepNo

The step number to be checked.

Return Value

Returns “0” if the step is going to be skipped or “1” if it is executable.

Description

The GetStepStatus method returns “0” if the step at *StepNo* is marked to be skipped or not.

SetStepStatus Method

CurrentTask2 Example

Sets the active flag for the specified step.

Declaration

Function SetStepStatus (StepNo As Integer, ActiveState As Integer)

Description

Specifies whether or not the step specified by StepNo is active or not. Setting the status to “0” means the step is disabled (i.e., it will be skipped when the task is executed); setting it to “1” means the step is enabled.

ClearSteps Method

CurrentTask2 Example

Clears all the steps in the task.

Declaration
Sub ClearSteps

Description
Deletes all the steps from the task, and sets the StepCount to 0.

SaveToFileEx Method

CurrentTask2 Example

Saves the task to a file.

Declaration

Sub SaveToFileEx (FileName As Variant)

Description

Use SaveToFileEx to export the task new STFF file format. This file can then be imported into another AutoMate™ installation through AutoMate's Import Task file command or by using the LoadFromFileEx method of the CurrentTask2 object.

LoadFromFileEx Method

CurrentTask2 Example

Loads the task data from a file.

Declaration

Sub LoadFromFileEx (FileName As Variant, ExtractDeployables As Integer)

Description

Use LoadFromFileEx to import a task specified in the FileName parameter into the task. The task to be loaded must have previously been saved in STFF format by using the SaveToFileEx method or by previously exporting the task to be loaded. The task's information is overwritten by the information from the task file.

LastRASErrorCode Property

AutoMate2 READ ONLY Example

Contains the error code of the last RAS command attempted.

Declaration

Dim LastRASErrorCode As Integer

Description

The LastRASErrorCode property contains the integer error code of the last RAS command attempted in the script. Use LastRASErrorCode to find out why a particular RAS command, such as RASConnect or RASDisconnect, failed.

LastRASErrorCode returns "0" if the last RAS command was successful; otherwise, it returns a non-zero value.

LastRASErrorText Property

AutoMate2 READ ONLY Example

Contains the error text of the last RAS command attempted.

Declaration

Dim LastRASErrorText As Variant

Description

The LastRASErrorText property contains a textual explanation of the error code of the last RAS command attempted.

RASHandle Property

AutoMate2 READ ONLY Example

Contains the RAS handle to a previously established dial-up connection.

Declaration

Dim RASHandle As Integer

Description

The RASHandle property contains a handle to an active dial-up connection established using the RASConnect method. The property is “0” if no active dial-up connection is present for this task.

SystemDir Property

AutoMate2 READ ONLY Example

Contains the System directory of the system.

Declaration

Dim SystemDir As Variant

Description

The SystemDir property contains the fully qualified path to the System directory of the current installation.

TaskCount Property

AutoMate2 READ ONLY Example

Contains the number of tasks in the current task list.

Declaration

Dim TaskCount As Integer

Description

Read TaskCount to obtain the total number of tasks in the task list currently in use by AutoMate™.

TaskList Property

AutoMate2 READ ONLY Example

Contains the path to the task list in use by AutoMate™.

Declaration

Dim TaskList As Variant

Description

Use the TaskList property to obtain the fully qualified path to the task list file AutoMate™ is currently using. A task list is a collection of tasks.

WindowsDir Property

AutoMate2 READ ONLY Example

Contains the Windows directory of the system.

Declaration

Dim WindowsDir As Variant

Description

The WindowsDir property contains the fully qualified path to the Windows directory of the current installation.

AutoMate2

Properties Methods

The AutoMate2 object encapsulates the global properties of the AutoMate™ and provides methods to manipulate the execution of a task or tasks.

Description

The AutoMate2 object provides properties that provide information about the current state of AutoMate™, the current task list, and any running tasks. It also provides methods that allow you the change the running and execution flow of the current task, such as waiting for a window to appear or running another task.

AutoMate2 properties

AutoMate2 Legend

- SystemDir
- TaskCount
- TaskList
- WindowsDir

AutoMate2 methods

AutoMate2

- FindWindowByTitle
- FindWindowByClass
- FindWindowContaining

FindWindowByTitle Method

AutoMate2 Example

Returns the handle of the window that matches a specified windowk title.

Declaration

Function FindWindowByTitle (WindowTitle As Variant) As Integer

Description

Use FindWindowByTitle to obtain a handle to the window with the title specified in WindowTitle. The window must be open for this method to be successful. The method returns “0” if the window is not found, or it returns a non-zero value (the window handle) if the window is located.

FindWindowByClass Method

Returns the handle of the window that matches a specified class name.

Declaration

Function FindWindowByClass (WindowClass As Variant) As Integer

Description

Use FindWindowByClass to obtain a handle to the window with the class name specified in WindowClass. The window must be open for this method to be successful. If multiple windows with the specified class name are open, FindWindowByClass will return the handle to the first window it enumerates. The method returns “0” if the window is not found, or it returns a non-zero value (the window handle) if the window is located.

FindWindowContaining Method

Returns the handle of the window that contains a specified text.

Declaration

Function FindWindowContaining (TextToFind As Variant, WindowClass As Variant) As Integer

Description

Use FindWindowContaining to obtain a handle to the window that contains the text specified by TextToFind. The window must be open for this method to be successful. If multiple windows that contain the specified text are open, the method will return the handle to the first window it finds. The method returns "0" if there are no windows with the text, or it returns a non-zero value (the window handle) if the window is located.

The WindowClass parameter is only needed if there is a need to specify the Window class that the window belongs to. Normally this should just be left as empty quotes.

RASConnect Method

AutoMate2 Example

Attempts a dial-up connection using a specified phonebook entry and returns a handle to the connection on success.

Declaration

Function RASConnect (PhonebookEntry As Variant) As Integer

Description

Use RASConnect to attempt to establish a dial-up connection using the phonebook entry name supplied in PhonebookEntry parameter.

On success, RASConnect returns a handle to a RAS object. RASConnect returns “0” if the connection fails. On failure, use RASLastErrorCode and RASLastErrorText for information about why the method failed.

RASDisconnectEx Method

AutoMate2 Example

Attempts to disconnect a previously established dial-up connection.

Declaration

Function RASDisconnectEx (PhonebookEntry As Variant) As Integer

Description

Use RASDisconnectEx to disconnect an active dial-up connection. The connection does not have to have been established by AutoMate™. Pass the name of the connection to be disconnected in the PhonebookEntry variable.

RunTask Method

AutoMate2 Example

Runs another task.

Declaration

Sub RunTask (TaskName As Variant)

Description

Use RunTask to start another task. The task specified in TaskName must be in the current TaskList (use the TaskList property to obtain the path to the current task list in use by the AutoMate object). The task run by RunTask executes asynchronously with other tasks (i.e., the execution of the script using the RunTask method will not stop).

Active Example

The following example sets the task to inactive. This will prevent the task from executing the rest of the task (but *not* the next line of the script).

```
Sub Main
    CurrentTask2.Active = 0
End Sub
```

Details Example

The following example displays the task's details in a message box:

```
Sub Main
    MsgBox CurrentTask2.Details
End Sub
```

This example sets the task's details to "This task is responsible for rebooting the server at 5AM each morning."

```
Sub Main
    CurrentTask2.Details = "This task is responsible for rebooting the server at 5AM each morning."
End Sub
```

This example sets the task's details to "You love AutoMate™."

```
Sub Main
    CurrentTask2.Details = "You love AutoMate™."
End Sub
```

Events Example

The following example sets the current task to trigger only when the spooler changes (i.e., when a print job is added, removed or completed from the printer queue) or the system date/time changes

```
Sub Main
    ‘ Set the task to trigger on only Windows Events
    CurrentTask2.TaskTriggers = 2

    ‘ Set the window events to spooler or time change
    CurrentTask2.Events = 33
End Sub
```

Frequency Example

The following example will set the current task to run only when manually launched.

```
Sub Main
    CurrentTask2.Frequency = 2
End Sub
```

This example sets the current task to run itself every 3 days starting on September 15, 1999 at 11:29AM. When the task runs, it will reschedule itself relative to the time it was originally set to launch. If the task is late, it will not run at all.

```
Sub Main
    ‘ Set the task to “every”
    CurrentTask2.Frequency = 1

    ‘ Start launching on September 15, 1999
    CurrentTask2.LaunchDate = “09/15/1999”

    ‘ At 11:59 AM
    CurrentTask2.LaunchTime = “11:29 AM”

    ‘ Set the interval type to “days”
    CurrentTask2.IntervalType = 2

    ‘ Set the interval to 3, for every three days
    CurrentTask2.Interval = 3

    ‘ If we are late, don’t run at all
    CurrentTask2.RunLate = 1

    ‘ Reschedule the task relative to the launch date/time if we are late
    CurrentTask2.ScheduleLate = 0
End Sub
```

FrequencyString Example

The following example displays the current task's scheduling parameters in a friendly, textual form.

```
Sub Main
    MsgBox CurrentTask2.FrequencyString
End Sub
```

Hotkey Example

The following example sets the task to trigger on a hotkey and sets the hotkey to Ctrl-S.

```
Sub Main
    ‘ Set the hotkey to Ctrl-S
    CurrentTask2.Hotkey = “%s”

    ‘ Set to trigger on a hotkey
    CurrentTask2.TaskTriggers = 8
End Sub
```

Interval Example

This example sets the current task to run itself every 3 days starting on September 15, 1999 at 11:29AM. When the task runs, it will reschedule itself relative to the time it was originally set to launch. If the task is late, it will not run at all.

```
Sub Main
    ‘ Set the task to “every”
    CurrentTask2.Frequency = 1

    ‘ Start launching on September 15, 1999
    CurrentTask2.LaunchDate = “09/15/1999”

    ‘ At 11:59 AM
    CurrentTask2.LaunchTime = “11:29 AM”

    ‘ Set the interval type to “days”
    CurrentTask2.IntervalType = 2

    ‘ Set the interval to 3, for every three days
    CurrentTask2.Interval = 3

    ‘ If we are late, don’t run at all
    CurrentTask2.RunLate = 1

    ‘ Reschedule the task relative to the launch date/time if we are late
    CurrentTask2.ScheduleLate = 0
End Sub
```

IntervalType Example

This example sets the current task to run itself every 3 days starting on September 15, 1999 at 11:29AM. When the task runs, it will reschedule itself relative to the time it was originally set to launch. If the task is late, it will not run at all.

```
Sub Main
    ‘ Set the task to “every”
    CurrentTask2.Frequency = 1

    ‘ Start launching on September 15, 1999
    CurrentTask2.LaunchDate = “09/15/1999”

    ‘ At 11:59 AM
    CurrentTask2.LaunchTime = “11:29 AM”

    ‘ Set the interval type to “days”
    CurrentTask2.IntervalType = 2

    ‘ Set the interval to 3, for every three days
    CurrentTask2.Interval = 3

    ‘ If we are late, don’t run at all
    CurrentTask2.RunLate = 1

    ‘ Reschedule the task relative to the launch date/time if we are late
    CurrentTask2.ScheduleLate = 0
End Sub
```

LaunchDate Example

This example sets the current task to run itself every 3 days starting on September 15, 1999 at 11:29AM. When the task runs, it will reschedule itself relative to the time it was originally set to launch. If the task is late, it will not run at all.

```
Sub Main
    ‘ Set the task to “every”
    CurrentTask2.Frequency = 1

    ‘ Start launching on September 15, 1999
    CurrentTask2.LaunchDate = “09/15/1999”

    ‘ At 11:59 AM
    CurrentTask2.LaunchTime = “11:29 AM”

    ‘ Set the interval type to “days”
    CurrentTask2.IntervalType = 2

    ‘ Set the interval to 3, for every three days
    CurrentTask2.Interval = 3

    ‘ If we are late, don’t run at all
    CurrentTask2.RunLate = 1

    ‘ Reschedule the task relative to the launch date/time if we are late
    CurrentTask2.ScheduleLate = 0
End Sub
```

LaunchTime Example

This example sets the current task to run itself every 3 days starting on September 15, 1999 at 11:29AM. When the task runs, it will reschedule itself relative to the time it was originally set to launch. If the task is late, it will not run at all.

```
Sub Main
    ‘ Set the task to “every”
    CurrentTask2.Frequency = 1

    ‘ Start launching on September 15, 1999
    CurrentTask2.LaunchDate = “09/15/1999”

    ‘ At 11:59 AM
    CurrentTask2.LaunchTime = “11:29 AM”

    ‘ Set the interval type to “days”
    CurrentTask2.IntervalType = 2

    ‘ Set the interval to 3, for every three days
    CurrentTask2.Interval = 3

    ‘ If we are late, don’t run at all
    CurrentTask2.RunLate = 1

    ‘ Reschedule the task relative to the launch date/time if we are late
    CurrentTask2.ScheduleLate = 0
End Sub
```

RunLate Example

This example sets the current task to run itself every 3 days starting on September 15, 1999 at 11:29AM. When the task runs, it will reschedule itself relative to the time it was originally set to launch. If the task is late, it will not run at all.

```
Sub Main
    ‘ Set the task to “every”
    CurrentTask2.Frequency = 1

    ‘ Start launching on September 15, 1999
    CurrentTask2.LaunchDate = “09/15/1999”

    ‘ At 11:59 AM
    CurrentTask2.LaunchTime = “11:29 AM”

    ‘ Set the interval type to “days”
    CurrentTask2.IntervalType = 2

    ‘ Set the interval to 3, for every three days
    CurrentTask2.Interval = 3

    ‘ If we are late, don’t run at all
    CurrentTask2.RunLate = 1

    ‘ Reschedule the task relative to the launch date/time if we are late
    CurrentTask2.ScheduleLate = 0
End Sub
```

ScheduleLate Example

This example sets the current task to run itself every 3 days starting on September 15, 1999 at 11:29AM. When the task runs, it will reschedule itself relative to the time it was originally set to launch. If the task is late, it will not run at all.

```
Sub Main
    ‘ Set the task to “every”
    CurrentTask2.Frequency = 1

    ‘ Start launching on September 15, 1999
    CurrentTask2.LaunchDate = “09/15/1999”

    ‘ At 11:59 AM
    CurrentTask2.LaunchTime = “11:29 AM”

    ‘ Set the interval type to “days”
    CurrentTask2.IntervalType = 2

    ‘ Set the interval to 3, for every three days
    CurrentTask2.Interval = 3

    ‘ If we are late, don’t run at all
    CurrentTask2.RunLate = 1

    ‘ Reschedule the task relative to the launch date/time if we are late
    CurrentTask2.ScheduleLate = 0
End Sub
```

StepCount Example

This example displays the number of steps in the current task in a message box.

```
Sub Main
    MsgBox CurrentTask2.StepCount
 End Sub
```

TaskTriggers Example

The following example sets the task to trigger on a hotkey and sets the hotkey to Ctrl-S.

```
Sub Main
    ‘ Set the hotkey to Ctrl-S
    CurrentTask2.Hotkey = ‘‘%s’’

    ‘ Set to trigger on a hotkey
    CurrentTask2.TaskTriggers = 8
End Sub
```

TaskName Example

The following example displays the name of the currently running task in a message box.

```
Sub Main
    MsgBox CurrentTask2.TaskName
End Sub
```

This example displays the original name of the running task, changes it to “New Task Name”, and displays the new task name in a message box.

```
Sub Main
    ‘ Display the task’s current name
        MsgBox CurrentTask2.TaskName

    ‘ Change it to something else
    CurrentTask2.TaskName = “New Task Name”

    ‘ Now display the task’s name again
    MsgBox CurrentTask2.TaskName
 End Sub
```

WindowName Example

The following example sets the current task to trigger only when the window with the title of “Untitled – Notepad” appears.

```
Sub Main
    ‘ Turn on the “Window Watcher” trigger
    CurrentTask2.TaskTriggers = 4

    ‘ Set the only property of the “Window Watcher”
    CurrentTask2.WindowName = “Untitled – Notepad”
End Sub
```

AppendTaskStep Example

The following example will append a step to the current task that displays a message box with the words “This is AutoMate™.” and make it pop to the front.

```
Sub Main
    CurrentTask2.AppendTaskStep (“MESSAGE: “ + Chr$(13) + “ This is AutoMate™. “ + Chr$(13) + “,1”
End Sub
```

ClearSteps Example

The following example will erase all the steps in the current task, which consequently sets the TaskCount property to zero.

```
Sub Main
    CurrentTask2.ClearSteps
End Sub
```

DeleteTaskStep Example

The following example deletes the second step of the currently running task (note that, since AutoMate™ uses a zero based index for steps, step two in the step builder is actually index 1 in the scripting language).

```
Sub Main
    CurrentTask2.DeleteTaskStep (2)
    MsgBox “Step 2 deleted.”
End Sub
```

GetStepCommand Example

The following example returns the command of step 2 into the variable varStepCommand, then displays it in a message box.

```
Sub Main
    Dim varStepCommand As Variant

    varStepCommand = CurrentTask2.GetStepCommand (1)
    MsgBox varStepCommand
End Sub
```

GetStepParameter Example

The following example returns the number of parameters in step 2 into the variable iNumOfSteps. It then goes into a loop and displays each parameter, one by one, in a message box.

```
Sub Main
    Dim iNumOfSteps As Integer
    Dim varParameter As Variant

    iNumOfSteps = CurrentTask2.GetStepParamCount (1)

    for iIndex = 0 to iNumOfSteps
        varParameter = CurrentTask2.GetStepParameter (1, iIndex)
        MsgBox varParameter
    next iIndex
 End Sub
```

GetStepParamCount Example

The following example returns the number of parameters in step 2 into the variable iNumOfSteps. It then goes into a loop and displays each parameter, one by one, in a message box.

```
Sub Main
    Dim iNumOfSteps As Integer
    Dim varParameter As Variant

    iNumOfSteps = CurrentTask2.GetStepParamCount (1)

    for iIndex = 0 to iNumOfSteps
        varParameter = CurrentTask2.GetStepParameter (1, iIndex)
        MsgBox varParameter
    next iIndex
 End Sub
```

GetStepStatus Example

The following example displays a message box with the text “Step will be skipped” if step 2 is set to be skipped during execution, and “Step will be executed” otherwise.

```
Sub Main
    Dim iStepStatus As Integer

    iStepStatus = CurrentTask2.GetStepStatus (1)

    if iStepStatus = 0 then
        MsgBox “Step will be skipped”
    else
        MsgBox “Step will be executed”
    End if
End Sub
```

GetTaskStep Example

The following example will display step 2 of the task in a message box.

```
Sub Main
    Dim varStep As Variant

    varStep = CurrentTask2.GetTaskStep (1)

    MsgBox varStep
End Sub
```

InsertTaskStep Example

The following example will add a step as the second step of the task that displays a message box with the words “This is AutoMate™.” and make it pop to the front.

```
Sub Main
    CurrentTask2.InsertTaskStep (2, “MESSAGE: “ + Chr$(13) + “ This is AutoMate™. “ + Chr$(13) +
“,1”
End Sub
```

LoadFromFileEx Example

This example loads the STFF found at c:\mytasks\example.amt", overwriting the contents of the current task with the steps and properties of the example.amt task file.

```
Sub Main
    CurrentTask2.LoadFromFileEx ("c:\mytasks\example.amt", 0)
End Sub
```

SaveToFileEx Example

The example saves the properties and steps of the current task as an STFF file called “example.amt”, and places it in the “c:\mytasks\” directory. The file can then be loaded by another script using the CurrentTask2.LoadFromFile method.

```
Sub Main
    CurrentTask2.SaveToFileEx (“c:\mytasks\example.amt”, 0)
End Sub
```

SetStepStatus Example

The following example sets all the steps in the current task to “active”, so that no steps are skipped the next time the task is executed.

```
Sub Main
    Dim iTotalSteps As Integer

    iTotalSteps = CurrentTask2.StepCount

    For iIndex = 0 To iTotalSteps
        CurrentTask2.SetStepStatus (iTotalSteps, 1)
    Next iIndex
End Sub
```

SetTaskStep Example

This example sets step 2 to display a message box with the text “This is AutoMate™”, overwriting whatever the contents of step 2 was before.

```
Sub Main
    CurrentTask2.SetTaskStep (1, “MESSAGE: “ + Chr$(13) + “ This is AutoMate™. “ + Chr$(13) + “,1”
End Sub
```

LastRASErrorCode Example

This example demonstrates the use of a number of the AMOLE's methods and properties: RASConnect, RASDisconnectEx, RASHandle, LastRASErrorCode, and LastRASErrorText.

The example attempts to dial the connection named "MyConnection." If the connection succeeds, the script waits for the user to click the Ok button on the message box that appears, then disconnects. If the connection fails, a message box with the error code and error text is displayed.

```
Sub Main
   Dim iSuccess As Integer

   iSuccess = AutoMate2.RASConnect ("MyConnection")

   If iSuccess > 0 Then
      MsgBox "Connection successful with RAS handle " + AutoMate2.RASHandle + ". Click Ok to
disconnect."
      AutoMate2.RASDisconnectEx ("MyConnection")
   Else
      MsgBox "Connection failed with code " + AutoMate2.LastRASErrorCode + " and message: " +
AutoMate2.LastRASErrorText
   End If
End Sub
```

LastRASErrorText Example

This example demonstrates the use of a number of the AMOLE's methods and properties: RASConnect, RASDisconnectEx, RASHandle, LastRASErrorCode, and LastRASErrorText.

The example attempts to dial the connection named "MyConnection." If the connection succeeds, the script waits for the user to click the Ok button on the message box that appears, then disconnects. If the connection fails, a message box with the error code and error text is displayed.

```
Sub Main
   Dim iSuccess As Integer

   iSuccess = AutoMate2.RASConnect ("MyConnection")

   If iSuccess > 0 Then
      MsgBox "Connection successful with RAS handle " + AutoMate2.RASHandle + ". Click Ok to
disconnect."
      AutoMate2.RASDisconnectEx ("MyConnection")
   Else
      MsgBox "Connection failed with code " + AutoMate2.LastRASErrorCode + " and message: " +
AutoMate2.LastRASErrorText
   End If
End Sub
```

RASHandle Example

This example demonstrates the use of a number of the AMOLE's methods and properties: RASConnect, RASDisconnectEx, RASHandle, LastRASErrorCode, and LastRASErrorText.

The example attempts to dial the connection named "MyConnection." If the connection succeeds, the script waits for the user to click the Ok button on the message box that appears, then disconnects. If the connection fails, a message box with the error code and error text is displayed.

```
Sub Main
   Dim iSuccess As Integer

   iSuccess = AutoMate2.RASConnect ("MyConnection")

   If iSuccess > 0 Then
      MsgBox "Connection successful with RAS handle " + AutoMate2.RASHandle + ". Click Ok to
disconnect."
      AutoMate2.RASDisconnectEx ("MyConnection")
   Else
      MsgBox "Connection failed with code " + AutoMate2.LastRASErrorCode + " and message: " +
AutoMate2.LastRASErrorText
   End If
End Sub
```

SystemDir Example

This example displays the system directory in a message box.

```
Sub Main
    MsgBox AutoMate2.SystemDir
End Sub
```

TaskCount Example

The following example displays the path to the current task list and the total number of tasks in it.

```
Sub Main
    MsgBox (AutoMate2.TaskList + " has " + AutoMate2.TaskCount + " task(s).")
End Sub
```

TaskList Example

The following example displays the path to the current task list and the total number of tasks in it.

```
Sub Main
    MsgBox (AutoMate2.TaskList + " has " + AutoMate2.TaskCount + " task(s).")
End Sub
```

WindowsDir Example

This example displays the windows directory in a message box.

```
Sub Main
    MsgBox AutoMate2.WindowsDir
End Sub
```

FindWindowByTitle Example

The following example attempts to find a window with the title bar text “Untitled – Notepad.” If it is found, a message box is displayed with the text “Notepad is open”, otherwise a message box with the text “Notepad is not open” is displayed.

```
Sub Main
    If AutoMate2.FindWindowbyTitle (“Untitled – Notepad”) > 0
        MsgBox “Notepad is open”
    Else
        MsgBox “Notepad is not open”
End Sub
```

FindWindowByClass Example

The following example attempts to find a window with the classname of “Notepad.” If it is found, a message box is displayed with the text “Notepad class exists”, otherwise a message box with the text “Notepad class does not exist” is displayed.

```
Sub Main
    If AutoMate2.FindWindowbyClass (“Notepad”) > 0
        MsgBox “Notepad class exists”
    Else
        MsgBox “Notepad class does not exist”
End Sub
```

FindWindowContaining Example

The following example tries to find a window that contains the text “Access violation.” If a window is found, a message box is displayed with the text “Window found.” Otherwise, a message box with the text “Window NOT found” is displayed.

```
Sub Main
   If AutoMate2.FindWindowContaining (“Access violation”, “”) > 0
        MsgBox “Window found”
   Else
        MsgBox “Window NOT found”
   End If
End Sub
```

RASConnect Example

This example demonstrates the use of a number of the AMOLE's methods and properties: RASConnect, RASDisconnectEx, RASHandle, LastRASErrorCode, and LastRASErrorText.

The example attempts to dial the connection named "MyConnection." If the connection succeeds, the script waits for the user to click the Ok button on the message box that appears, then disconnects. If the connection fails, a message box with the error code and error text is displayed.

```
Sub Main
   Dim iSuccess As Integer

   iSuccess = AutoMate2.RASConnect ("MyConnection")

   If iSuccess > 0 Then
      MsgBox "Connection successful with RAS handle " + AutoMate2.RASHandle + ". Click Ok to
disconnect."
      AutoMate2.RASDisconnectEx ("MyConnection")
   Else
      MsgBox "Connection failed with code " + AutoMate2.LastRASErrorCode + " and message: " +
AutoMate2.LastRASErrorText
   End If
End Sub
```

RASDisconnectEx Example

This example demonstrates the use of a number of the AMOLE's methods and properties: RASConnect, RASDisconnectEx, RASHandle, LastRASErrorCode, and LastRASErrorText.

The example attempts to dial the connection named "MyConnection." If the connection succeeds, the script waits for the user to click the Ok button on the message box that appears, then disconnects. If the connection fails, a message box with the error code and error text is displayed.

```
Sub Main
   Dim iSuccess As Integer

   iSuccess = AutoMate2.RASConnect ("MyConnection")

   If iSuccess > 0 Then
      MsgBox "Connection successful with RAS handle " + AutoMate2.RASHandle + ". Click Ok to
disconnect."
      AutoMate2.RASDisconnectEx ("MyConnection")
   Else
      MsgBox "Connection failed with code " + AutoMate2.LastRASErrorCode + " and message: " +
AutoMate2.LastRASErrorText
   End If
End Sub
```

RunTask Example

This example attempts to run a task called “Focus Notepad.”

```
Sub Main
    AutoMate2.RunTask (“Focus Notepad”)
End Sub
```

AMCopyFile Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varSourceFile As Variant
Dim varDestFile As Variant

    varSourceFile = "C:\*.txt"
    varDestFile = "C:\AnotherDir\*.bak"

    intReturn = Action2.AMCopyFile (varSourceFile, varDestFile)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

AMChangeDir Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varDirectory As Variant

    varDirectory = "c:\My Documents\"

    intReturn = Action2.AMChangeDir (varDirectory)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

AMDeleteFile Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varFileName As Variant

    varFileName = "C:\*.txt"

    intReturn = Action2.AMDeleteFile (varFilename)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

AMMakeDir Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varDirectory As Variant

    varDirectory = "c:\DirToMake"

    intReturn = Action2.AMMakeDir (varDirectory)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

AMRemoveDir Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varDirectory As Variant

    varDirectory = "c:\DirToRemove"

    intReturn = Action2.AMRemoveDir(varDirectory)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

AMRenameFile Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varOldName As Variant
Dim varNewName As Variant

    varOldName = "C:\*.txt"
    varNewName = "C:\*.bak"

    intReturn = Action2.AMRenameFile (varOldName, varNewName)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

CheckForMail Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varServer As Variant
Dim varUsername As Variant
Dim varPassword As Variant
Dim intWaitingMailAction As Integer
Dim intNoWaitingMailAction As Integer

    varServer = "mail.mailserver.com"
    varUsername = "username"
    varPassword = "password"
    intWaitingMailAction = 1
    intNoWaitingMailAction = 0

    intReturn = Action2.CheckForMail (varServer, varUsername, varPassword, intWaitingMailAction,
intNoWaitingMailAction)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

ClipboardClear Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.ClipboardClear

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

ClipboardCopy Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.ClipboardCopy

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

ClipboardCut Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.ClipboardCut

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

ClipboardPaste Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.ClipboardPaste

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

ContService Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varServiceName As Variant

    varServiceName = "ServiceToContinue"

    intReturn = Action2.ContService (varServiceName)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

CreateVar Example

TO DO: Enter topic text here!

DDECommand Example

TO DO: Enter topic text here!

DisableThisTask Example

This example checks to see if the name of the window the user is currently working with is “Untitled – Notepad.” If not, the task disables itself.

```
Sub Main
    If Action2.GetFocusedWindowName <> “Untitled – Notepad” Then
        Action2.DisableThisTask
    End If
End Sub
```

FocusWindowContaining Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowTitle As Variant
Dim intExactTitleMatch As Integer
Dim varTextToFind As Variant
Dim intExactTextMatch As Integer

    varWindowTitle = "This is the window title to look for"
    intExactTitleMatch = 0
    varTextToFind = "This is the text on the dialog window to look for"
    intExactTextMatch = 1

    intReturn = Action2.FocusWindowContaining(varWindowTitle, intExactTitleMatch, varTextToFind,
intExactTextMatch)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

FTPChangeDirectory Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

‘ The Action2 FTP methods (with the exception of
‘ FTPQuickSend and FTPQuickRetrieve may not
‘ function properly in the AutoMate™ BASIC IDE.  To
‘ more accurately use these functions, run the script from
‘ within an AutoMate™ task using the “Run a BASIC
‘ Script” action.

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer
Dim intPassiveMode As Integer
Dim varLocalFile As Variant
Dim varRemoteFile As Variant
Dim intTransType As Integer
Dim varNewDirectory As Variant

   varHost = "www.hostname.com"
   varUsername = "username"
   varPassword = "password"
   intPort = 21
   varFTPLogFile = "c:\ftp.log"
   intOverwriteLog = 1

   varLocalFile = "C:\*.txt"
   varRemoteFile = "*.bak"
   varNewDirectory = "/MyDirectory"

   Action2.FTPLogin (varHost, varUsername, varPassword, intPort, varFTPLogFile, intOverwriteLog,
intPassiveMode)
   intReturn = Action2.FTPChangeDirectory(varNewDirectory)
   Action2.FTPUpload (varLocalFile, varRemoteFile, intTransType)
   Action2.FTPLogout

   If intReturn = 0 Then
      MsgBox "Failure"
      MsgBox( Action2.GetLastError )
   ElseIf intReturn = 1 Then
```

```
        MsgBox "Success"
    End If

End Sub
```

FTPDelete Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

‘ The Action2 FTP methods (with the exception of
‘ FTPQuickSend and FTPQuickRetrieve may not
‘ function properly in the AutoMate™ BASIC IDE.  To
‘ more accurately use these functions, run the script from
‘ within an AutoMate™ task using the “Run a BASIC
‘ Script” action.

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer
Dim intPassiveMode As Integer
Dim varFTPFilename As Variant
Dim intTransType As Integer

   varHost = "www.hostname.com"
   varUsername = "username"
   varPassword = "password"
   intPort = 21
   varFTPLogFile = "c:\ftp.log"
   intOverwriteLog = 1

   varFTPFilename = "/*.txt"

   Action2.FTPLogin (varHost, varUsername, varPassword, intPort, varFTPLogFile, intOverwriteLog,
intPassiveMode)
   intReturn = Action2.FTPDelete (varFTPFilename)
   Action2.FTPLogout

   If intReturn = 0 Then
      MsgBox "Failure"
      MsgBox( Action2.GetLastError )
   ElseIf intReturn = 1 Then
      MsgBox "Success"
   End If

End Sub
```

FTPDownload Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

‘ The Action2 FTP methods (with the exception of
‘ FTPQuickSend and FTPQuickRetrieve may not
‘ function properly in the AutoMate™ BASIC IDE.  To
‘ more accurately use these functions, run the script from
‘ within an AutoMate™ task using the “Run a BASIC
‘ Script” action.

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer
Dim intPassiveMode As Integer
Dim varLocalFile As Variant
Dim varRemoteFile As Variant
Dim intTransType As Integer

   varHost = "www.hostname.com"
   varUsername = "username"
   varPassword = "password"
   intPort = 21
   varFTPLogFile = "c:\ftp.log"
   intOverwriteLog = 1

   varLocalFile = "c:\"
   varRemoteFile = "/*.txt"

   Action2.FTPLogin (varHost, varUsername, varPassword, intPort, varFTPLogFile, intOverwriteLog,
intPassiveMode)
   intReturn = Action2.FTPDownload (varRemoteFile, varLocalFile, intTransType)
   Action2.FTPLogout

   If intReturn = 0 Then
      MsgBox "Failure"
      MsgBox( Action2.GetLastError )
   ElseIf intReturn = 1 Then
      MsgBox "Success"
   End If
```

```
End Sub
```

FTPLogin Example

```
Sub Main
Dim intReturn As Integer

‘ The Action2 FTP methods (with the exception of
‘ FTPQuickSend and FTPQuickRetrieve may not
‘ function properly in the AutoMate™ BASIC IDE.  To
‘ more accurately use these functions, run the script from
‘ within an AutoMate™ task using the “Run a BASIC
‘ Script” action.

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer
Dim intPassiveMode As Integer
Dim varLocalFile As Variant
Dim varRemoteFile As Variant
Dim intTransType As Integer

    varHost = "www.hostname.com"
    varUsername = "username"
    varPassword = "password"
    intPort = 21
    varFTPLogFile = "c:\ftp.log"
    intOverwriteLog = 1

    varLocalFile = "c:\"
    varRemoteFile = "/*.txt"

    Action2.FTPLogin (varHost, varUsername, varPassword, intPort, varFTPLogFile, intOverwriteLog,
intPassiveMode)
    intReturn = Action2.FTPDownload (varRemoteFile, varLocalFile, intTransType)
    Action2.FTPLogout

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

FTPLogout Example

```
Sub Main
Dim intReturn As Integer

‘ The Action2 FTP methods (with the exception of
‘ FTPQuickSend and FTPQuickRetrieve may not
‘ function properly in the AutoMate™ BASIC IDE.  To
‘ more accurately use these functions, run the script from
‘ within an AutoMate™ task using the “Run a BASIC
‘ Script” action.

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer
Dim intPassiveMode As Integer
Dim varLocalFile As Variant
Dim varRemoteFile As Variant
Dim intTransType As Integer

    varHost = "www.hostname.com"
    varUsername = "username"
    varPassword = "password"
    intPort = 21
    varFTPLogFile = "c:\ftp.log"
    intOverwriteLog = 1

    varLocalFile = "c:\"
    varRemoteFile = "/*.txt"

    Action2.FTPLogin (varHost, varUsername, varPassword, intPort, varFTPLogFile, intOverwriteLog,
intPassiveMode)
    intReturn = Action2.FTPDownload (varRemoteFile, varLocalFile, intTransType)
    Action2.FTPLogout

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

FTPMakeDirectory Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

‘ The Action2 FTP methods (with the exception of
‘ FTPQuickSend and FTPQuickRetrieve may not
‘ function properly in the AutoMate™ BASIC IDE.  To
‘ more accurately use these functions, run the script from
‘ within an AutoMate™ task using the “Run a BASIC
‘ Script” action.

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer
Dim intPassiveMode As Integer
Dim varDirectory As Variant

   varHost = "www.hostname.com"
   varUsername = "username"
   varPassword = "password"
   intPort = 21
   varFTPLogFile = "c:\ftp.log"
   intOverwriteLog = 1

   varDirectory = "/NewDir"

   Action2.FTPLogin (varHost, varUsername, varPassword, intPort, varFTPLogFile, intOverwriteLog,
intPassiveMode)
   intReturn = Action2.FTPMakeDirectory(varDirectory)
   Action2.FTPLogout

   If intReturn = 0 Then
      MsgBox "Failure"
      MsgBox( Action2.GetLastError )
   ElseIf intReturn = 1 Then
      MsgBox "Success"
   End If

End Sub
```

FTPQuickRetrieve Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varLocalFile As Variant
Dim varRemoteFile As Variant
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer

    varHost = "www.hostname.com"
    varUserName = "username"
    varPassword = "password"
    intPort = 21
    varLocalFile = "C:\mytest.txt"
    varRemoteFile = "/mytest.txt"
    varFTPLogFile = "c:\ftp.log"
    intOverwriteLog = 1

    intReturn = Action2.FTPQuickRetrieve(varHost, varUsername, varPassword, intPort, varLocalFile,
varRemoteFile, varFTPLogFile, intOverwriteLog)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

FTPQuickSend Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varLocalFile As Variant
Dim varRemoteFile As Variant
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer

    varHost = "www.hostname.com"
    varUserName = "username"
    varPassword = "password"
    intPort = 21
    varLocalFile = "C:\mytest.txt"
    varRemoteFile = "/mytest.txt"
    varFTPLogFile = "c:\ftp.log"
    intOverwriteLog = 1

    intReturn = Action2.FTPQuickSend (varHost, varUsername, varPassword, intPort, varLocalFile,
varRemoteFile, varFTPLogFile, intOverwriteLog)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

FTPRemoveDirectory Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

‘ The Action2 FTP methods (with the exception of
‘ FTPQuickSend and FTPQuickRetrieve may not
‘ function properly in the AutoMate™ BASIC IDE.  To
‘ more accurately use these functions, run the script from
‘ within an AutoMate™ task using the “Run a BASIC
‘ Script” action.

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer
Dim intPassiveMode As Integer
Dim varDirectory As Variant

   varHost = "www.hostname.com"
   varUsername = "username"
   varPassword = "password"
   intPort = 21
   varFTPLogFile = "c:\ftp.log"
   intOverwriteLog = 1

   varDirectory = "/DirToRemove"

   Action2.FTPLogin (varHost, varUsername, varPassword, intPort, varFTPLogFile, intOverwriteLog,
intPassiveMode)
   intReturn = Action2.FTPRemoveDirectory(varDirectory)
   Action2.FTPLogout

   If intReturn = 0 Then
      MsgBox "Failure"
      MsgBox( Action2.GetLastError )
   ElseIf intReturn = 1 Then
      MsgBox "Success"
   End If

End Sub
```

FTPRename Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

‘ The Action2 FTP methods (with the exception of
‘ FTPQuickSend and FTPQuickRetrieve may not
‘ function properly in the AutoMate™ BASIC IDE.  To
‘ more accurately use these functions, run the script from
‘ within an AutoMate™ task using the “Run a BASIC
‘ Script” action.

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer
Dim intPassiveMode As Integer
Dim varFTPOldName As Variant
Dim varFTPNewName As Variant

   varHost = "www.hostname.com"
   varUsername = "username"
   varPassword = "password"
   intPort = 21
   varFTPLogFile = "c:\ftp.log"
   intOverwriteLog = 1

   varFTPOldName = "/mytest.txt"
   varFTPNewName = "newname.txt"

   Action2.FTPLogin (varHost, varUsername, varPassword, intPort, varFTPLogFile, intOverwriteLog,
intPassiveMode)
   intReturn = Action2.FTPRename(varFTPOldName, varFTPNewName)
   Action2.FTPLogout

   If intReturn = 0 Then
      MsgBox "Failure"
      MsgBox( Action2.GetLastError )
   ElseIf intReturn = 1 Then
      MsgBox "Success"
   End If

End Sub
```

FTPUpload Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

‘ The Action2 FTP methods (with the exception of
‘ FTPQuickSend and FTPQuickRetrieve may not
‘ function properly in the AutoMate™ BASIC IDE.  To
‘ more accurately use these functions, run the script from
‘ within an AutoMate™ task using the “Run a BASIC
‘ Script” action.

Dim varHost As Variant
Dim varUserName As Variant
Dim varPassword As Variant
Dim intPort As Integer
Dim varFTPLogFile As Variant
Dim intOverwriteLog As Integer
Dim intPassiveMode As Integer
Dim varLocalFile As Variant
Dim varRemoteFile As Variant
Dim intTransType As Integer

   varHost = "www.hostname.com"
   varUsername = "username"
   varPassword = "password"
   intPort = 21
   varFTPLogFile = "c:\ftp.log"
   intOverwriteLog = 1

   varLocalFile = "c:\mytest.txt"
   varRemoteFile = "/MyDir/"

   Action2.FTPLogin (varHost, varUsername, varPassword, intPort, varFTPLogFile, intOverwriteLog,
intPassiveMode)
   intReturn = Action2.FTPUpload (varLocalFile, varRemoteFile, intTransType)
   Action2.FTPLogout

   If intReturn = 0 Then
      MsgBox "Failure"
      MsgBox( Action2.GetLastError )
   ElseIf intReturn = 1 Then
      MsgBox "Success"
   End If
```

```
End Sub
```

InputToVar Example

TO DO: Enter topic text here!

InstallService Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varExecutable As Variant
Dim varDisplayName As Variant
Dim varUsername As Variant
Dim varPassword As Variant
Dim intErrorType As Integer
Dim intServiceType As Integer
Dim intRunType As Integer
Dim intInteractive As Integer


    varExecutable = "c:\Program Files\Program\program.exe"
    varDisplayName = "MyProgram"
    varUsername = "username"
    varPassword = "password"
    intErrorType = 1
    intServiceType = 1
    intRunType = 0
    intInteractive = 0

    intReturn = Action2.InstallService (varExecutable, varDisplayName, varUsername, varPassword,
intErrorType, intServiceType, intRunType, intInteractive)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

LeftClick Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.LeftClick

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

LeftDblClick Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.LeftDblClick

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

LockKeyboard Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.LockKeyboard

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success( use mouse to clear this message )"
    End If

End Sub
```

LockMouse Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.LockMouse

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success( use keyboard to clear this message )"
    End If

End Sub
```

Login Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varUsername As Variant
Dim varPassword As Variant
Dim varDomainName As Variant
Dim intNewDesktop As Integer
Dim intInteractive As Integer
Dim intLaunchExplorer As Integer
Dim intLockMouse As Integer
Dim intLockKeyboard As Integer

    varUsername = "username"
    varPassword = "password"
    varDomainName = "domainname"
    intNewDesktop = 1
    intInteractive = 1
    intLaunchExplorer = 0
    intLockMouse = 0
    intLockKeyboard = 0

    intReturn = Action2.Login (varUsername, varPassword, varDomainName, intNewDesktop,
intInteractive, intLaunchExplorer, intLockMouse, intLockKeyboard)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

LogoffMachine Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.LogoffMachine

    If intReturn = 0 Then
        MsgBox "LogOff failed"
        MsgBox( Action2.GetLastError )
    End If

End Sub
```

Message Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varMessageText As Variant
Dim intModal As Integer

varMessageText = "This is an example of the Action2.Message command"
intModal = 1

    intReturn = Action2.Message(varMessageText, intModal)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

MiddleClick Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.MiddleClick

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

MiddleDblClick Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.MiddleDblClick

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

MoveMouse Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim intXPos As Integer
Dim intYPos As Integer

    intXPos = 10
    intYPos = 10

    intReturn = Action2.MoveMouse (intXPos, intYPos)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

OpenDocument Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varDocument As Variant

    varDocument = "c:\mytest.txt"

    intReturn = Action2.OpenDocument (varDocument)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

OpenWebpage Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWebpageAddress As Variant

    varWebpageAddress = "www.unisyn.com"

    intReturn = Action2.OpenWebpage (varWebpageAddress)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

Password Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varMessageText As Variant
Dim varPassword As Variant
Dim intMaxRetries As Integer

    varMessageText = "Enter Password"
    varPassword = "password"
    intMaxRetries = 3

    intReturn = Action2.Password (varMessageText, varPassword, intMaxRetries)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

PasteKeys Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varKeysToPaste As Variant

    varKeysToPaste = "Paste these keys"

    intReturn = Action2.PasteKeys (varKeysToPaste)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

PauseService Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varServiceName As Variant

    varServiceName = "ServiceToPause"

    intReturn = Action2.PauseService (varServiceName)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

PingMachine Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varMachineAddress As Variant
Dim intPingTimeout As Integer
Dim varVarToSet As Variant
Dim intPingAction As Integer
Dim varPingTaskToStart As Variant
Dim intNoPingAction As Integer
Dim varNoPingTaskToStart As Variant

    varMachineAddress = "www.unisyn.com"
    intPingTimeOut = 1500
    varVarToSet = ""
    intPingAction = 2
    varPingTaskToStart = ""
    intNoPingAction = 1
    varNoPingTaskToStart    = ""

    intReturn = Action2.PingMachine (varMachineAddress, intPingTimeout, varVarToSet, intPingAction,
varPingTaskToStart, intNoPingAction, varNoPingTaskToStart)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

PlayMouse Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varFilename As Variant

    varFilename = "C:\RecordTest.dat"

    intReturn = Action2.PlayMouse (varFilename)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

PlaySound Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varSoundToPlay As Variant
Dim varSound As Variant
Dim intWaitUntilFinished As Integer

    varSoundToPlay = "c:\somewavefile.wav"
    intWaitUntilFinished = 0

    intReturn = Action2.PlaySound (varSoundToPlay, intWaitUntilFinished)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

PrintDocument Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varDocument As Variant

    varDocument = "c:\mytest.txt"

    intReturn = Action2.PrintDocument (varDocument)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

PromptUser Example

```
Sub Main
Dim intReturn As Integer

Dim varText As Variant
Dim intType As Integer
Dim intActionOnOK As Integer
Dim intFailOnCancel As Integer
Dim intFailOnNo As Integer
Dim varTaskToStart As Variant

    varText = "Text to prompt the user"
    intType = 1
    intActionOnOK = 0
    intFailOnCancel = 1
    intFailOnNo = 1
    varTaskToStart = "TaskToStartOnCancel"

    intReturn = Action2.PromptUser (varText, intType, intActionOnOK, intFailOnCancel, intFailOnNo,
varTaskToStart)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    ElseIf intReturn = 2 Then
        MsgBox "Task Was Stopped"
    End If

End Sub
```

QuitAutoMate™ Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.QuitAutoMate

    If intReturn = 0 Then
        MsgBox "QuitAutoMate failed"
        MsgBox( Action2.GetLastError )
    End If

End Sub
```

RebootMachine Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.RebootMachine

    If intReturn = 0 Then
        MsgBox "Reboot failed"
        MsgBox( Action2.GetLastError )
    End If

End Sub
```

Reminder Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varReminderText As Variant
Dim intReschedule As Integer

    varReminderText = "This is a reminder"
    intReschedule = 5

    intReturn = Action2.Reminder (varReminderText, intReschedule)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

RemoveService Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varServiceName As Variant
Dim intStopFirst As Integer

    varServiceName = "ServiceToRemove"
    intStopFirst = 1

    intReturn = Action2.RemoveService (varServiceName, intStopFirst)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

RightClick Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.RightClick

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

RightDblClick Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.RightDblClick

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

RunScript Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varFilename As Variant

    varFilename = "c:\My Script.BAS"

    intReturn = Action2.RunScript (varFilename)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

SendEmail Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varServer As Variant
Dim varUsername As Variant
Dim varTo As Variant
Dim varCC As Variant
Dim varSubject As Variant
Dim varBodyFileName As Variant
Dim varAttatchmentFilename As Variant

    varServer = "mail.mailserver.com"
    varUsername = "user@domain.com"
    varTo = "recipient@anotherdomain.com"
    varCC = ""
    varSubject = "Subject"
    varBodyFilename = "c:\Message.amm"
    varAttachmentFilename = "c:\bakup.zip"

    intReturn = Action2.SendEmail (varServer, varUsername, varTo, varCC, varSubject,
varBodyFilename, varAttachmentFilename)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

SendKeystrokes Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varKeystrokes As Variant
Dim intPauseFirst As Integer
Dim intPauseLength As Integer

    varKeystrokes = "Send these keystrokes"
    intPauseFirst = 1
    intPauseLength = 2000

    intReturn = Action2.SendKeystrokes (varKeystrokes, intPauseFirst, intPauseLength)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

SendRawStep Example

This example sends a message box step in raw format to the AutoMate™ Task Service. Note that you must exercise caution when using this approach; AutoMate™ will not check your syntax, but instead passes the step verbatim to the interpreter for processing.

```
Sub Main
    Action2.SendRawStep (“MESSAGE: “ + Chr$(13) + “This is a step.” + Chr$(13) + “, 1”)
End Sub
```

SendVar Example

TO DO: Enter topic text here!

SetVar Example

TO DO: Enter topic text here!

ShutdownMachine Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.ShutdownMachine

    If intReturn = 0 Then
        MsgBox "Shutdown failed"
        MsgBox( Action2.GetLastError )
    End If

End Sub
```

StartApplication Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varCommandLine As Variant
Dim varParameters As Variant
Dim varDefaultDir As Variant
Dim intWindowState As Integer
Dim intWaitUntilReady As Integer
Dim intWaitUntilDone As Integer

varCommandLine = "notepad"
varParameters = ""
varDefaultDir = ""
intWindowState = 0
intWaitUntilReady = 1
intWaitUntilDone = 0

    intReturn = Action2.StartApplication (varCommandLine, varParameters, varDefaultDir,
intWindowState, intWaitUntilReady, intWaitUntilDone)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

StartService Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varServiceName As Variant

    varServiceName = "ServiceToStart"

    intReturn = Action2.StartService (varServiceName)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

StartTask Example

This example first checks to see if the currently active window is titled “Untitled – Notepad.” If it is, it starts the task “Close Notepads”, and waits for that task to complete before displaying a message box that says “Close Notepads Task Complete.”

```
Sub Main
    If Action2.GetFocusedWindowName = “Untitled – Notepad” Then
        Action2.StartTask (“Close Notepads”, 1, 0)
        MsgBox “Close Notepads Task Complete”
    End If
End Sub
```

StopService Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varServiceName As Variant

    varServiceName = "ServiceToStop"

    intReturn = Action2.StopService (varServiceName)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

StopSound Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.StopSound

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

UnlockKeyboard Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    Action2.LockKeyboard
    Wait 1
    intReturn = Action2.UnlockKeyboard

    If intReturn = 0 Then
        MsgBox "Failure( use mouse to clear this message )"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success( use keyboard to clear this message )"
    End If

End Sub
```

UnlockMouse Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    Action2.LockMouse
    Wait 1
    intReturn = Action2.UnlockMouse

    If intReturn = 0 Then
        MsgBox "Failure( use keyboard to clear this message )"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success( use mouse to clear this message )"
    End If

End Sub
```

UpdateAutoMate™ Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim intUpdateLocation As Integer
Dim varFTPServer As Variant
Dim varNetworkPath As Variant
Dim intShowProgress As Integer
Dim intForceReboot As Integer

    intUpdateLocation = 0
    varFTPServer = "ftp.unisyn.com"
    varNetworkPath = ""
    intShowProgress = 1
    intForceReboot = 0

    intReturn = Action2.UpdateAutoMate (intUpdateLocation, varFTPServer, varNetworkPath,
intShowProgress, intForceReboot)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

Wait Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim intTimeToWait As Integer

    intTimeToWait = 3

    intReturn = Action2.Wait (intTimeToWait)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

WaitForWindow Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowText As Variant
Dim intExactMatch As Integer
Dim intIncludeChildren As Integer
Dim intWaitTime As Integer
Dim intCheckInterval As Integer
Dim intNoWindowAction As Integer
Dim varTaskToStart As Variant

    varWindowText = "Untitled - Notepad"
    intExactMatch = 0
    intIncludeChildren = 0
    intWaitTime = 5
    intCheckInterval = 1
    intNoWindowAction = 0
    varTaskToStart = "TaskNameToStart"

    intReturn = Action2.WaitForWindow (varWindowText, intExactMatch, intIncludeChildren,
intWaitTime, intCheckInterval, intNoWindowAction, varTaskToStart)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    ElseIf intReturn = 2 Then
        MsgBox "Task Was Stopped"
    End If

End Sub
```

WaitForWindowToDisappear Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowTitle As Variant
Dim intExactMatch As Integer
Dim intIncludeChildren As Integer
Dim intWaitInfinite As Integer
Dim intInterval As Integer
Dim intWaitTime As Integer
Dim intStillPresentAction As Integer
Dim varTaskToStart As Variant

    varWindowTitle = "Untitled - Notepad"
    intExactMatch = 0
    intIncludeChildren = 0
    intWaitInfinite = 1
    intInterval = 1
    intWaitTime = 5
    intStillPresentAction = 0
    varTaskToStart = "TaskNameToStart"

    intReturn = Action2.WaitForWindowToDisappear (varWindowTitle, intExactMatch,
intIncludeChildren, intWaitInfinite, intInterval, intWaitTime, intStillPresentAction,    varTaskToStart)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    ElseIf intReturn = 2 Then
        MsgBox "Task Was Stopped"
    End If

End Sub
```

WindowClose Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowTitle As Variant
Dim intExactMatch As Integer

    varWindowTitle = "Untitled - Notepad"
    intExactMatch = 0

    intReturn = Action2.WindowClose(varWindowTitle, intExactMatch)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

WindowFocus Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowTitle As Variant
Dim intExactMatch As Integer

    varWindowTitle = "Untitled - Note"
    intExactMatch = 0

    intReturn = Action2.WindowFocus(varWindowTitle, intExactMatch)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

WindowHide Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowTitle As Variant
Dim intExactMatch As Integer

    varWindowTitle = "Untitled - Notepad"
    intExactMatch = 0

    intReturn = Action2.WindowHide(varWindowTitle, intExactMatch)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

WindowMaximize Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowTitle As Variant
Dim intExactMatch As Integer

    varWindowTitle = "Untitled - Notepad"
    intExactMatch = 0

    intReturn = Action2.WindowMaximize(varWindowTitle, intExactMatch)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

WindowMinimize Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowTitle As Variant
Dim intExactMatch As Integer

    varWindowTitle = "Untitled - Notepad"
    intExactMatch = 0

    intReturn = Action2.WindowMinimize(varWindowTitle, intExactMatch)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

WindowRestore Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowTitle As Variant
Dim intExactMatch As Integer

    varWindowTitle = "Untitled - Notepad"
    intExactMatch = 0

    intReturn = Action2.WindowRestore(varWindowTitle, intExactMatch)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

WindowUnhide Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varWindowTitle As Variant
Dim intExactMatch As Integer

    varWindowTitle = "Untitled - Notepad"
    intExactMatch = 0

        intReturn = Action2.WindowUnhide(varWindowTitle, intExactMatch)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

AMMoveFile Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

Dim varSourceFile As Variant
Dim varDestFile As Variant

    varSourceFile = "C:\*.txt"
    varDestFile = "C:\AnotherDir\"

    intReturn = Action2.AMMoveFile (varSourceFile, varDestFile)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

PopulateFromClipboard Method

Action2 Example

Encapsulates the AutoMate™ action in the “Variables” group, “Populate variable with clipboard.” The method will populate the variable name you pass into it with the current contents of the clipboard.

Declaration

function PopulateFromClipboard (varVariableName As Variant) As Integer

Parameters

varVariableName

The name of the AutoMate™ variable to populate with the clipboard’s contents

Description

The PopulateFromClipboard method will copy the current contents of the Windows System clipboard and put them into the variable passed by *varVariableName*.

The variable must already exist before the script begins by using a Create Variable step in the step building, or the CreateVar method.

The clipboard must have valid text or else the action will fail.

Zip Method

Action2 Example

Encapsulates the AutoMate™ action in the “Zip” group, “Zip files”. he method zips the files you specify into a valid zip file.

Declaration

function Zip (varFilesToZip As Variant,
varFilesToExclude As Variant,
varZipFilename As Variant,
intIncludeSubDirs As Integer,
intPreservePaths As Integer) As Integer

Parameters

varFilesToZip

Contains the filenames (including paths) to be included in the zip file. Separate multiple files with a semi-colon (;)

varFilesToExclude

Contains the filenames (including paths) to be excluded from the list provided in *varFilesToZip*. See **description** below for more details.

varZipFilename

The filename and path of the zip file to create.

intIncludeSubDirs

Set this parameter to “0” if you do not want to include subdirectories in the zip file, or set it to “1” if you do.

intPreservePaths

Set this parameter to “0” if you do not want to preserve the paths of the zip files that are being added (i.e., they all become relative to wherever the zip file is extracted to). Or set it to “1” if you want to keep the directory information in the zip file

Description

The Zip method compresses the files contained in *varFilesToZip* into one valid zip archive with the filename *varZipFilename*. You may use any number of wildcards in *varFilesToZip* to compress one or more directories. In this case, you can use *varFilesToExclude* to exclude certain files that match another mask. For example, if you want to compress everything in the directory “C:\scripts” *except* files ending with .txt, you would set *varFilesToZip* to “C:\scripts”, and *varFilesToExclude* to “C:*.txt.”

Setting *intIncludeSubDirs* to “1” will cause the Zip method to compress the directories you specify as well as any directories contained within. Setting it to “0” will cause the Zip action to ignore any directories it encounters.

Setting *intPreservePaths* will cause the Zip method to store information about the path of the file being compressed into the zip archive. This path can then be used when unzipping the archive to restore the same file structure as it existed when the file was compressed. Setting *intPreservePaths* to “0” will

cause all the files to be compressed into one area, and unzipping the archive will unzip all the files into the same directory.

Unzip Method

Action2 Example

Encapsulates the AutoMate™ action in the “Zip” group, “Unzip files.” The method unzips a valid archive in .zip format into the specified directory.

Declaration

```
function UnZip (varFileToUnzip As Variant,
        varUnzipPath As Variant,
        intOverwriteExisting As Integer,
        intPreservePath As Integer) As Integer
```

Parameters

varFileToUnzip

The filename (including path) of the zip file to uncompress.

varUnzipPath

A valid and existing path to where the contents of the zip file should be extracted to.

intOverwriteExisting

If set to “1”, any existing files will be automatically overwritten by the contents of the zip file without warning. If set to “0”, existing files will be skipped and not uncompressed.

intPreservePath

If set to “1”, AutoMate™ unzips the files into the directories contained in the zip file, restoring directories as necessary. If set to “0”, AutoMate™ will unzip the contents of the zip file into the same directory (as specified by *varUnzipPath*) without restoring any directories, effectively ignoring any directory information contained in the zip file.

Description

Use the Unzip method to unzip a valid zip file (created using any of the popular zip compressors, or the AutoMate™ Zip method) into a directory of your choice.

GetFocusedWindowName Method

Action2 Example

Returns the title of the top most, focused window.

Declaration

function GetFocusedWindowName () As Variant

Parameters

None

Description

The GetFocusedWindowName method will return the name of the current foreground window (i.e., the title bar text of the window the user is currently working with). This is almost always the top-level window.

DialUp Method

Action2 Example

Attempts to establish or end a dial up connection.

Declaration

function DialUp (varConnectionName As Variant,
intAction As Integer) As Variant

Parameters

varConnectioName

The name of the phonebook entry to dial.

intAction

If set to “0”, AutoMate™ will attempt to disconnect the connection. If set to “1”, AutoMate™ will attempt to make a connection using *varConnectionName*.

Description

The DialUp method attempts to establish or end a dial-up session by dialing the phonebook entry pointed to by *varConnectionName*. The phonebook entry must already exist in your Dial-Up Networking preferences before running the step.

If *intAction* is set to “0”, AutoMate™ will attempt to find a connection already made using *varConnectionName* and, if found, will try to disconnect the session. If this can be done successfully, the step returns “1.” Otherwise, it returns “0.”

If *intAction* is set to “1”, AutoMate™ will attempt to start a dial-up session using the parameters of the phonebook entry *varConnectionName*. If the connection is established, the method returns “1.” Otherwise, it will return “0.”

DialUp Example

This example will attempt to start a dial-up session using the phonebook entry “My ISP.” If successful, it will open a webpage to Yahoo!, then disconnect.

```
Sub Main
    int iConnectionOkay As Integer

    iConnectionOkay = Action2.DialUp (“My ISP”, 1)
    If iConnectionOkay = 1 Then
        ‘ Connection established!   Try to open a webpage and disconnect
        Action2.OpenWebpage (http://www.yahoo.com);
        Action2.DialUp (“My ISP”, 0)
    Else
        MsgBox “Connection could not be established.”
    End If
End Sub
```

Unzip Example

```
Sub Main
Dim intReturn As Integer

Dim varFileToUnzip As Variant
Dim varUnzipPath As Variant
Dim intOverwriteExisting As Integer
Dim intPreservePaths As Integer

    varFileToUnzip = "c:\*.zip"
    varUnzipPath = "c:\My Documents\"
    intOverwriteExisting = 1
    intPreservePaths = 1

    intReturn = Action2.Unzip (varFileToUnzip, varUnzipPath, intOverwriteExisting, intPreservePaths)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

Zip Example

```
Sub Main
Dim intReturn As Integer

Dim varFilesToZip As Variant
Dim varFilesToExclude As Variant
Dim varZipFilename As Variant
Dim intIncludeSubDirs As Integer
Dim intPreservePaths As Integer

varFilesToZip = "c:\My Documents\*.*"
varFilesToExclude = "*.zip"
varZipFilename ="c:\TodaysZip.zip"
intIncludeSubDirs = 1
intPreservePaths = 1

    intReturn = Action2.Zip (varFilesToZip, varFilesToExclude, varZipFilename, intIncludeSubDirs,
intPreservePaths)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

LeftMouseUp Method

Action2 Example

Simulates releasing the left mouse button after being held down by a call to the LeftMouseDown method.

Declaration

function LeftMouseUp () As Variant

Parameters

None

Description

Use the LeftMouseUp method to simulate releasing the left hand mouse button after it has been held down by a call to LeftMouseDown .

LeftMouseDown Method

Action2 Example

Simulates pressing and holding down the left mouse button.

Declaration

function LeftMouseDown () As Variant

Parameters

None

Description

Use the LeftMouseDown method to simulate holding down the left hand mouse button. Use the LeftMouseUp method to simulate releasing the left mouse button..

MiddleMouseUp Method

Action2 Example

Simulates releasing the middle mouse button after being held down by a call to the MiddleMouseDownmethod.

Declaration

function MiddleMouseUp () As Variant

Parameters

None

Description

Use the MiddleMouseUp method to simulate releasing the middle mouse button after it has been held down by a call to MiddleMouseDown .

MiddleMouseDown Method

Action2 Example

Simulates pressing and holding down the middle mouse button.

Declaration

function MiddleMouseDown () As Variant

Parameters

None

Description

Use the MiddleMouseDown method to simulate holding down the middle mouse button. Use the MiddleMouseUp method to simulate releasing the middle mouse button..

RightMouseUp Method

Simulates releasing the right mouse button after being held down by a call to the RightMouseDown method.

Declaration

function RightMouseUp () As Variant

Parameters

None

Description

Use the RightMouseUp method to simulate releasing the right hand mouse button after it has been held down by a call to RightMouseDown .

RightMouseDown Method

Action2 Example

Simulates pressing and holding down the right mouse button.

Declaration

function RightMouseDown () As Variant

Parameters

None

Description

Use the RightMouseDown method to simulate holding down the right hand mouse button. Use the RightMouseUp method to simulate releasing the right mouse button..

LeftMouseUp Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.LeftMouseUp

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

LeftMouseDown Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.LeftMouseDown

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

MiddleMouseUp Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.MiddleMouseUp

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

MiddleMouseDown Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.MiddleMouseDown

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

RightMouseUp Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.RightMouseUp

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

RightMouseDown Example

```
Option Explicit
Sub Main
Dim intReturn As Integer

    intReturn = Action2.RightMouseDown

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If

End Sub
```

GetLastError Example IAMAction2

```
Option Explicit
Sub Main
Dim intReturn As Integer
Dim varDocument as Variant

    varDocument = “c:\NoFile.txt”
        IntReturn = Action2.OpenDocument (varDocument)

    If intReturn = 0 Then
        MsgBox "Failure"
        MsgBox( Action2.GetLastError )
    ElseIf intReturn = 1 Then
        MsgBox "Success"
    End If
```

GetFocusedWindowName Example IAMAction2

```
Option Explicit
Sub Main
Dim varFocusedWindow As Variant
Dim x As Integer

    For x = 1 To 10
        varFocusedWindow = Action2.GetFocusedWindowName
        If varFocusedWindow = "" Then
                Exit For
        Else
                Action2.WindowMinimize(varFocusedWindow, 0)
        End If
    Next x

End Sub
```