

---

# Installer VISE Examples

---

**Default Install Location Example .....1-1**

The Default Install Location example shows how to dynamically set up a default install location. This can set the install directory to a possibly pre-existing directory at run time.

**Backup Example .....2-1**

The Backup example demonstrates how to set up an installer which will save the existing version of a file before installing a new version.

**Default Registration Information Example .....3-1**

The Default Registration Information example demonstrates how to display the Registration Information screen and set default values for the User Name, Organization and Serial Number fields.

**Launching an Installed App Example .....4-1**

The Launching an Installed App example demonstrates how to allow the user to launch an installed application and/or view a Readme file immediately after installation.

**Shortcut to a Folder Example .....5-1**

The Shortcut to a Folder example illustrates how to create a shortcut to a folder and place the shortcut on the desktop.

**Uninstall an Existing App Example .....6-1**

The Uninstall an Existing App example demonstrates how to launch the uninstaller for a previously installed application from your installer. This example will run the uninstaller for Installer VISE 3.0 if the application resides on the target computer.

**Conditional Package Display Example .....7-1**

The Conditional Package Display example illustrates how to conditionally display packages in the Select Components screen.

**Key Code Example .....8-1**

The Key Code example illustrates how to conditionally install files based on a key code.

**Edit NT Path Example .....9-1**

The Edit NT Path example demonstrates how to edit the environment variable “Path” on a Windows NT machine. For this example, we install an application and then edit the Path variable so that we can launch the application from the installer (or the Command Prompt) without specifying its path.

**Plug-ins Example .....10-1**

The Plug-ins example demonstrates how to install a plug-in for Internet Explorer and Netscape Navigator.

**AutoPlay Example .....11-1**

The AutoPlay example demonstrates how to support AutoPlay for a CD-ROM installer.

**Adding Files to Uninstall Log Example.....12-1**

This example demonstrates how to add a file entry to the uninstall log file. Your installer will automatically make a file entry for files it installs, but there may be instances where you want to remove a file that the installer didn’t create. These could be files the application creates after the installer runs, such as a preferences file.

**Does Registration Key Exist Example.....13-1**

This example demonstrates how to determine if a registry key exists. It does this by calling external code. The external code returns 1 if the key exists and returns 0 if it doesn’t.

**Active Web Install Example .....14-1**

This example demonstrates how to create an Active Web Installer.

**Build Targets Example .....15-1**

This example demonstrate the use of build targets. It is the same example as the Active Web Install example with the added ability to create three different installers from the same project file.

**Message Dialog Example.....16-1**

The Message Dialog example demonstrates how to display a small, modeless dialog during an installation.

## Example 1

# Default Install Location Example

## Default Install Location Example

The Default Install Location example shows how to dynamically set up a default install location. This can set the install directory to a possibly pre-existing directory at run time.

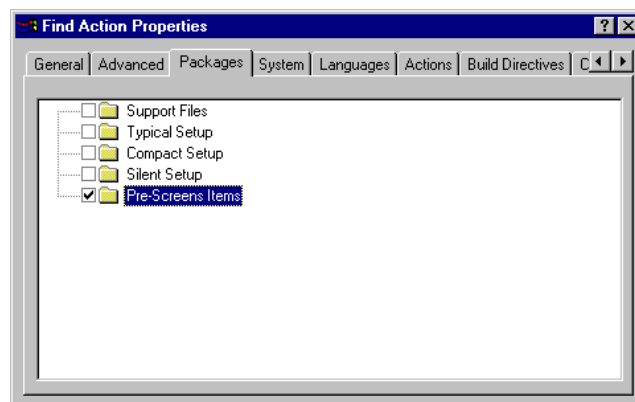
### Features Demonstrated **Look at this example to see how to use:**

- Pre-Screens Items
- Find actions
- Select Install Directory

## How It Works

This example will use the parent directory of a file as the install location. The installer uses a Find action to search the Program Files folder for the file “multipad.exe.”

From the main Installer VISE window, double-click on the Find action. Then click the **Packages** tab. Note that we linked the Find action only to Pre-Screens Items. The installer will execute items from the Pre-Screens Package before it displays any of the dialogs listed under the Screens menu.

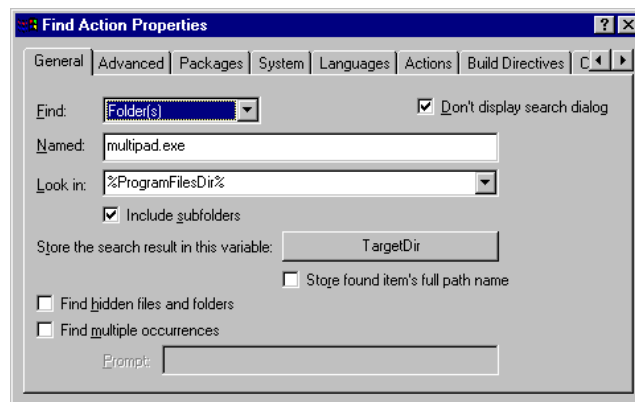


**Illustration 1-1: Find Action Properties Packages Tab**

Now click on the **General** tab. Here the Find action searches for multipad.exe in %ProgramFilesDir% and stores the result in the variable TargetDir.

**Note:** %ProgramFilesDir% and TargetDir are runtime variables - placeholders that will contain a value when the installer runs on the customer's computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

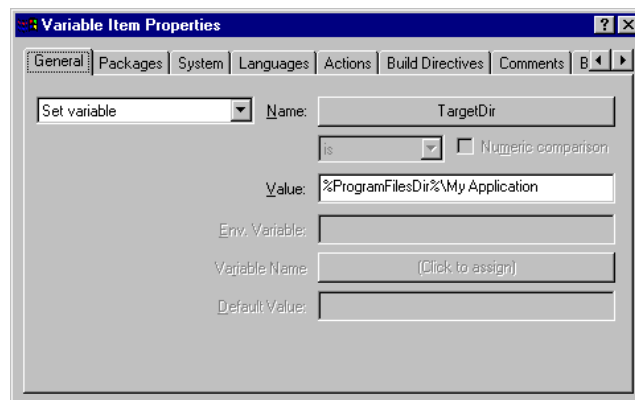
When this installer runs, it will read the registry to find %ProgramFilesDir%, the path name of the Program Files folder. It will then run the Find action to search that directory for multipad.exe. If it finds the file, the installer will store the path in TargetDir, which represents the target directory for installing our files.



**Illustration 1-2: Find Action Properties General Tab**

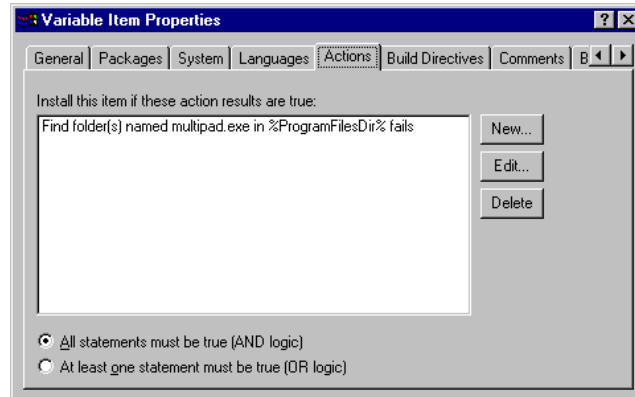
Next, the installer uses a Set Variable action. If executed, this action will change the default install directory to %ProgramFilesDir%\My Application.

Double-click on the Set Variable item to view its properties.



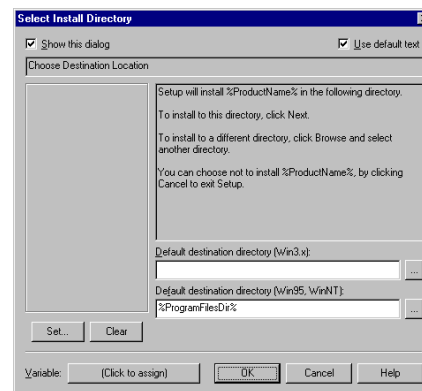
**Illustration 1-3: Variable Item Properties General Tab**

Click on the **Actions** tab. Note that the installer will only use the Set Variable action if the search for multipad.exe fails.



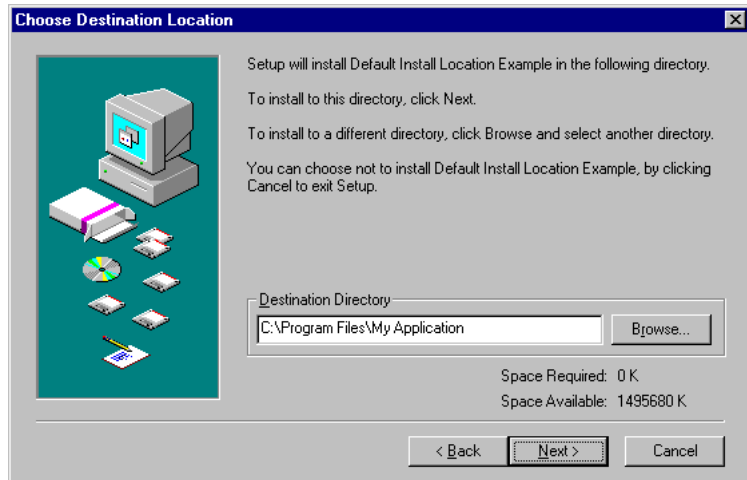
**Illustration 1-4: Variable Item Properties Actions Tab**

From the Screens menu, choose **Select Install Directory**. The default destination directory is %ProgramFilesDir%. However, because our Pre-Screen items will execute first, TargetDir will contain the destination directory at run time.



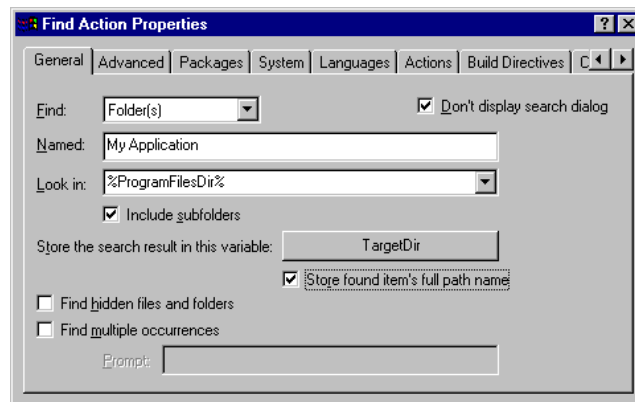
**Illustration 1-5: Select Install Directory Dialog**

When this installer runs, the Select Install Directory dialog will bring up a default destination directory. This will be the directory for `multipad.exe` if our Find action succeeded. Otherwise, the default destination will be `%ProgramFilesDir%\My Application`.



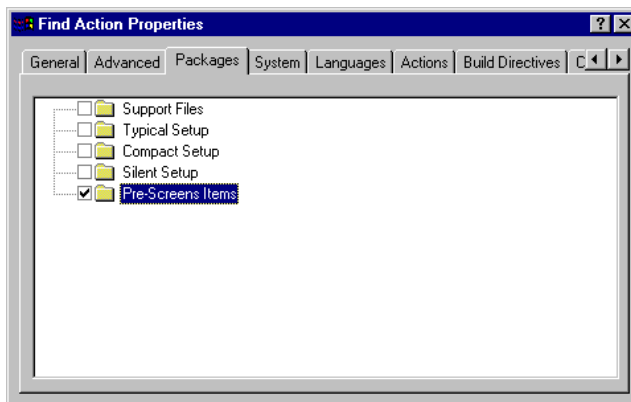
**Illustration 1-6: Choose Destination Location Dialog**

You can also search for a directory and use it as the install location. To do this, set up the Find action as illustrated below:



**Illustration 1-7: Find Action Properties General Tab**

Like the previous example, be sure to restrict this Find action to the Pre-Screens Package.



**Illustration 1-8: Find Action Properties Packages Tab**

## Example 2

# Backup Example

---

### Backup Example

The Backup example demonstrates how to set up an installer which will save the existing version of a file before installing a new version.

### Features Demonstrated

**Look at this example to see how to use:**

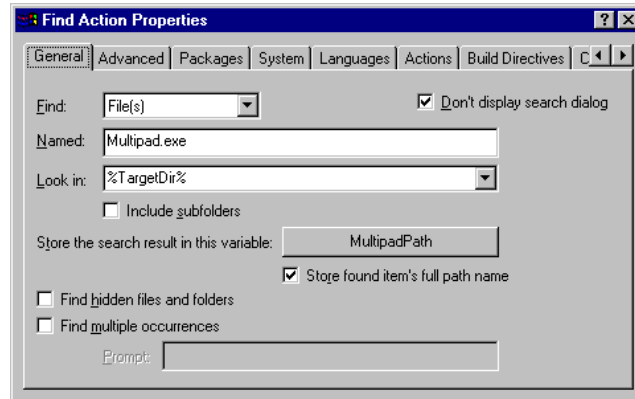
- Find actions
- Move actions
- Message Box

### How It Works

The Backup example uses a Find action to first search the target directory (%TargetDir%) for a file named "Multipad.exe." If the Find action locates the file, it stores the file's full path name in the variable MultipadPath.

**Note:** %TargetDir% and MultipadPath are runtime variables - placeholders that will contain a value when the installer runs on the customer's computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

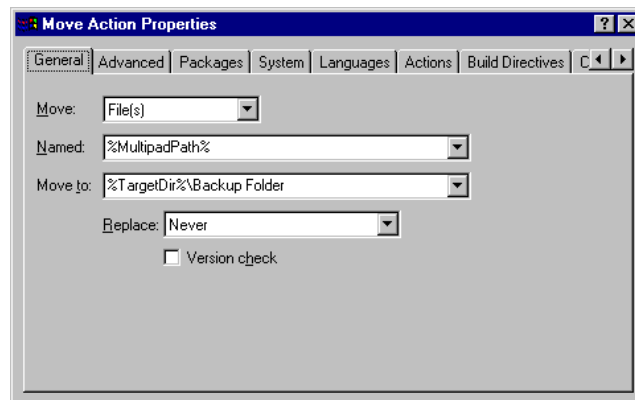
From the main Installer VISE window, double-click on the Find action to display its properties.



**Illustration 2-1: Find Action Properties General Tab**

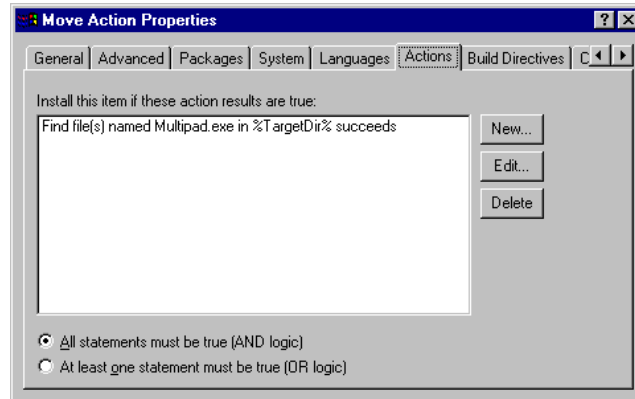
If the Find action succeeds, the installer performs a Move action. This moves the file to a folder named “Backup Folder” in the target directory. We also set Replace to Never, so if the installer runs more than once, the newer copy will not replace the original file.

Double-click on the Move action to view these settings.



**Illustration 2-2: Move Action Properties General Tab**

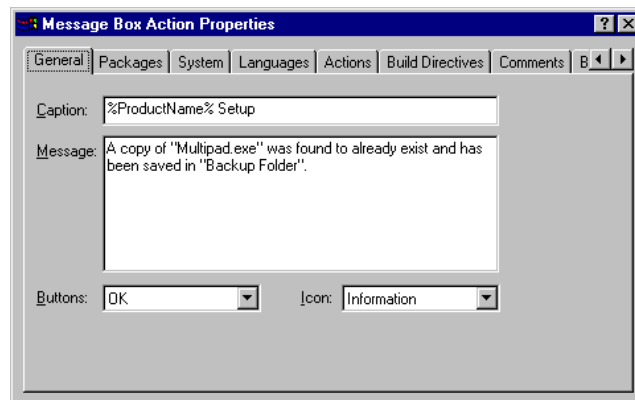
Now click the **Actions** tab. Note that we install the Move action only if the Find action succeeded.



**Illustration 2-3: Move Action Properties Actions Tab**

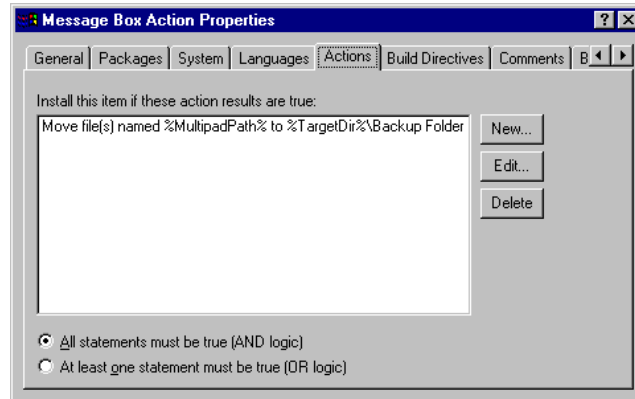
If the Move action was successful, the installer displays a Message Box that states, “A copy of ‘Multipad.exe’ was found to already exist and has been saved in ‘Backup Folder.’”

Double-click the Message Box action to view its properties. Note that this message box will display an Information icon and an OK button.



**Illustration 2-4: Message Box Action Properties General Tab**

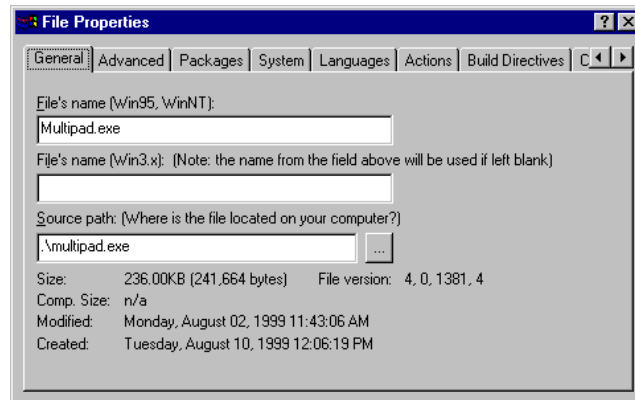
Click the **Actions** tab. Here we set the Installer to display the message only if the Move action occurred:



**Illustration 2-5: Message Box Action Properties Actions Tab**

Finally, we install the file “Multipad.exe” to the target directory.

Double-click on the last action item used in our installer. In this Install File action, we specified the file name and its location on our computer.



**Illustration 2-6: General File Properties Dialog**

---

## Example 3

# Default Registration Information Example

---

## Default Registration Information Example

The Default Registration Information example demonstrates how to display the Registration Information screen and set default values for the User Name, Organization and Serial Number fields.

### Features Demonstrated

**Look at this example to see how to use:**

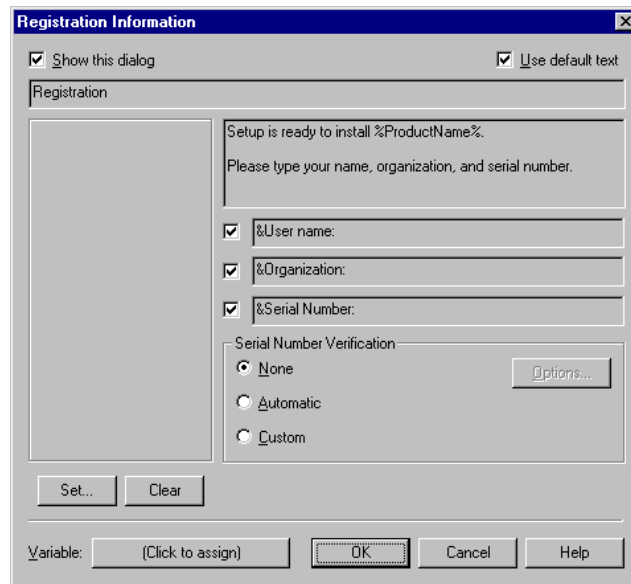
- Registration Information screen
- Read Registry Entry actions
- Write Registry Entry actions

### How It Works

This installer checks the registry for user name, organization and serial number information. If it finds this information, it stores the results in variables that appear as defaults in the Registration Information screen. At that time, the user can accept existing information or enter new information. After the user clicks **OK**, the installer writes the registration information to the registry.

First, we set up this installer to display the Registration Information dialog. To review this setting, choose **Registration Information** from the **Screens** menu.

Here we selected “Show this dialog.” We used Installer VISE’s defaults for the user name, organization and serial number fields (the “&” creates a “hot key” so the user can hold down **Alt** and click an underlined letter to skip to that field).



**Illustration 3-1: Registration Information Dialog**

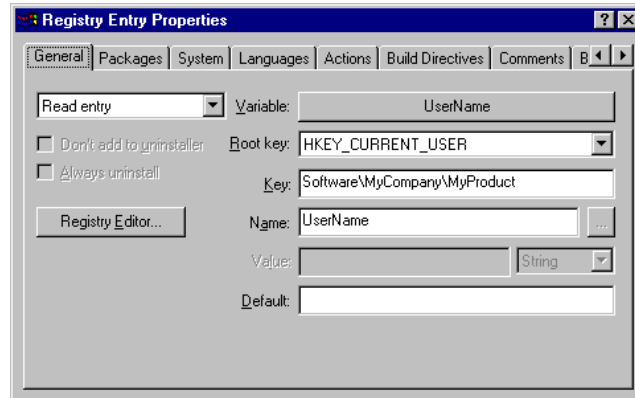
This installer uses three Read Registry actions that are part of the Pre-Screens Items package. The installer will execute these actions before it displays dialogs to the user.

The Read Registry actions look in the registry under the key `HKEY_CURRENT_USER\Software\MyCompany\MyProduct`.

The first Read Registry reads the user name (if present) and saves it in a variable called “UserName.”

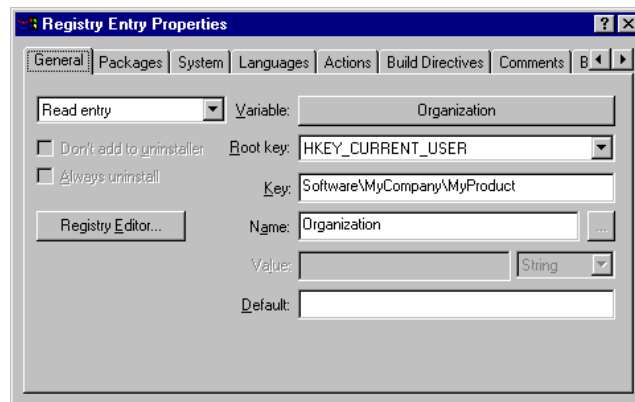
**Note:** UserName is a runtime variable - a placeholder that will contain a value when the installer runs on the customer’s computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

From the main Installer VISE window, double-click on the first Read Registry Entry item.



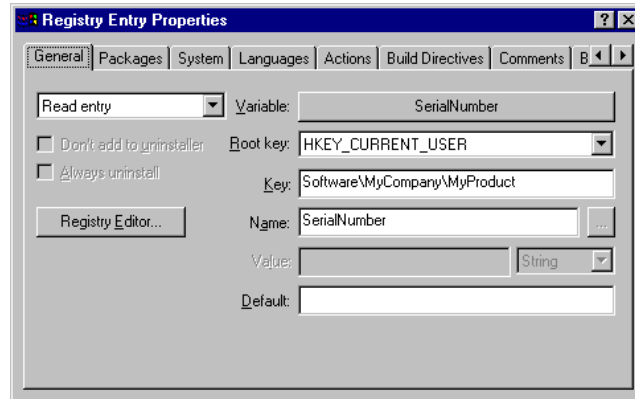
**Illustration 3-2: Read Registry Entry (1)**

The second Read Registry finds and stores existing organization information. Double-click the second action item to review these settings.



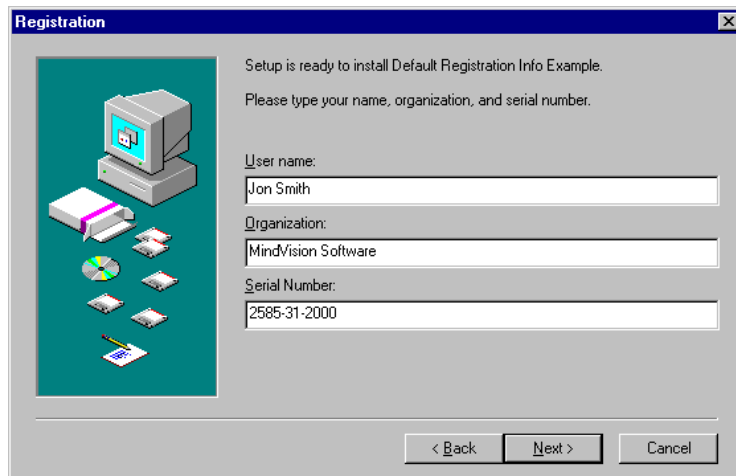
**Illustration 3-3: Read Registry Entry (2)**

The final Read Registry finds and stores existing serial number information. Double-click the third action item to review these settings.



**Illustration 3-4: Read Registry Entry (3)**

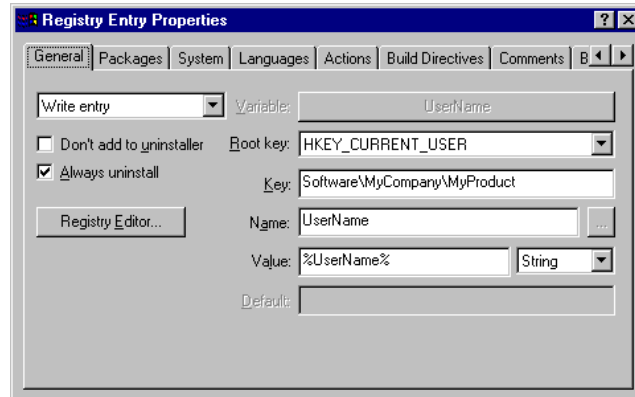
When this installer runs, it will display existing registration information in the Registration Information screen. The user can accept this information or enter new information.



**Illustration 3-5: Registration Screen**

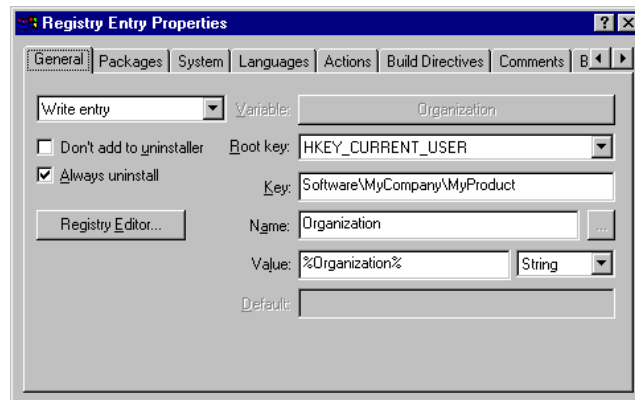
Next, the installer uses three Write Registry Entry actions to record this information to the registry. These actions are part of the Typical Setup package, so the installer will execute them after it displays the Registration Information screen.

The first Write Registry Entry records the user name. From the main Installer VISE window, double-click on the first Write Registry Item to review these settings.



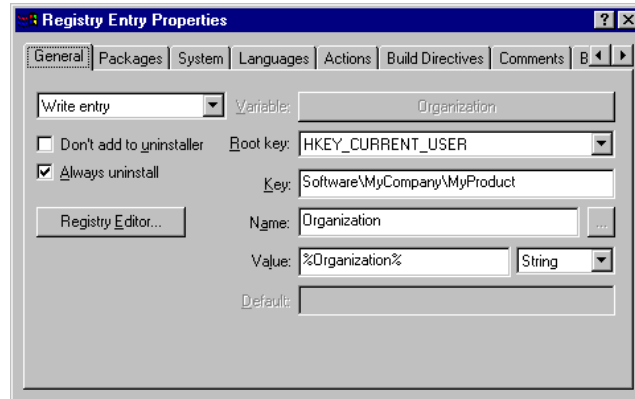
**Illustration 3-6: Write Registry Entry (1)**

The second Write Registry Entry records organization information. Double-click on it to review these settings.



**Illustration 3-7: Write Registry Entry (2)**

The final Write Registry Entry records serial number information. Double-click on it to review these settings.



**Illustration 3-8: Write Registry Entry (3)**

Now that we've recorded the registration information, the installer will display it by default. To see this, run the installer, enter registration information and complete the installation. Then run the installer again to see the defaults appear in the Registration Information screen.

## Example 4

# Launching an Installed App Example

---

## Launching an Installed App Example

The Launching an Installed App example demonstrates how to allow the user to launch an installed application and/or view a Readme file immediately after installation.

### Features Demonstrated    Look at this example to see how to use:

- Installer Properties
- Select Install Directory screen
- Add Files action item
- Finished Message screen

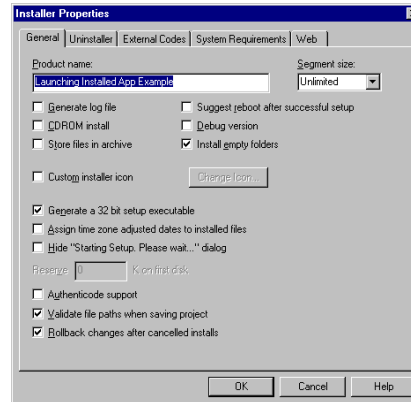
## How It Works

The sample vct installs two files, “Multipad.exe” and “Readme.txt.” It then displays the Finished Message screen, where users can choose to launch the application and/or view the Readme file when the installation concludes.

**Note:** This example uses runtime variables - placeholders that will contain values when the installer runs on the customer’s computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

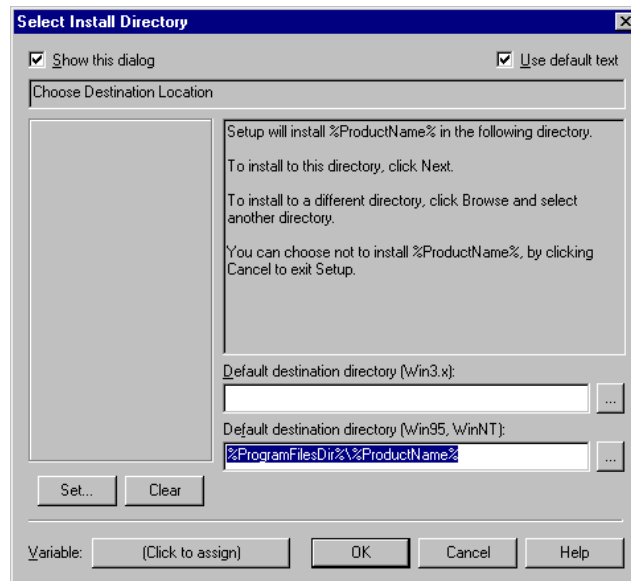
- The %ProductName% variable holds the product’s name. We set this value in the product name field of the Installer Properties dialog box.
- The %ProgramFilesDir% variable contains the path name of the “Program Files” folder as read from the registry entry:  
HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ProgramFilesDir.
- The %TargetDir% variable contains the long path name of the target directory for the installation files. For example, “C:\Program Files\MyProduct.”
- The %WinDir% variable contains the path name of the current Windows directory. For example, “C:\Windows.”

We assigned %ProductName% the value “Launching Installed App Example.” To view this setting, select **Installer Properties** from the **File** menu.



**Illustration 4-1: Installer Properties General Tab**

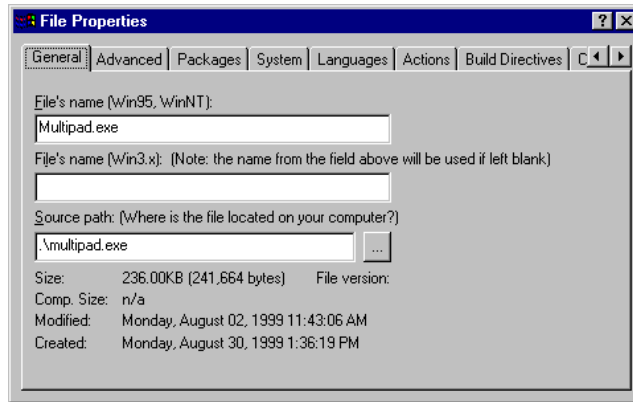
From the **Screens** menu, choose **Select Install Directory**. Notice that the default install directory is %ProgramFilesDir%\%ProductName%.



**Illustration 4-2: Select Install Directory Dialog**

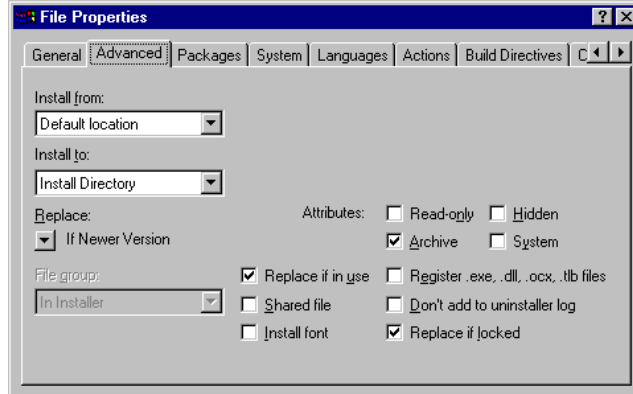
Now double-click on the first action item to display its General properties. We used the Add Items menu to create this action item and the one proceeding it.

**Note:** To install one or many files, select **Items** from the **Add** menu, then double-click **Files**. Installer VISE will prompt you to select files. Just browse your computer system and double-click each file you want to install. When you are finished, click **OK**.



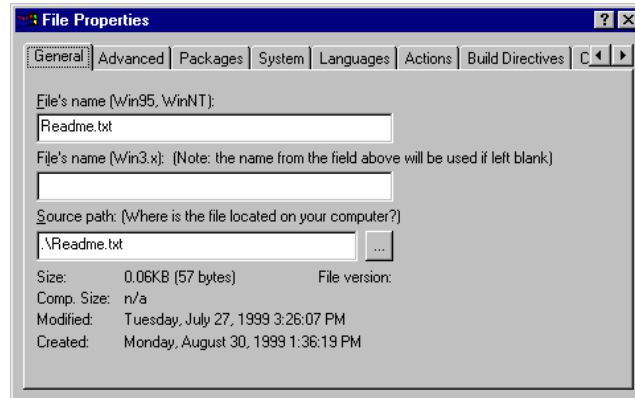
**Illustration 4-3: File Properties General Tab**

Next, click the **Advanced** tab. We used Installer VISE’s defaults here. Note that we’ll install the file to Install Directory. When the installer runs, it will install to the location the user chooses in the Select Install Directory dialog.



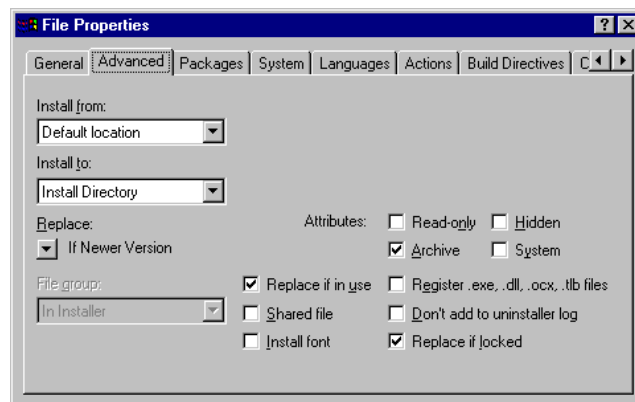
**Illustration 4-4: File Properties Advanced Tab**

Double-click on the second action item to display its General properties.



**Illustration 4-5: File Properties General Tab**

Click on the **Advanced** tab. We used Installer VISE's defaults here.



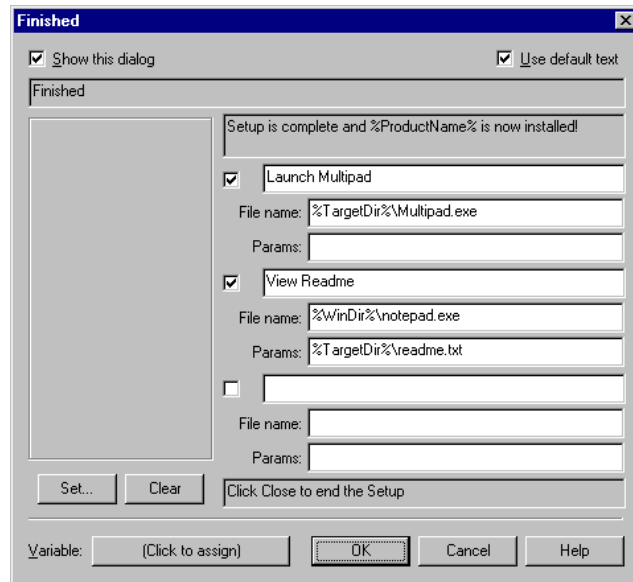
**Illustration 4-6: File Properties Advanced Tab**

Now choose **Finished Message** from the **Screens** menu. Note that we have checked “Show this dialog,” and set it up to present the user with two options, “Launch Multipad” and “View Readme.” We will leave both options on by default.

If the user checks “Launch Multipad,” the installer will launch that application when the installation concludes.

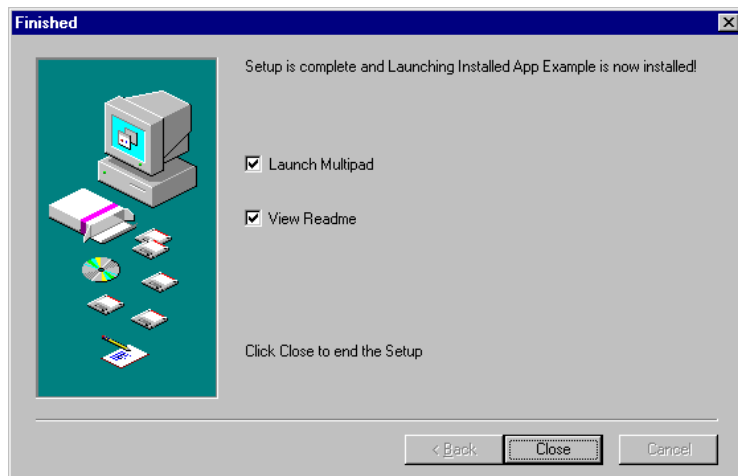
The file name field specifies the path to Multipad, “%TargetDir%\Multipad.exe.”

If the user chooses “View Readme,” the installer will launch Notepad. The Params field specifies the parameter, or path of the file Notepad will open. Notice that the installer opens Notepad from %WinDir%, the path name of the current Windows directory.



**Illustration 4-7: Finished Message Screen Dialog**

The illustration below shows the dialog box displayed at run time.



**Illustration 4-8: Finished Screen Dialog**

## Example 5

# Shortcut to a Folder Example

---

### Shortcut to a Folder Example

The Shortcut to a Folder example illustrates how to create a shortcut to a folder and place the shortcut on the desktop.

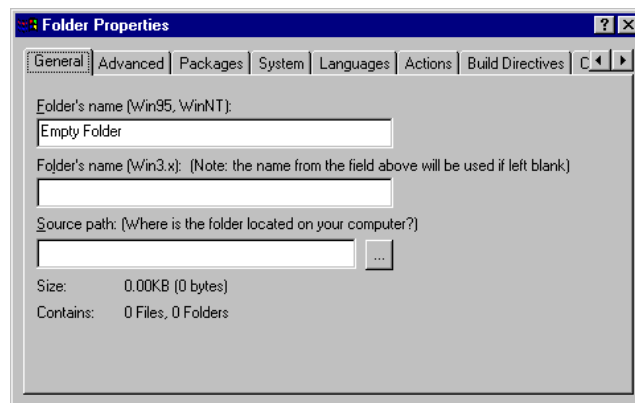
### Features Demonstrated

Look at this example to see how to use:

- Add Folder action items
- Shortcut action items

### How It Works

The example installer first uses an Add Folder action item to create a folder named “Empty Folder.” From the main Installer VISE window, double-click the first action item to display its properties.



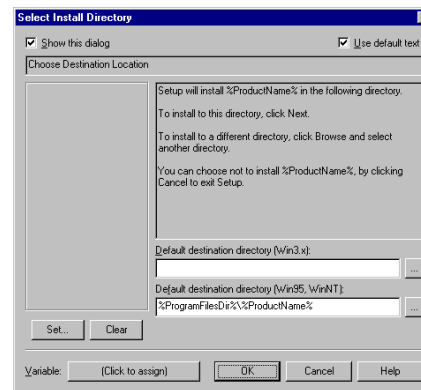
**Illustration 5-1: Folder Properties General Tab**

When this installer runs, the user will choose an install location for the folder.

From the **Screens** menu, choose **Select Install Directory**. Notice that this installer will display %ProgramFilesDir%\%ProductName% as the default install location.

**Note:** %ProgramFilesDir% and %ProductName% are runtime variables - placeholders that will contain a value when the installer runs on the customer's computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

- The %ProductName% variable holds the product's name. We set this value in the product name field of the Installer Properties dialog box.
- The %ProgramFilesDir% variable contains the path name of the "Program Files" folder as read from the registry entry: HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\Current Version\ProgramFilesDir.

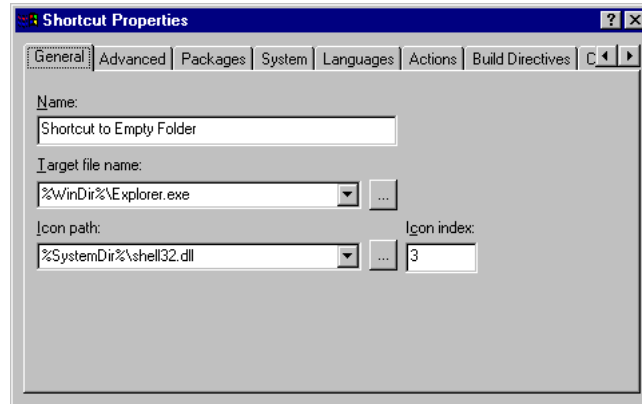


**Illustration 5-2: Select Install Directory Dialog**

After it installs the folder, the installer creates a shortcut to the folder and places the shortcut on the desktop. To view these settings, double-click on the second action item.

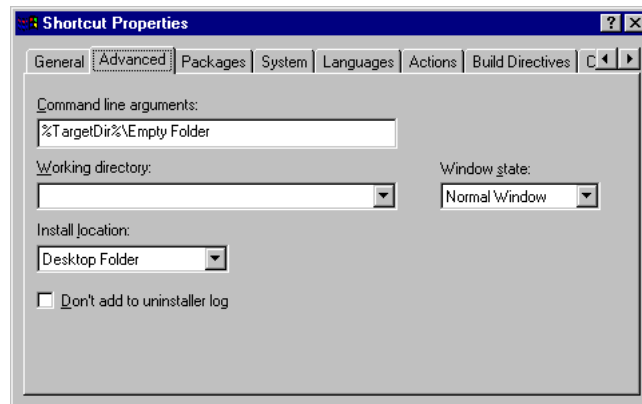
In the second field, we entered "%WinDir%\Explorer.exe." This points the installer to the current Windows directory, where it runs Windows Explorer (the command line argument in Advanced Properties tells Explorer what folder to open).

The icon path “%SystemDir%\shell32.dll” assigns the shortcut a folder icon. The installer will read that specific icon from the current Windows/system directory.



**Illustration 5-3: Shortcut Properties General Tab**

In Advanced Properties, we entered the path to our folder into the command line arguments field. We also set the install location to Desktop Folder.



**Illustration 5-4: Shortcut Properties Advanced Tab**

## Example 6

# Uninstall an Existing App Example

---

## Uninstall an Existing App Example

The Uninstall an Existing App example demonstrates how to launch the uninstaller for a previously installed application from your installer. This example will run the uninstaller for Installer VISE 3.0 if the application resides on the target computer.

### Features Demonstrated

**Look at this example to see how to use:**

- UninstallString
- Read Registry Entry action
- Test Variable action
- Edit Text File action
- Run Application action
- Batch files

### How It Works

The installer example first checks the target computer to determine if Installer VISE 3.0 is present. For this, the installer uses a Read Registry Entry action to find existing uninstall information. Many applications provide this information so that Windows can easily remove them.

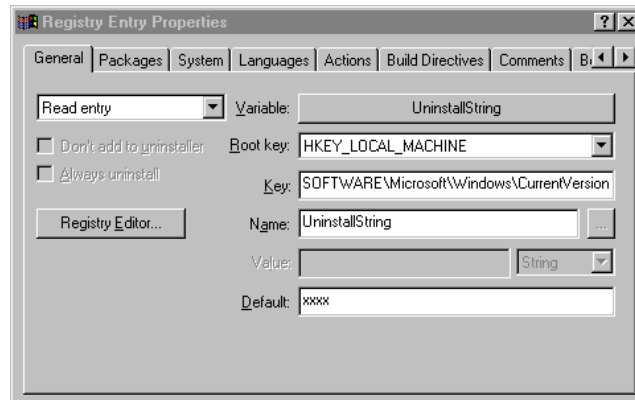
Information for the “Add/Remove Programs” Control Panel applet is stored in the registry under the key `HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall`.

Each product creates a sub-key under this key. This sub-key contains two values, “DisplayName” and “UninstallString.” DisplayName is the string that displays in the “Add/Remove Programs” list. UninstallString is the command line that executes to uninstall the product.

From the main Installer VISE window, double-click the first action item to bring up the Properties dialog.

Here we specified the root key as HKEY\_LOCAL\_MACHINE and the key as SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Installer VISE 3.0. We will store the result in the variable UninstallString and set its default to “xxxx.”

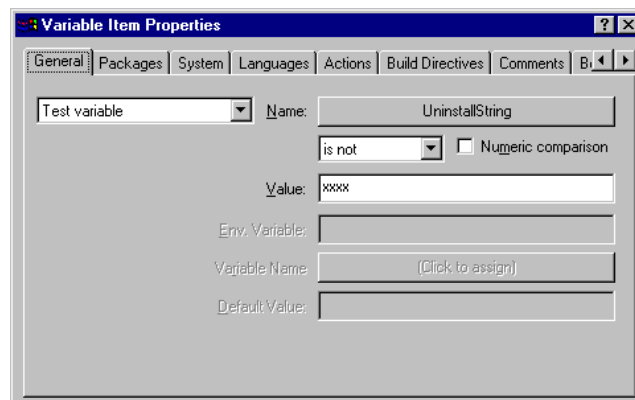
**Note:** UninstallString is a runtime variable - a placeholder that will contain a value when the installer runs on the customer’s computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).



**Illustration 6-1: Registry Entry Properties General Tab**

Next, we use a Test Variable action that compares the value of the variable UninstallString against “xxxx.” If the value is not “xxxx,” we know the Read Registry Entry action succeeded, so Installer VISE 3.0 is present.

Double-click the second action item to display its properties.

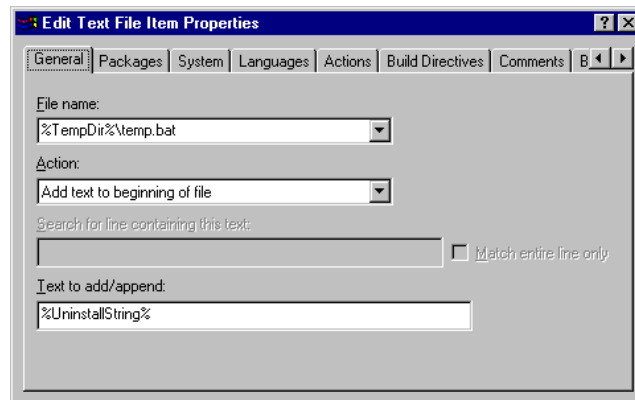


**Illustration 6-2: Variable Item Properties General Tab**

If UninstallString is not “xxxx,” the installer writes the command line to a temporary batch file. It uses an Edit Text File action to do this.

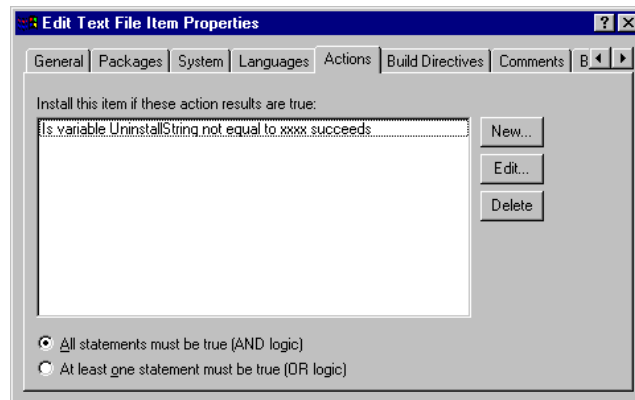
**Note:** A batch file is a text file that contains one or more DOS commands and has a .BAT file name extension. When a batch file runs, it will execute the commands in sequential order. A batch file is a handy alternative to entering each command individually at the Command Prompt.

Double-click the third action item to display its properties. In the General tab, we created a batch file in %TempDir%. The %TempDir% variable contains the destination path name of the temporary directory that the installer creates at runtime and deletes when it is finished. These settings create the batch file “temp.bat” and add %UninstallString% to the beginning of that batch file.



**Illustration 6-3: Edit Text File Item Properties General Tab**

Click on the **Actions** tab. Note that we will only install this action item if the variable UninstallString is not “xxxx.”

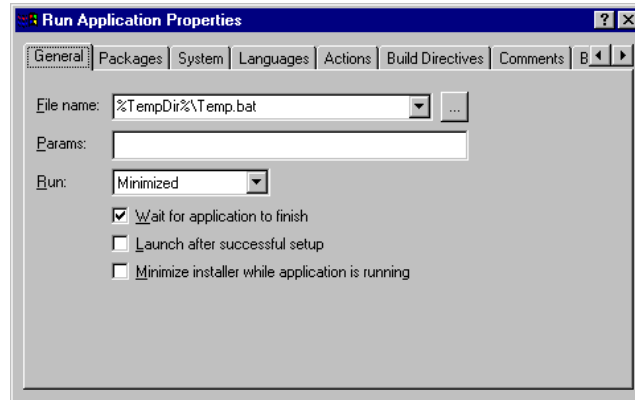


**Illustration 6-4: Edit Text File Item Properties Actions Tab**

Finally, we add a Run Application item to launch the batch file, which will uninstall Installer VISE 3.0.

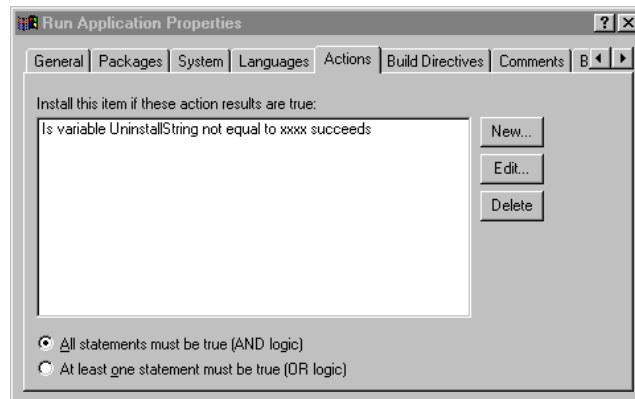
Double-click the fourth action item to display its General properties. These settings will launch temp.bat minimized, so Windows will not display the Command Prompt window.

**Note:** If you want to run an Installer VISE uninstaller silently, type -s or /s in the “Params” field of the Run Application action.



**Illustration 6-5: Run Application Properties General Tab**

Now click the **Actions** tab. Once again, we will only install this action item if the variable “UninstallString” is not “xxxx.”



**Illustration 6-6: Run Application Properties Actions Tab**

## Example 7

# Conditional Package Display Example

---

## Conditional Package Display Example

The Conditional Package Display example illustrates how to conditionally display packages in the Select Components screen.

### Features Demonstrated

**Look at this example to see how to use:**

- Input Box actions
- Label actions
- Goto actions
- Message Box actions
- Test Variable actions

### How It Works

Installers built with Installer VISE will display the Select Components screen in either of the following instances:

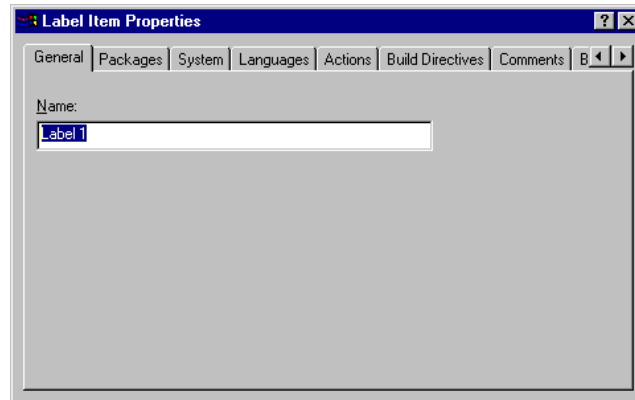
- The installer displays the Select Install Type screen and the user selects “Custom Setup.”
- The installer includes custom packages but it does not display the Select Install Type screen.

For this example we added four packages: “Package 1 (Demo),” “Package 2 (Demo),” “Package 1 (Full),” and “Package 2 (Full).” When this installer runs, it prompts the user to enter a key code. If the key code is invalid, the installer displays a message that notifies the user of an invalid entry. Then it displays the prompt for a key code again, so the user can re-enter the code.

- If the key code is blank, the installer will make only the “Demo” packages available to the user for install.
- If the user enters the key code “Peanuts,” the installer will make only the “Full” packages available for install.

To create this installer, we first added a Label action. This action item creates an insertion point for the Goto item, which we will add later.

From the main Installer VISE window, double-click the first action item to display its properties. In this dialog, we named the label action “Label1.”

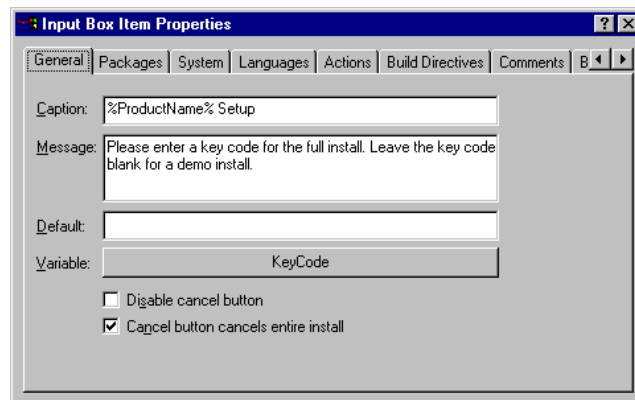


**Illustration 7-1: Label Item Properties General Tab**

Next, we added an Input Box action. This Input Box will display instructions for entering a key code. When the user enters a code, the installer will save the value in the variable “KeyCode.”

**Note:** KeyCode is a runtime variable - a placeholder that will contain a value when the installer runs on the customer’s computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

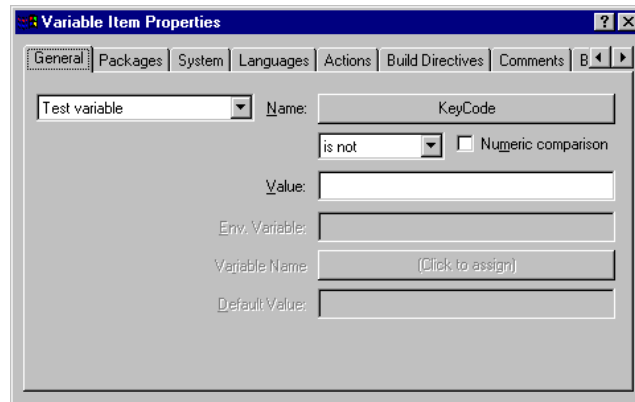
Double-click the second action item to display its General properties.



**Illustration 7-2: Input Box Item Properties General Tab**

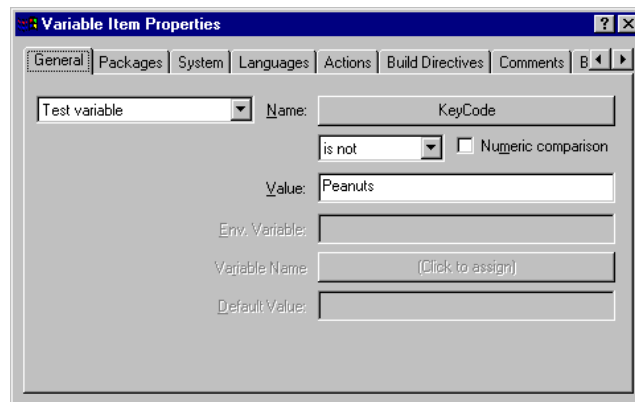
Next, we add two Test Variable actions in order to test the key code the user enters.

The first Test Variable action will check whether the KeyCode variable is blank (contains no value). Double-click the third action item to view these settings.



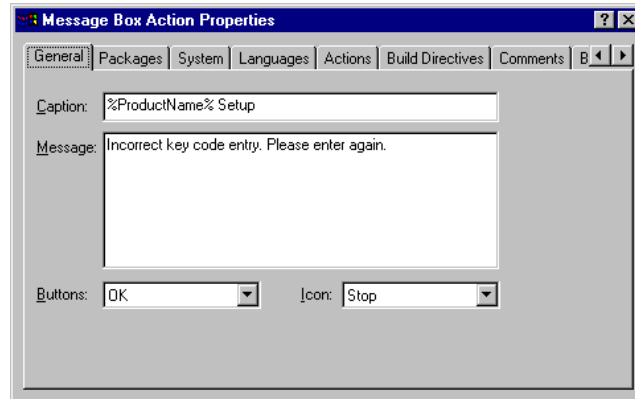
**Illustration 7-3: Variable Item Properties General Tab**

The second Test Variable action will check whether the KeyCode variable contains the value "Peanuts." Double-click the fourth action item to view these settings.



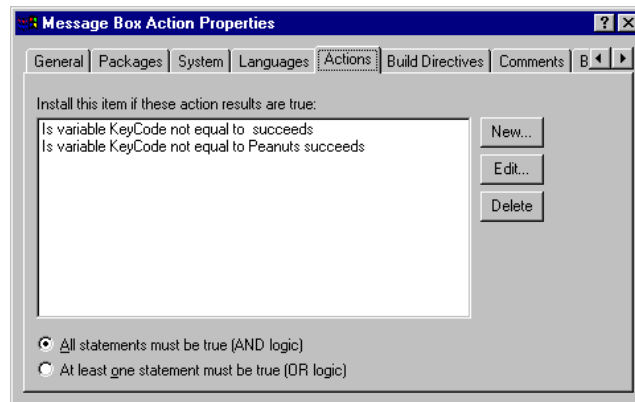
**Illustration 7-4: Variable Item Properties General Tab**

Next, we added a Message Box the installer will display when the user input is invalid. Double-click the fifth action item to view these settings.



**Illustration 7-5: Message Box Action Properties General Tab**

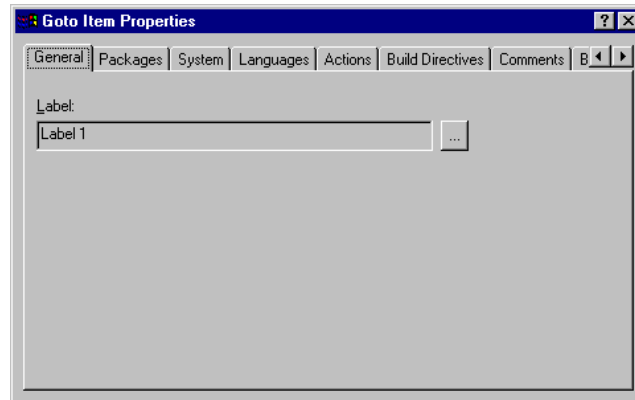
Now click on the **Actions** tab. Here we set the installer to install the Message Box action only if both of our variable tests succeeded. Notice that we checked the “AND logic” option to specify that both tests must succeed.



**Illustration 7-6: Message Box Action Properties Actions Tab**

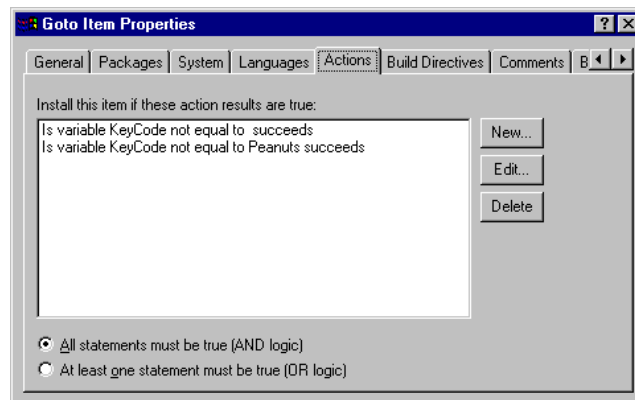
Finally, we added a Goto action that will loop the installer back to the Input Box if user input is invalid.

Double-click on the sixth action item and display the General tab. Here we set the Goto to loop back to Label1.



**Illustration 7-7: Goto Item Properties General Tab**

Click on the Actions tab. Once again, we checked the “AND logic” option so that we will only install this Goto if both our variable tests succeeded.



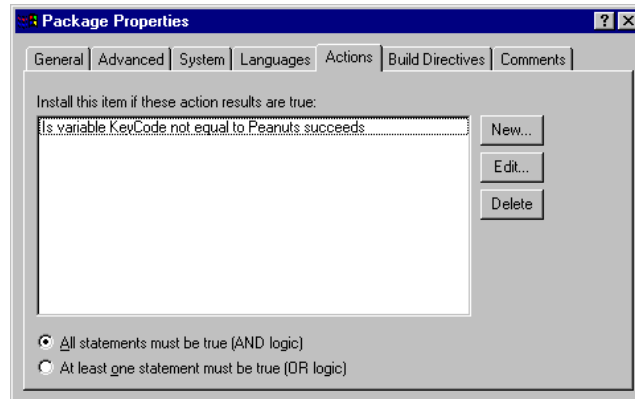
**Illustration 7-8: Goto Item Properties Actions Tab**

We assigned all the above actions to the Pre-Screens Items package. The installer will execute these actions before it displays any dialogs listed under the Screens menu.

**Note:** An item can be a part of none or all of the available packages. To select one or more packages, open the item’s properties, click on **Packages**, and place a check mark next to each package you choose.

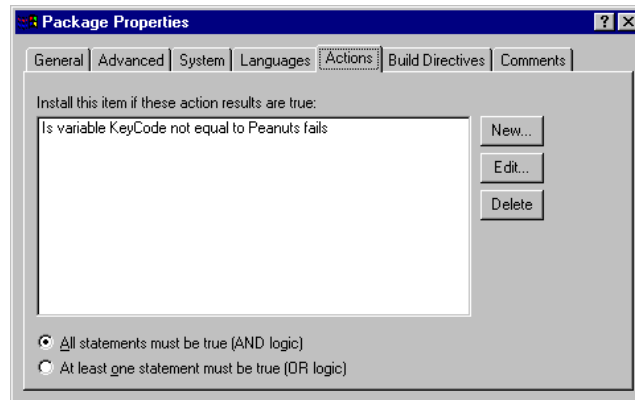
Our installer also must determine what packages to make available for install. For this, it uses the Actions property.

From the Packages window of Installer VISE, double-click on Package 1 (Demo), then click the **Actions** tab. This installer will only make Package 1 (Demo) and Package 2 (Demo) available for install if the user enters a blank return at the key code prompt. The Actions tab for both “Demo” install packages is illustrated below.



**Illustration 7-9: Package Properties Actions Tab**

If the user enters “Peanuts” for a key code, the installer will make both “Full” install packages available for install. Double-click on Package 1 (Full) to display its Actions properties. Package 2 (Full) uses the same settings.



**Illustration 7-10: Package Properties Actions Tab**

## Example 8

# Key Code Example

---

**Key Code Example** The Key Code example illustrates how to conditionally install files based on a key code.

### Features Demonstrated

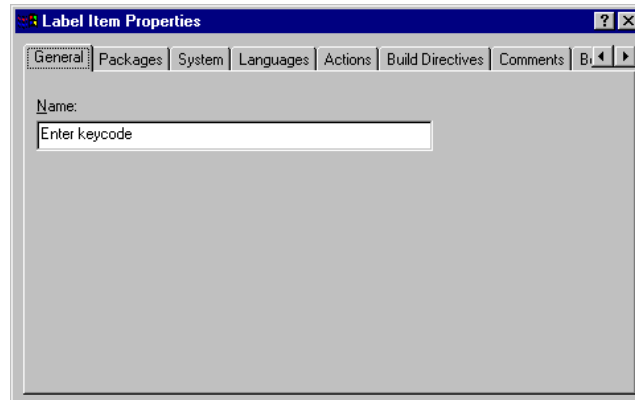
- Label actions
- Input Box actions
- Test Variable Item actions
- Message Box actions
- Goto actions

### How It Works

For this example, we added three files to the installer: “PackageA.txt,” “PackageB.txt,” and “PackageC.txt.” When this installer runs, it will prompt the user to enter a key code. If the user enters “keycodeA,” the installer will install the file PackageA.txt. For “keycodeB,” it will install the file PackageB.txt. For “keycodeC,” it will install the file PackageC.txt. However, if the user enters a key code that is not equal to any of these three values, the installer will again prompt the user for a valid key code.

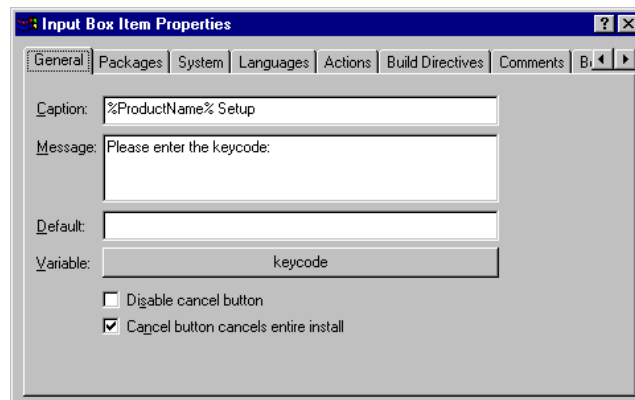
To create this installer, we first added a Label action. This action item creates an insertion point for the Goto item, which we will add later.

From the main Installer VISE window, double-click the first action item to display its properties. Note that we named the Label “Enter keycode.”



**Illustration 8-1: Label Item Properties General Tab**

We then added an Input Box action to the project. This will prompt the user to enter a key code. The installer will save the results in a variable called “keycode.” Double-click the second action item to view these settings.

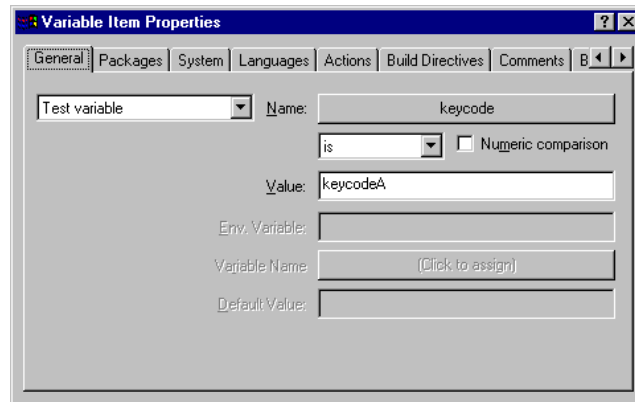


**Illustration 8-2: Input Box Item Properties General Tab**

**Note:** %ProductName% and keycode are runtime variables - placeholders that will each contain a value when the installer runs on the customer’s computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value). Installer VISE gets the value of %ProductName% from the “Product name” field in Installer Properties.

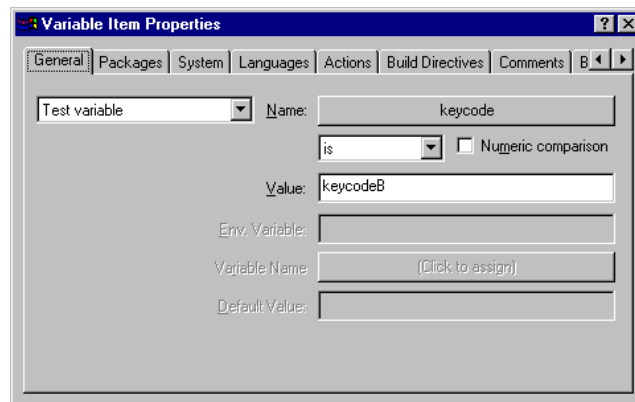
Next, we added three Test Variable Item actions to test for our three valid values. Double-click on the third action item to view its settings.

This test will succeed if the value of the keycode variable is “keycodeA.”



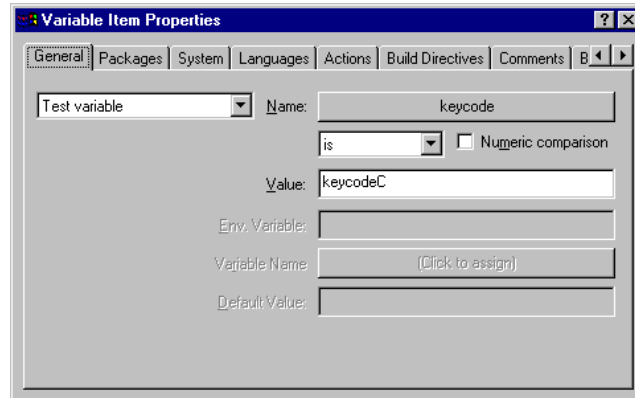
**Illustration 8-3: Variable Item Properties General Tab**

The next test succeeds if the value of keycode is “keycodeB.”



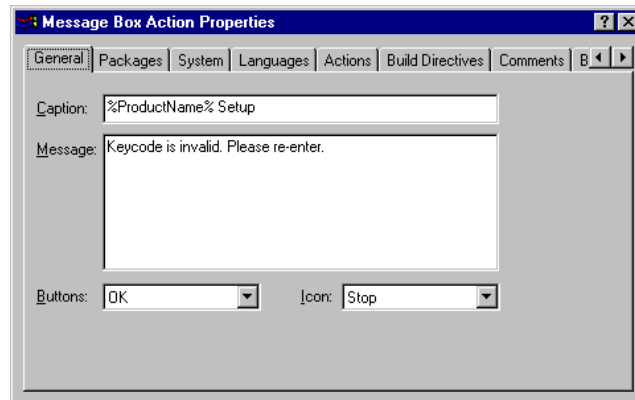
**Illustration 8-4: Variable Item Properties General Tab**

The final test succeeds if the value of keycode is “keycodeC.”



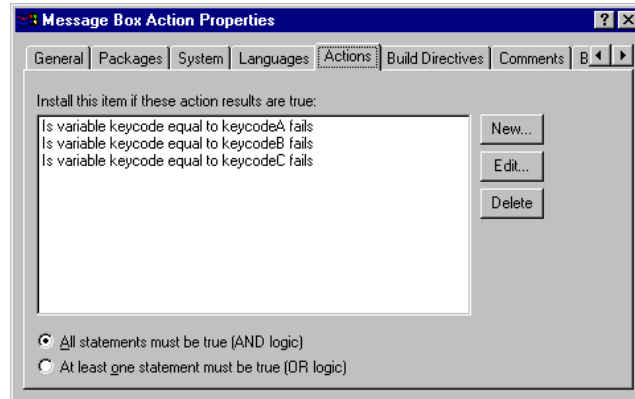
**Illustration 8-5: Variable Item Properties General Tab**

Next, we added a Message Box action to the project. If executed, this Message Box will prompt the user to re-enter the key code. Double-click this action to display its properties.



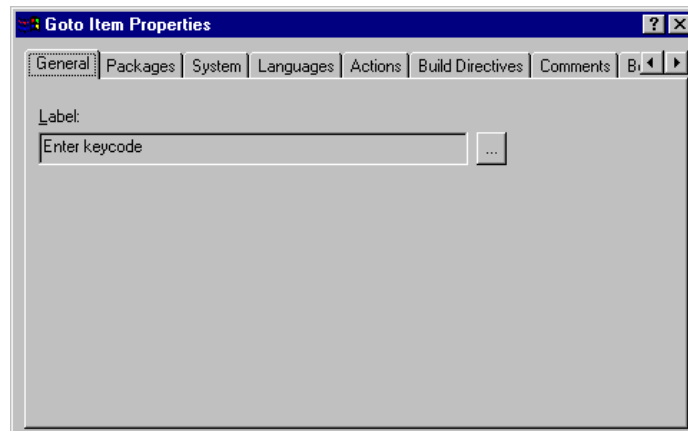
**Illustration 8-6: Message Box Action Properties General Tab**

Click the **Actions** tab. Here we set the installer to only install this action if all three of our variable tests failed.



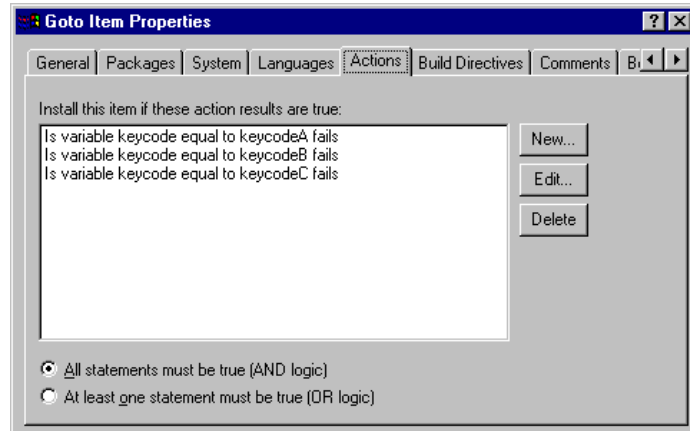
**Illustration 8-7: Message Box Action Properties Advanced Tab**

If all three tests failed, we need to display the Input Box so the user can enter the key code again. For this, we added a Goto action. In the “Label” field, we entered “Enter keycode” so the installer will return to that label.



**Illustration 8-8: Goto Item Properties General Tab**

Click the **Actions** tab. Note that the installer will only install this item if all three of our tests failed.

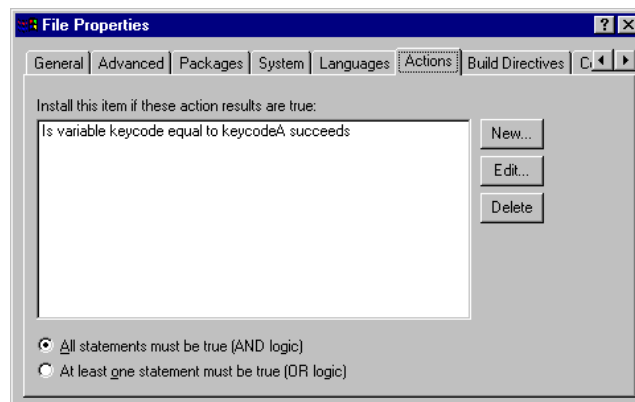


**Illustration 8-9: Goto Item Properties Actions Tab**

Once the user enters a valid key code, the installer moves on to the three Add File action items.

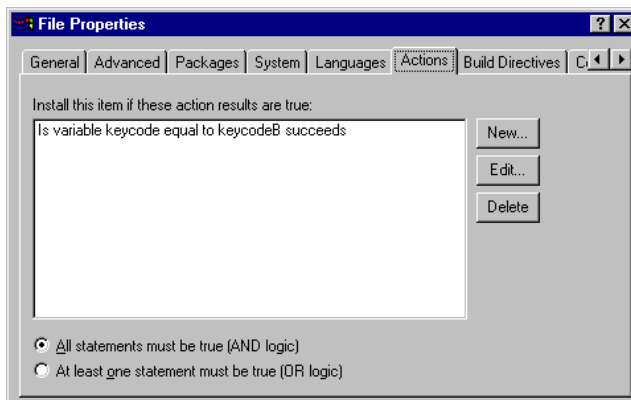
**Note:** We made the Add Files actions part of the Typical Setup package. We made the other actions part of the Pre-Screens Items package so the installer will execute them before it displays any screens from the Screens menu.

Double-click PackageA.txt and click the **Actions** tab to display that property. This illustrates how to install that text file only if the user entered “keycodeA.”



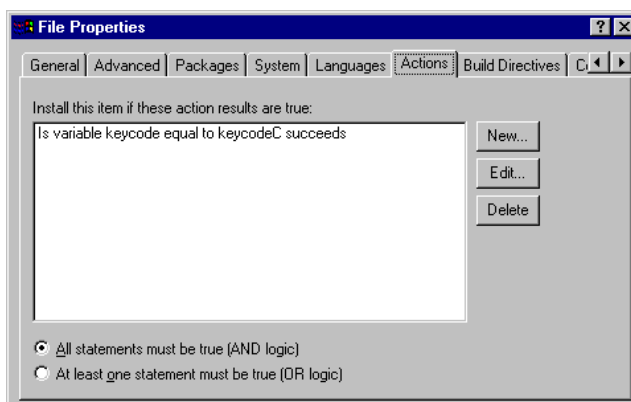
**Illustration 8-10: File Properties Advanced Tab**

Accordingly, we will only install PackageB.txt if the user entered “keycodeB.”



**Illustration 8-11: File Properties Advanced Tab**

The following setting will install PackageC.txt if the user entered “keycodeC.”



**Illustration 8-12: File Properties Advanced Tab**

---

## Example 9

# Edit NT Path Example

---

### Edit NT Path Example

The Edit NT Path example demonstrates how to edit the environment variable “Path” on a Windows NT machine. For this example, we install an application and then edit the Path variable so that we can launch the application from the installer (or the Command Prompt) without specifying its path.

#### Features Demonstrated    Look at this example to see how to use:

- Add Files actions
- Read Registry Entry actions
- Write Registry Entry actions
- Test Variable actions
- Call External Code actions
- Run Application actions

#### How It Works

This installer first installs the file “Multipad.exe.” It then reads the environment variable Path from the registry. If it successfully reads that variable, it appends the value to the target directory path and writes the information back to Path in the registry. This makes the path to Multipad.exe known to Windows NT.

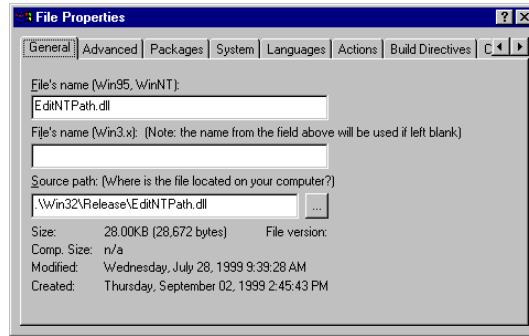
One consideration is that the Path environment variable may contain system variables that Windows must expand in order to read them. An example is %SystemRoot%, which expands to provide the path to the current Windows directory.

The Write Registry action writes the value of Path back to the registry as a string value. However, Windows requires us to write that value back as an expandable string value, so it can correctly expand variables such as %SystemRoot% and Path will be valid. To accomplish this, the installer calls on external code, which reads the revised Path variable out of the registry and writes it back as an expandable string value. The external code then broadcasts a message that tells all running applications that we have changed an environment variable.

Finally, the installer launches Multipad.exe from a Run Application action. It does this without specifying the path to Multipad.exe, in order to test our example.

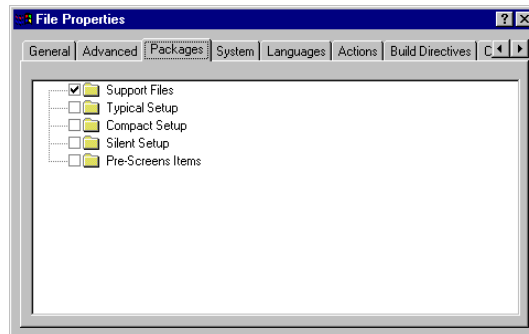
To create this installer, we first used an Add Files action to add the file “EditNTPath.dll” to the archive. This is an external code file that we will call from the installer.

From the main Installer VISE window, double-click the first action item to display its properties.



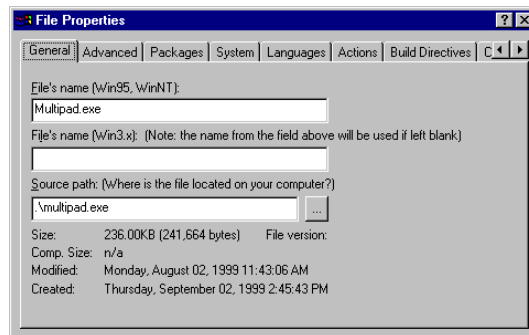
**Illustration 9-1: File Properties General Tab**

Click the **Packages** tab. We linked this file to the Support Items package. The installer will copy this file and any others in the Support Files package to a temporary folder on the user's hard drive. It will delete that folder and its contents when the installation concludes.



**Illustration 9-2: File Properties Programs Tab**

The second action item installs Multipad.exe to the target directory. The user will determine this directory location at runtime when the installer displays the Select Install Directory screen. Double-click this item to view its properties.

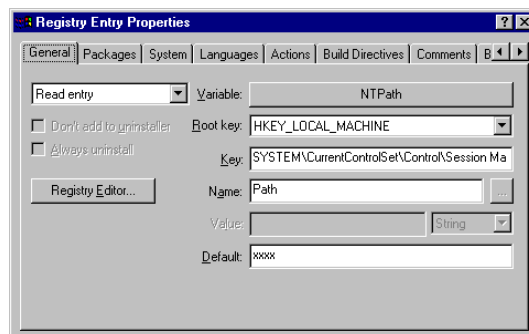


**Illustration 9-3: File Properties General Tab**

Next, a Read Registry entry reads the environment variable Path from the registry and saves the result in the runtime variable "NTPath." It finds the Path variable in Root Key: HKEY\_LOCAL\_MACHINE and Key: SYSTEM\CurrentControlSet\Control\Session Manager\Environment. This action also sets the default value of NTPath to "xxxx."

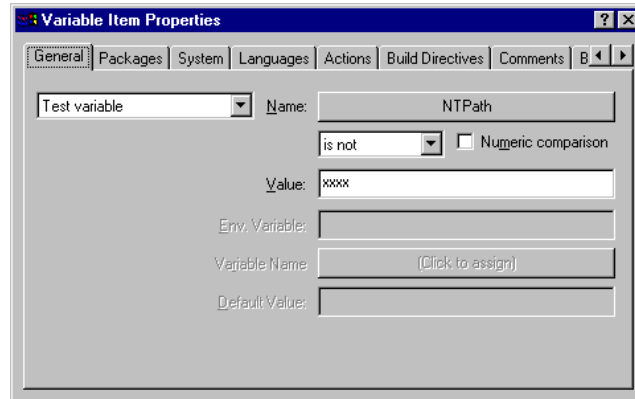
**Note:** NTPath is a runtime variable - a placeholder that will contain a value when the installer runs on the customer's computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

Double-click this action item to display its properties.



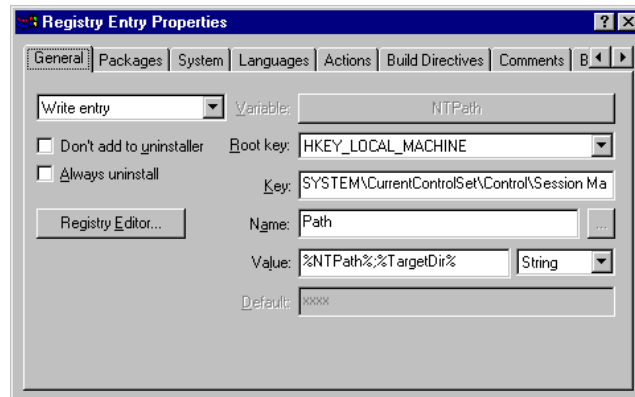
**Illustration 9-4: Read Registry Entry Properties General Tab**

Now we use a Test Variable action to determine if the installer found a value for NTPath. Double-click this item to view its properties.



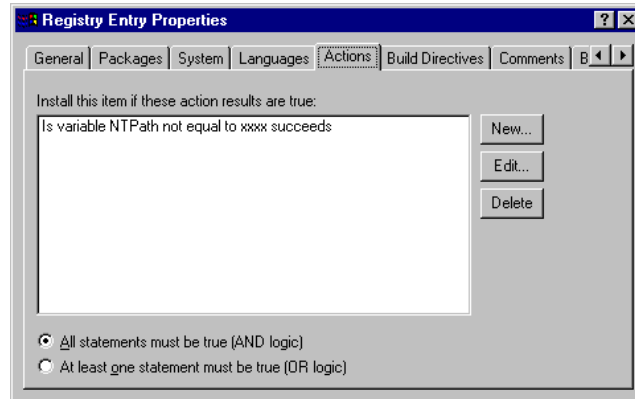
**Illustration 9-5: Variable Item Properties General Tab**

If our test shows that NTPath contains a value, we will append that value to %TargetDir%, which is the path to our current directory. We use a Write Registry Entry item to assign Path the value of %NTPath%;%TargetDir%, and then write Path back to the registry. Double-click this action item to display its General properties.



**Illustration 9-6: Write Registry Entry Properties General Tab**

Click on the **Actions** tab. This specifies that we will only write Path back to the registry if NTPath contains a value.

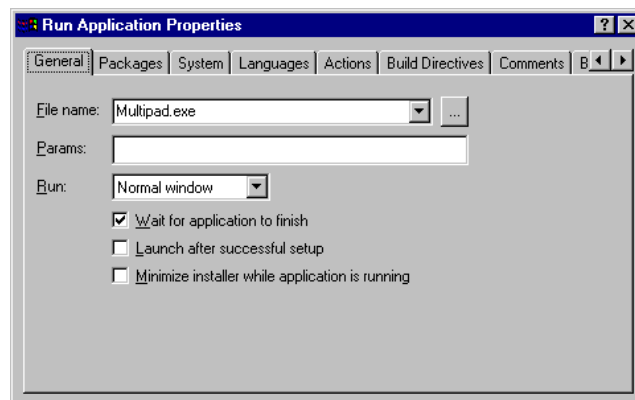


**Illustration 9-7: Write Registry Entry Properties Actions Tab**

We then use a Call External Code action to call EditNTPath.dll. This code will reset the registry entry to a type of REG\_EXPAND\_SZ so that Windows can expand the variable %SystemRoot% correctly. The code also notifies all other running applications of the change.

Like the action that precedes it, we will only execute the Call External Code action if NTPath contains a value.

Finally, the installer launches Multipad.exe. This will work if we successfully edited the NT path. Double-click the last action item in our archive to display its General properties. Notice that we did not specify the path to the application. Before we edited the NT path, we would have needed to specify the path in the "File name" or "Params" fields.



**Illustration 9-8: Run Application Properties General Tab**

---

## Example 10

# Plug-ins Example

---

### Plug-ins Example

The Plug-ins example demonstrates how to install a plug-in for Internet Explorer and Netscape Navigator.

### Features Demonstrated

**Look at this example to see how to use:**

- Read Registry Entry actions
- Set Variable actions
- Test Variable actions
- Custom Screen actions
- Shadow files

### How It Works

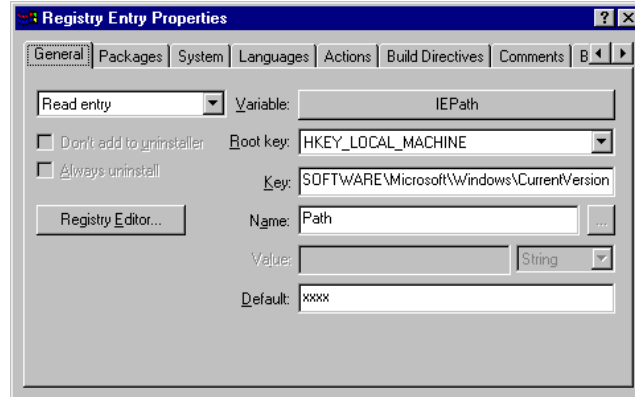
This installer checks the target computer's registry for the existence of Internet Explorer, and then for Netscape Navigator. If it finds both applications, the installer displays a dialog box that prompts the user to choose which application(s) will receive the plug-in. If it finds only one of the applications, the installer won't display the dialog. It then installs the plug-in "myplugin.dll" to the appropriate "Plugins" folder(s) at the location(s) it found in the registry.

The first Read Registry entry reads the following App Path entry:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\iexplore.exe\Path. It stores the result in the variable "IEPath" and sets the default value to "xxxx."

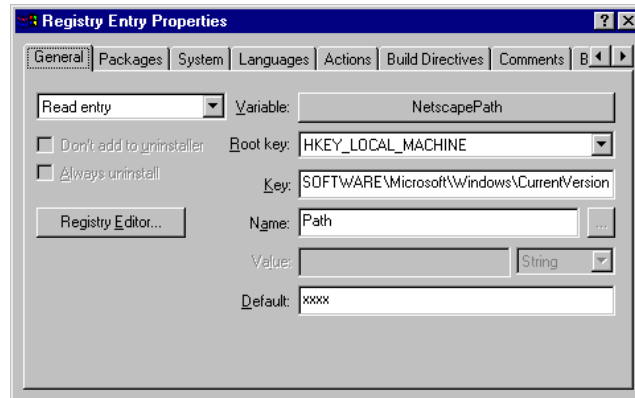
**Note:** IEPath is a runtime variable - a placeholder that will contain a value when the installer runs on the customer's machine (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

From the main Installer VISE window, double-click the first action to display these settings.



**Illustration 10-1: Registry Entry Properties General Tab**

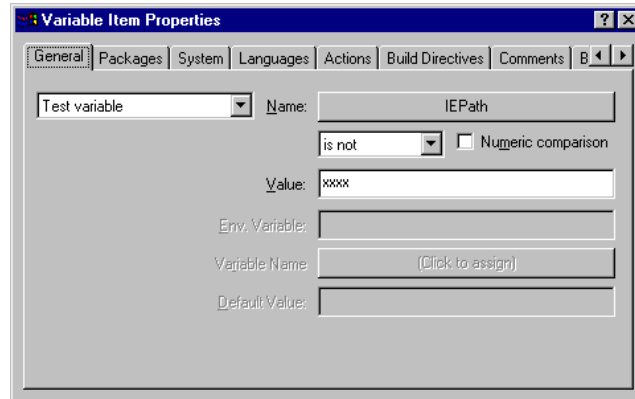
The next Read Registry Entry uses a similar path to determine the existence of Netscape Navigator: HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\netscape.exe\Path. It stores the result in the variable “NetscapePath” and sets the default value to “xxxx.” To view these settings, double-click the second action item in our archive.



**Illustration 10-2: Registry Entry Properties General Tab**

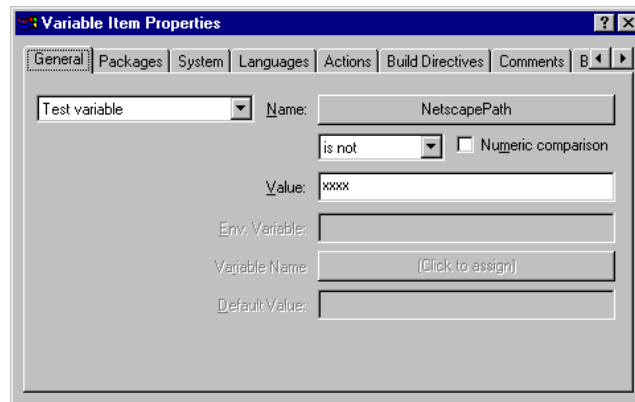
The installer then uses two Test Variable actions to determine whether IEPath and NetscapePath contain values.

The first Test Variable action tests the variable IEPATH for the value “xxxx.” Double-click the action to display its properties.



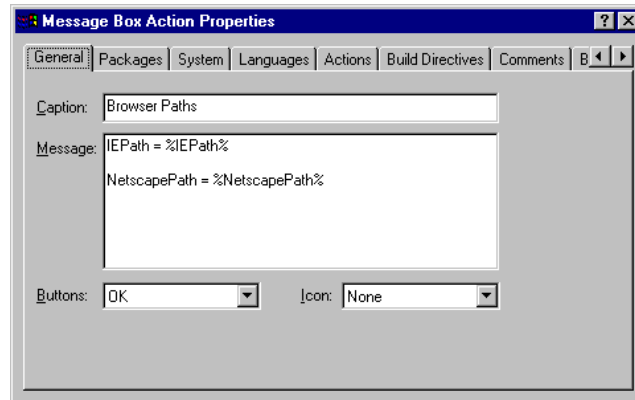
**Illustration 10-3: Test Variable Item Properties General Tab**

The second Test Variable action tests the variable NetscapePath for the value “xxxx.” Double-click the action to display its properties.



**Illustration 10-4: Test Variable Item Properties General Tab**

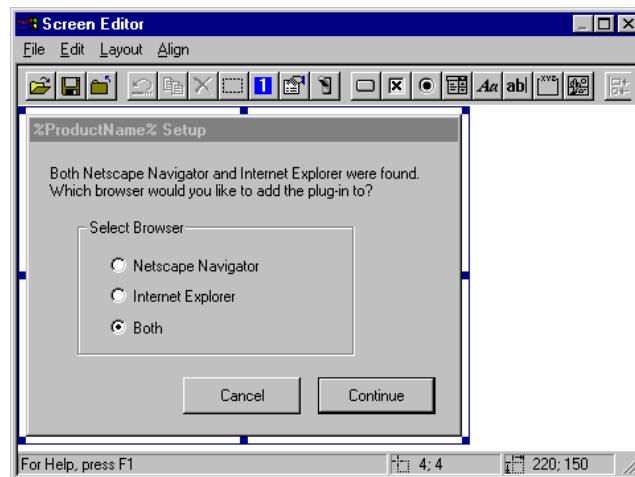
Next in our archive is a Message Box action. Check the box to the left of this item if you want to display the values of IEPATH and NetscapePath to the user. Double-click on the action to view its properties.



**Illustration 10-5: Message Box Action Properties General Tab**

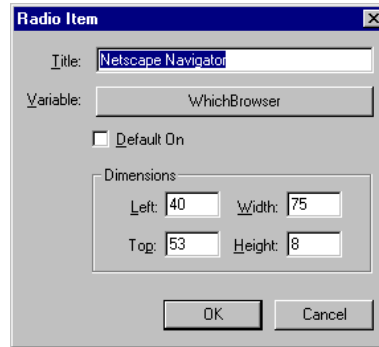
If the installer finds both Internet Explorer and Netscape Navigator on the target computer, it then uses a Custom Screen action to prompt the user to choose whether to install the plug-in for one or both of these applications

Double-click the Custom Screen item to display its properties. Now click on **Edit Screen**. Here in the Screen Editor, we set up three radio buttons - one for "Netscape Navigator," one for "Internet Explorer," and one for "Both."



**Illustration 10-6: Screen Editor Dialog**

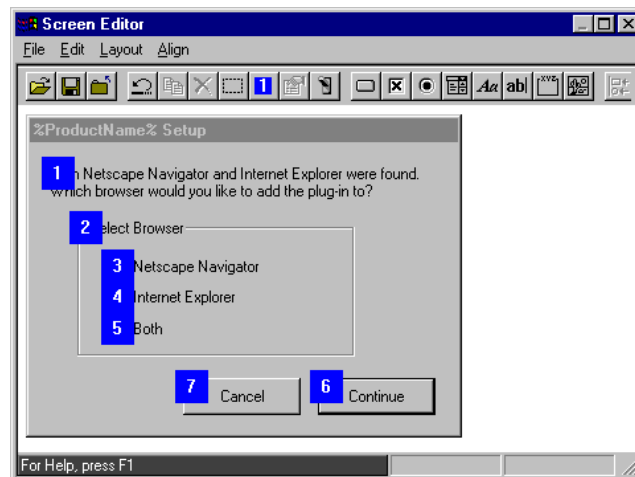
Each of these radio buttons will store the user’s input into the variable “WhichBrowser.” Double-click on one of the buttons to view this setting.



**Illustration 10-7: Radio Item Dialog**

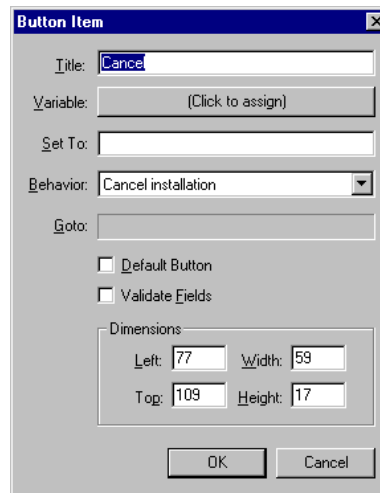
When a user clicks one of these radio buttons, the value returned for WhichBrowser depends upon the button’s tab order. In this case, clicking “Netscape Navigator” sets WhichBrowser to a value of one, while clicking “Internet Explorer” sets WhichBrowser to two, and clicking “Both” sets WhichBrowser to three.

To view the tab order, close the Radio Item dialog and then select **Set Tab Order** from the **Edit** menu.



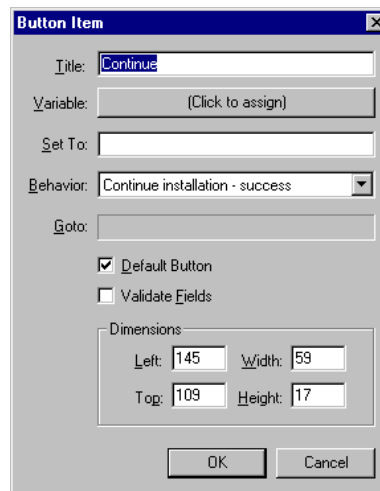
**Illustration 10-8: Screen Editor Set Tab Order Dialog**

Double-click the “Cancel” button. Notice that in the “Behavior” field we chose “Cancel installation.”



**Illustration 10-9: Button Item Dialog**

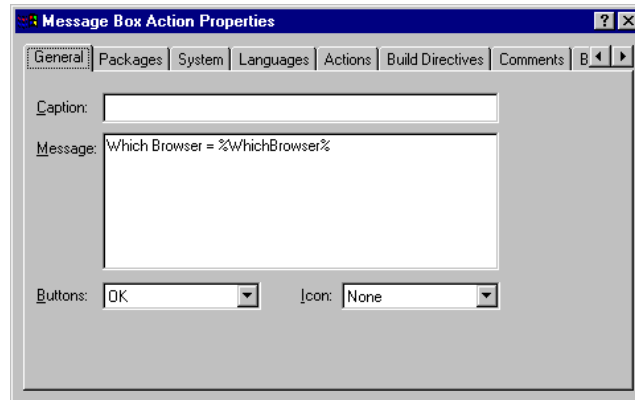
Now double-click the “Continue” button. Notice that in the “Behavior” field we chose “Continue installation - success.”



**Illustration 10-10: Button Item Dialog**

Next in the archive is an optional Message Box that will display the value of Which-Browser to the user. This item will only display if you click the box to the left of it, and our Custom Screen succeeds (the user makes a selection).

Double-click this action item to display its properties.

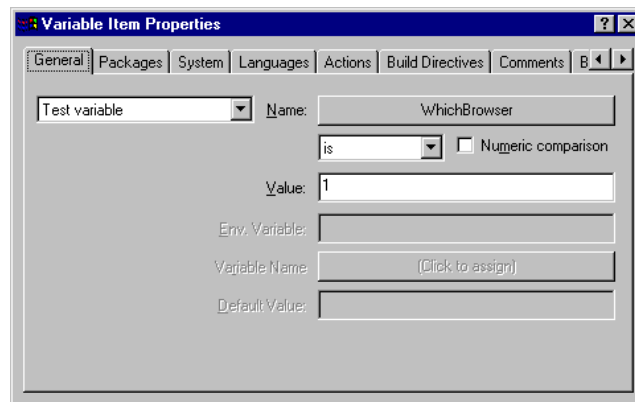


**Illustration 10-11: Message Box Action Properties General Tab**

Now we use four variable actions to determine what application(s) the user selected to receive the plug-in. We linked the Test Variable actions to the success of Custom Screen, and linked the Set Variable actions to the results of the corresponding Test Variable actions.

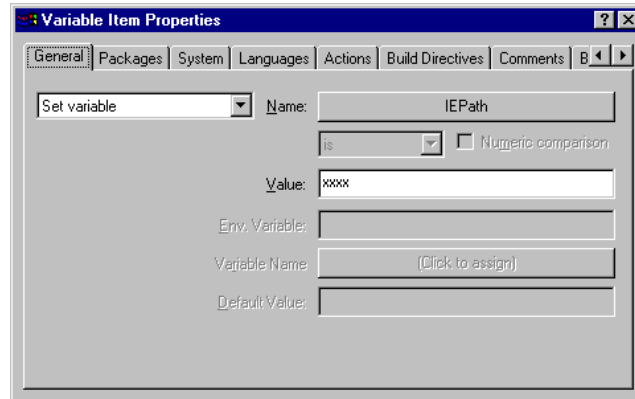
If the user selected “Netscape Navigator,” (WhichBrowser is equal to 1) we will set the variable IEPATH to a known value, “xxxx.” If the user selected “Internet Explorer,” (Which-Browser is equal to 2) we will set the variable NetscapePath to “xxxx.”

Double-click the action item “Is WhichBrowser equal to 1” to display its properties.



**Illustration 10-12: Variable Item Properties General Tab**

Now double-click the next action item. This will set IEPATH to “xxxx” if WhichBrowser is equal to 1.

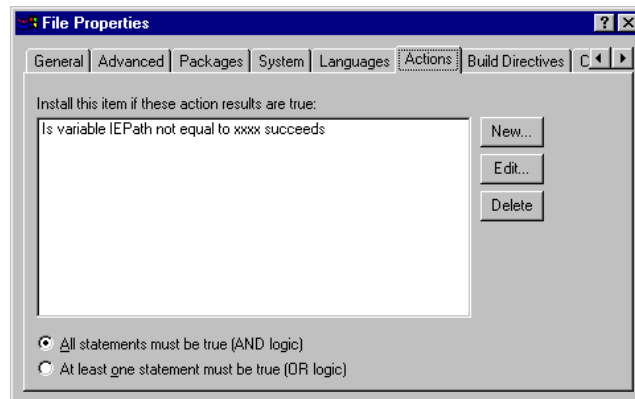


**Illustration 10-13: Set Variable Item Properties General Tab**

When these Variable items are complete, we install the plug-in.

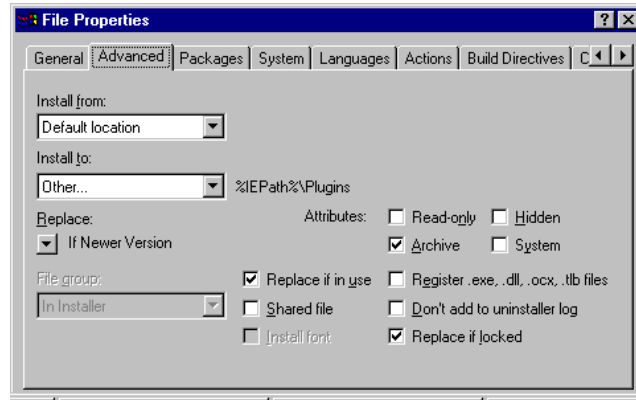
First, we test the variable IEPATH to determine that it is not equal to “xxxx.” If this test succeeds, we install myplugin.dll to the appropriate Plugins folder.

Double-click on the first myplugin.dll and then click the **Actions** tab. Note that we will only install this action item if IEPATH contains a value.



**Illustration 10-14: File Properties Actions Tab**

Next, click the **Advanced** tab. Here we set the install location to %IEPath%\Plugins.



**Illustration 10-15: File Properties Advanced Tab**

We use the same process to install the plug-in for Netscape Navigator, if necessary. In this case, the install location for myplugin.dll is %NetscapePath%\Plugins.

Notice that the final item in our archive, myplugin.dll, is dimmed. This indicates a shadow file. Shadow files support installing a file to more than one location. Rather than adding a file twice to your installer, which will increase the installer size, you can use shadow files to only add the file once.

You can create shadows by duplicating files in your project. To do this, use the right mouse button to click on a file, and then select **Duplicate** from the pop-up menu. Another option is to select the file and choose **Duplicate** from the **Edit** menu.

---

## Example 11

# AutoPlay Example

---

### AutoPlay Example

The AutoPlay example demonstrates how to support AutoPlay for a CD-ROM installer.

### Features Demonstrated

**Look at this example to see how to use:**

- Write Registry Entry actions
- Read Registry Entry actions
- Program Item actions
- Test Variable actions
- Run Application actions
- Stop actions

### How It Works

When the user inserts our CD, the AutoPlay installer will launch. This installer will first look to the registry to see if our application already resides on the target computer. If it finds a previous install, the installer will launch the application at the location of the previous install. If it doesn't find a previous install, it will launch the application installer from the CD.

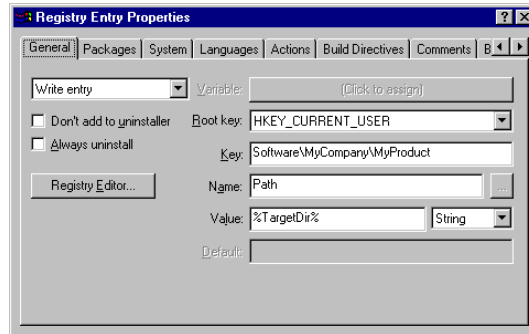
This example consist of three parts: the application installer, the AutoPlay installer, and the autoplay.inf file.

First, we'll cover the application's installer, "MyProductsInstaller.VCT." This installer consists of just three action items.

The application installer uses a Write Registry entry to write the application's install path to the registry. It writes the variable "Path" to root Key: HKEY\_CURRENT\_USER and key: Software\MyCompany\MyProduct. We set the value of Path to be equal to %TargetDir%.

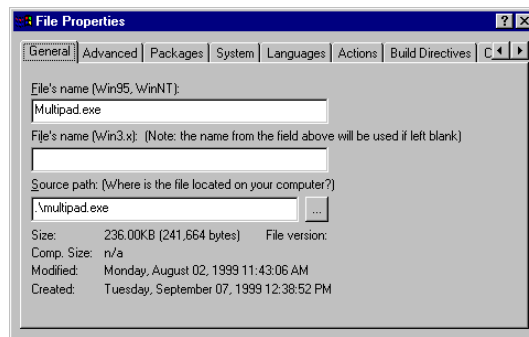
**Note:** %TargetDir% is a runtime variable - a placeholder that will contain a value when the installer runs on the customer's computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

Double-click on the Write Registry Entry action item to display its properties.



**Illustration 11-1: Write Registry Entry Properties General Tab**

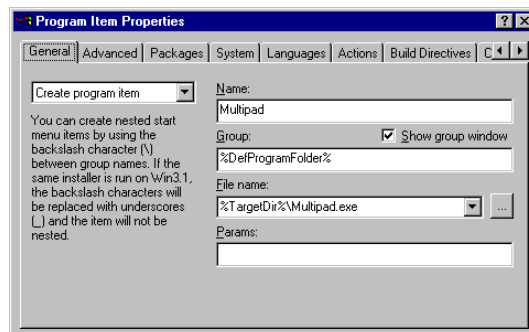
After it makes the registry entry, the installer installs our application (Multipad.exe) to the target directory. Double-click the second action item to display these settings.



**Illustration 11-2: File Properties General Tab**

Next, this installer uses a Program Item action to create an entry for the application in the “Programs” folder of the customer’s computer.

The “Group” field names the program group where we’ll create the item. In this field, we entered %DefProgramFolder%. This variable holds the default program folder’s name. The “File name” field names the application we will install. Double-click the Program Item action to display these settings.

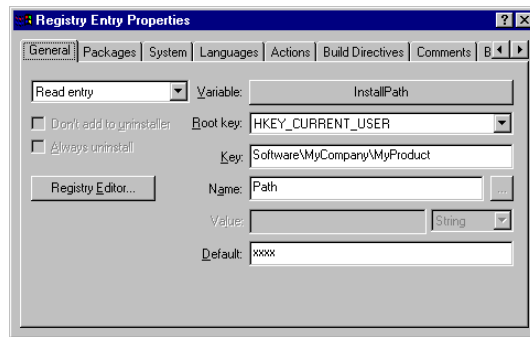


**Illustration 11-3: Program Item Properties**

Next, we'll look at the AutoPlay installer, "AutoPlay.VCT." This executable doesn't have an interface and it doesn't install anything. It uses five action items. The first four action items are part of the Pre-Screens Package, so the installer will execute them before it displays any screens to the user.

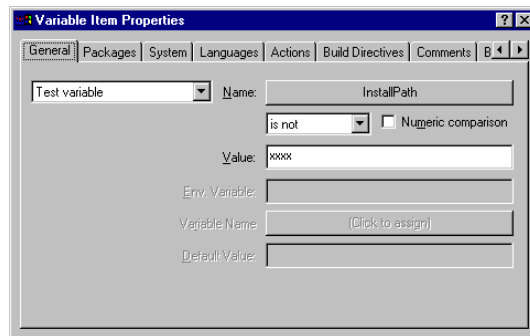
First, the AutoPlay installer reads the registry to determine if our application already resides on the target computer. It reads the entry "Path" from root key: HKEY\_CURRENT\_USER and key: Software\MyCompany\MyProduct. We store the result in the variable "InstallPath." If we don't find the needed registry information, we set the default value of Path to "xxxx."

Double-click this action to display its properties.



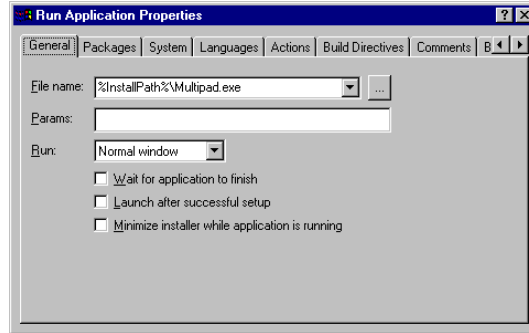
**Illustration 11-4: Read Registry Entry Properties General Tab**

Next, we use a Test Variable action to determine whether the InstallPath variable contains a value. Double-click this action to display its properties.



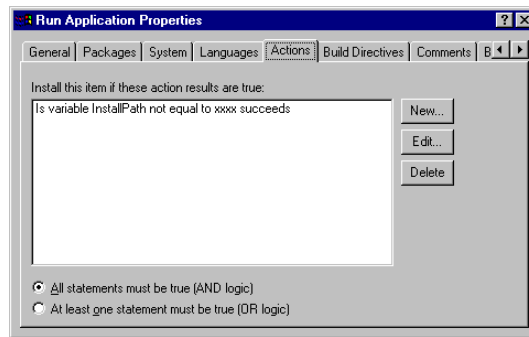
**Illustration 11-5: Test Variable Item Properties General Tab**

If we find the path to our application, we use a Run Application action to launch the program. Double-click this action to display its properties.



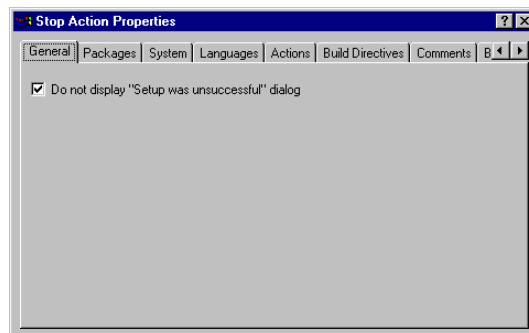
**Illustration 11-6: Run Application Properties General Tab**

We will only install this action if InstallPath contains a value. Click the Actions tab to view this setting.



**Illustration 11-7: Run Application Properties Actions Tab**

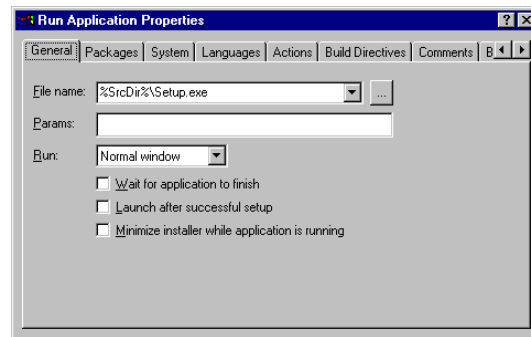
At this point, we will end the installation if InstallPath contains a value. For this, we use a Stop action. Double-click this action to display its properties.



**Illustration 11-8: Stop Action Properties**

However, if the variable test determines that our application is not resident on the customer's computer, we will launch an installer to put it there.

Double-click the final action item. This Run Application action launches “Setup.exe” in %SrcDir%. The %SrcDir% variable contains the path name of the drive from which we launched the installer. Note that this item is part of the Typical Setup package.



**Illustration 11-9: Run Application Properties General Tab**

To make the Autoplay installer invisible to the user, we used the following settings:

- From the **File** menu, select **Installer Properties**. Here we checked “Hide ‘Starting Setup. Please wait...’ dialog.” This prevents the installer from showing the startup dialog. Next, click the **Uninstaller** tab. Here we unchecked “Include uninstaller.”
- We unchecked all screens listed under the **Screens** menu.
- From the **Screens** menu, select **Background Window**. Here we checked “Hide progress dialog.”

Finally, we have the autoplay.inf file. This is a simple text file that must reside on the root directory of our CD. Here is the format:

- [autorun]
- open=autorun.exe
- icon=multipad.exe

The “open” key is the name of the file to launch. The “icon” key is the name of a file that contains the icon for the CD.

When we create this CD, we would put Setup.exe, Autoplay.exe, and Autoplay.inf on the root directory of the CD.

## Example 12

# Adding Files to Uninstall Log Example

---

## Adding Files to Uninstall Log Example

This example demonstrates how to add a file entry to the uninstall log file. Your installer will automatically make a file entry for files it installs, but there may be instances where you want to remove a file that the installer didn't create. These could be files the application creates after the installer runs, such as a preferences file.

### Features Demonstrated **Look at this example to see how to use:**

- Read Ini File Entry actions
- Write Ini File Entry actions
- Increment Variable Item actions

### How It Works

This installer will make the file "Settings.dat" available for uninstall when the user chooses to remove our application.

First, the installer reads the current file count from the uninstal.log file. Next, it writes the full path name of Settings.dat to the uninstal.log file. The installer then increments the file count and writes the file count back to the uninstal.log file.

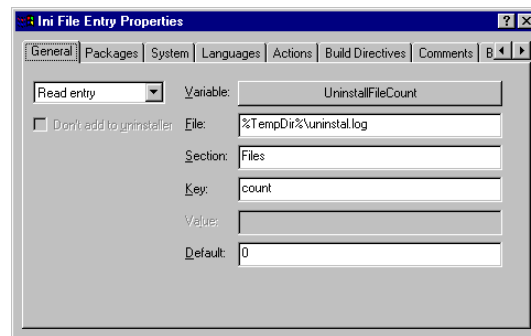
The uninstal.log file uses an .ini type format. Files appear under the section [Files]. The key "count" is the total number of entries in this section. The other keys are numbered 0 to count minus 1 and their values are the full path name of the files to uninstall.

Our example installer uses four action items to accomplish its goal.

First, the Read Ini File Entry action reads the current file count. From the main Installer VISE window, double-click this action item to display its properties. Note the following settings:

- Variable = UninstallFileCount
- File = %TempDir%\uninstal.log
- Section = Files
- Key = count
- Default = 0

**Note:** UninstallFileCount and %TempDir% are runtime variables - placeholders that will contain a value when the installer runs on the customer's computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

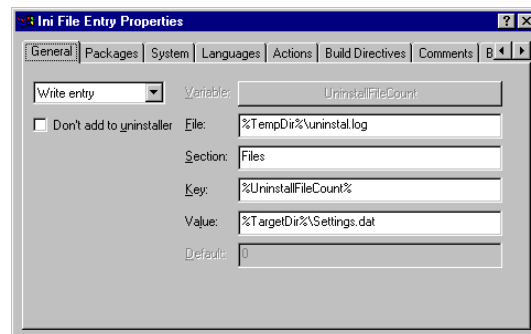


**Illustration 12-1: Read Ini File Entry Properties General Tab**

Next, we use a Write Ini File Entry action to record the full path name of the file “Settings.dat” in the uninstal.log. For this action, we used the following settings:

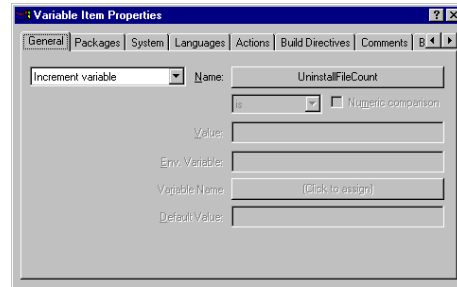
- File = %TempDir%\uninstal.log
- Section = Files
- Key = %UninstallFileCount%
- Value = %TargetDir%\Settings.dat

Double-click the second action item to view these settings.



**Illustration 12-2: Write Ini File Entry Properties General Tab**

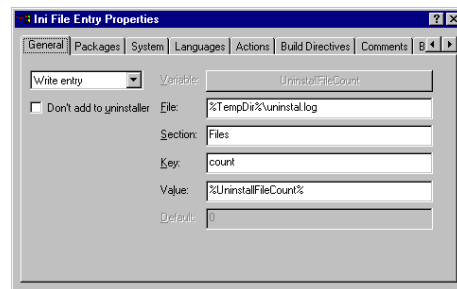
Next, we increment the value of the UninstallFileCount variable. Double-click the Increment Variable Item action to view these settings.



**Illustration 12-3: Increment Variable Item Properties General Tab**

Finally, we write the updated file count back to the uninstal.log file. We used the following settings for the Write Ini File Entry:

- File = %TempDir%\uninstal.log
- Section = Files
- Key = count
- Value = %UninstallFileCount%



**Illustration 12-4: Write Ini File Entry Properties General Tab**

## Example 13

# Does Registration Key Exist Example

---

## Does Registration Key Exist Example

This example demonstrates how to determine if a registry key exists. It does this by calling external code. The external code returns 1 if the key exists and returns 0 if it doesn't.

## Features Demonstrated **Look at this example to see how to use:**

- Call External Code actions

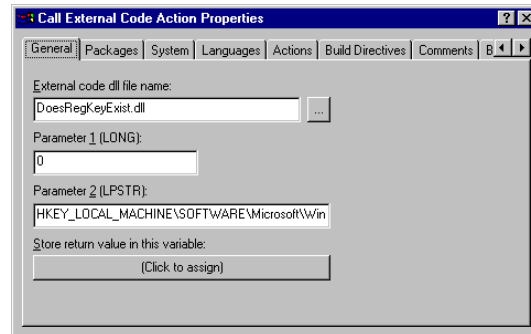
## How It Works

The external code uses the registry key's name as a parameter. It then tries to open the key. If it successfully opens the key, or if the procedure fails and returns an `ERROR_ACCESS_DENIED` error, the code returns a 1. If the code can't open the key, it returns a 0. The source for the external code is in the file `DoesRegKeyExist.cpp`.

We accomplished this goal with just two items, the external code DLL file "Does-RegKeyExist.dll" and a Call External Code action. We excluded the external code file from all packages except "Support Files."

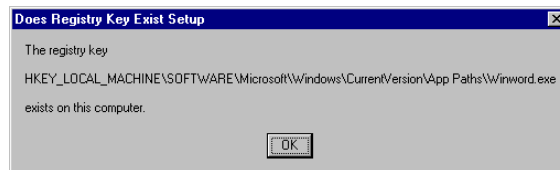
The Call External Code uses the following settings:

- External code dll file name = DoesRegKeyExist.dll
- Parameter1 = 0
- Parameter2 = HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\Winword.exe



**Illustration 13-1: Call External Code Action Properties General Tab**

If our specified registry key exists, the installer will display a message box to the user. It will display a similar message box in the event that the registry key does not exist. The affirmative message is below.



**Illustration 13-2: Message Box**

## Example 14

# Active Web Install Example

---

## Active Web Install Example

This example demonstrates how to create an Active Web Installer.

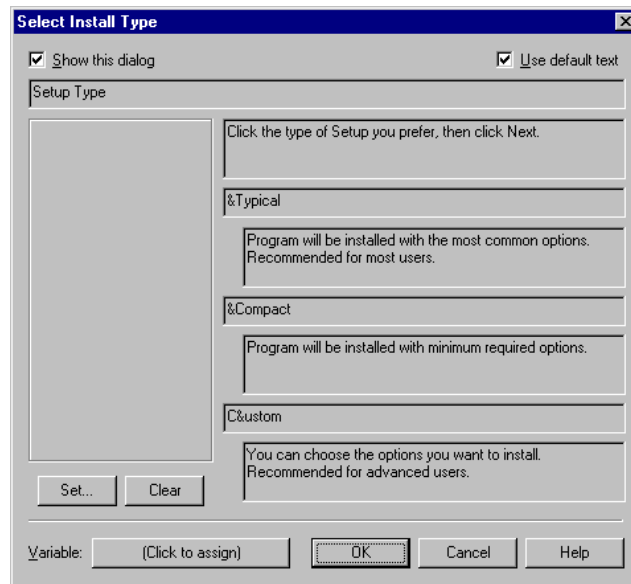
### Features Demonstrated **Look at this example to see how to use:**

- Select Install Type screens
- Download Sites
- File Groups
- Build Targets

### How It Works

This installer consists of three packages: Main, Documentation and Examples. We placed the files for these packages into three file groups: Main, Docs and Examples. We will store these file groups on two servers: an Ftp server and an Http server. We associated each file group with a package so the installer will only download a file group if it installs the associated package.

From the **Screens** menu of ActiveWebInstall.VCT, choose **Select Install Type**. This dialog gives the user the choice of Typical, Compact, and Custom setups. Note that we checked “Show this dialog.”



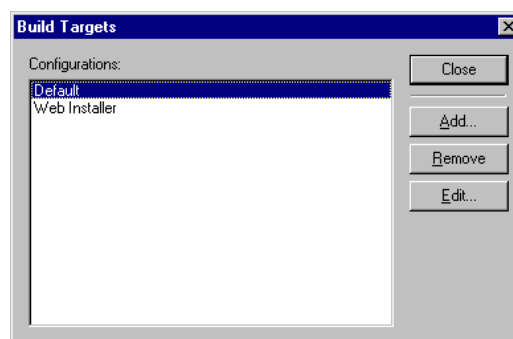
**Illustration 14-1: Select Install Type Window**

From the **Internet** menu, select **Download Sites**. This displays a list of servers we’re using. If more than one server appears here, the user will get to choose a server at install time.

Now choose **File Groups** from the **Internet** menu. This displays a list of group files. We create these files at build time.

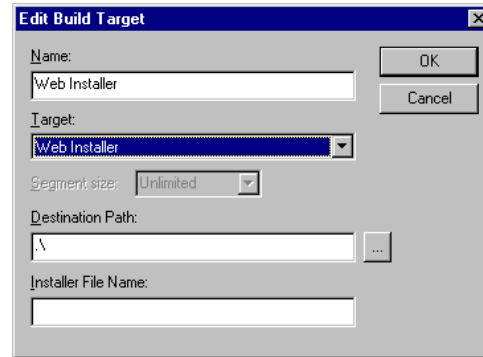
Group files consist of compressed file data. These files are what we upload to our servers. We download entire file groups from the server, rather than individual file data. For this reason, you should structure your installer so that the files it installs together are grouped in the same file groups.

Go to the **Archive** menu and choose **Build Targets**. In this window, we clicked **Add** and named a new build target, “Web Installer.”



**Illustration 14-2: Build Targets Window**

Now double-click on Web Installer to display the Edit Build Target dialog. Note that we selected “Web Installer” in the Target field.



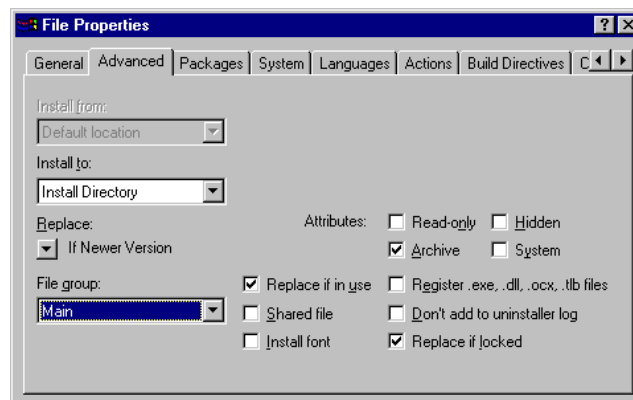
**Illustration 14-3: Edit Build Target Window**

Since we added a Build Target, the drop-down menu in our archive contains two items: Default and Web Installer. Note that we selected Web Installer. This specifies Web Installer as the build target.

This installer contains six files with the following settings:

- notepad.exe - Typical, Compact, and Main packages - Main file group
- notepad.hlp - Typical and Documentation packages - Docs file group
- notepad.cnt - Typical and Documentation packages - Docs file group
- example2.txt - Typical and Examples packages - Examples file group
- example1.txt - Typical and Examples packages - Examples file group
- example3.txt - Typical and Examples packages - Examples file group

**Note:** To assign a file to a file group, double-click on the file from the archive to display File Properties. Then click the **Advanced** tab. Make a selection from the “File group” pull-down menu, as illustrated below.



**Illustration 14-4: File Properties Advanced Tab**

When we build this installer (select **Build Installer** from the **File** menu), it creates six files:

- ActiveWebInstall.exe
- viseicat.cat
- viseicat.idx
- main
- docs
- examples

We will need to upload all these files except ActiveWebInstall.exe to our server(s). All these files must remain together. Whenever we create a new build we must re-post these files to our server(s). ActiveWebInstall.exe is the file we distribute to our users.

If the user selects a Typical setup, the installer will download all three group files. If the user selects a Compact install, the installer will only download the Main group file. If the user selects a Custom install, the installer will only download the group files that correspond to the selected packages.

When the installer downloads the data it stores the data on the local disk with the greatest amount of free disk space. It creates a folder on that drive with the name “%ProductName% Download Data” (at runtime, the installer will replace %ProductName% with the name of our product). Then the installer creates the file %ProductName%.dat in the folder.

**Note:** %ProductName% is a runtime variable - a placeholder that will contain a value when the installer runs on the customer’s computer (when you want to use a value stored in a runtime variable, use percent signs before and after the name; leave them off when you assign a value).

When this installer runs, it will look through the root folders on each local drive for a file cache it can use. If it finds one, it will use the locally stored data from the file cache rather than get it from the Internet. If the installer doesn't need to get any file data from the Internet it will prompt the user to choose whether to install with existing file data or check the Internet for an update. If we check the Internet for an update and find a newer installer, we will download that installer to the %ProductName% Download Data folder and then launch it.

## Example 15

# Build Targets Example

---

### Build Targets Example

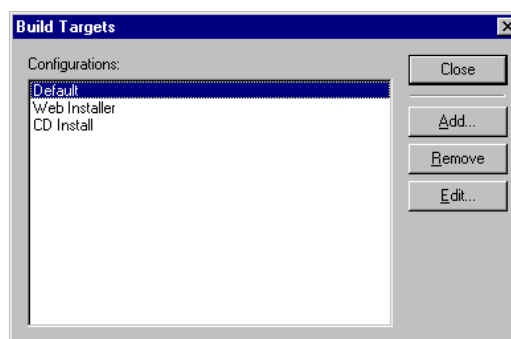
This example demonstrate the use of build targets. It is the same example as the Active Web Install example with the added ability to create three different installers from the same project file.

### Features Demonstrated **Look at this example to see how to use:**

- Batch Build
- Build Targets

### How It Works

Open BuildTargets.VCT and then select **Build Targets** from the **Archive** menu. This is where we added three build targets: Default, Web Installer and CD Install.

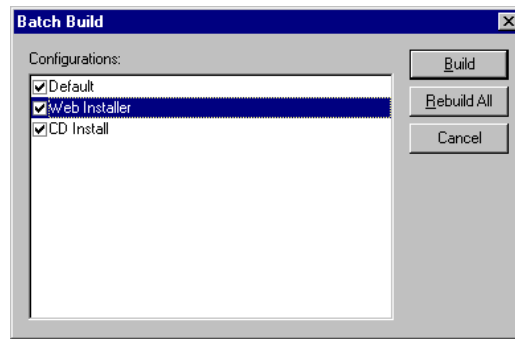


**Illustration 15-1: Build Targets Window**

- The Default build target creates a single file installer.
- The Web Installer build target creates an Active Web Installer.
- The CD Install build target creates a CD-ROM install (this option will not compress file data into the installer).

After we added the build targets, the next step was to build installers with each of them. There are two ways you can do this:

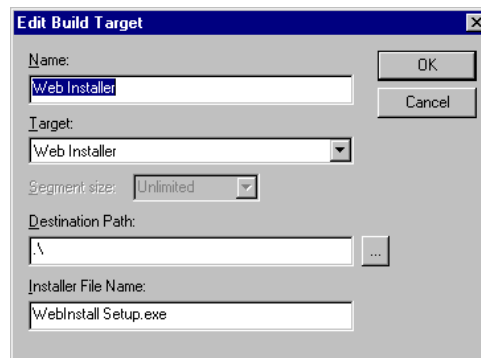
- To build a single installer, select a build target from the pull-down menu in the archive. From the **File** menu, select **Build Installer**. Repeat this process for additional build targets.
- To build multiple installers all at once, use the Batch Build feature. From the **File** menu, select **Batch Build**. Here you have two options to specify which installers to build (or which installers to rebuild if you change their build target). You can place a check mark next to your selection(s) and click **Build**. Or you can click **Rebuild All** to rebuild all the installers.



**Illustration 15-2: Batch Build Window**

Batch Build will build the installers to the locations specified for each build target. To change this, go to the **Archive** menu and choose **Build Targets**. Double-click one of the build targets, and edit the “Destination Path” and “Installer File Name” fields.

In the illustration below, notice that we used “.\” in the “Destination Path” field. This will force the builder to create the installer at the location of the open .vct file.



**Illustration 15-3: Edit Build Target Window**

## Example 16

# Message Dialog Example

## Message Dialog Example

The Message Dialog example demonstrates how to display a small, modeless dialog during an installation.

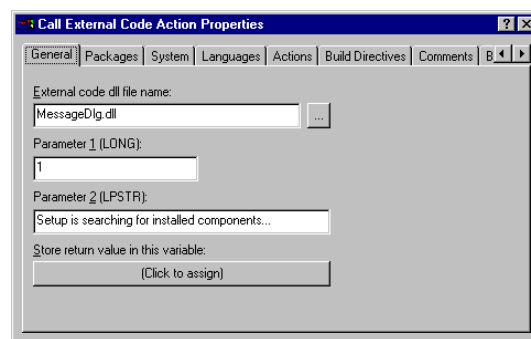
### Features Demonstrated **Look at this example to see how to use:**

- Call External Code actions

### How It Works

The dialog handling code is in the file “MessageDlg.dll.” We added this file to the archive and put it in the Support Files package. The source code for this file is in the project MessageDlg. We’ve included all the project files in the folder called “MessageDlg,” so you can modify them if you wish.

This installer uses a Call External Code action to display a message. From the main Installer VISE window, double-click the first Call External Code action to display its properties.



**Illustration 16-1: Call External Code Action Properties General Tab**

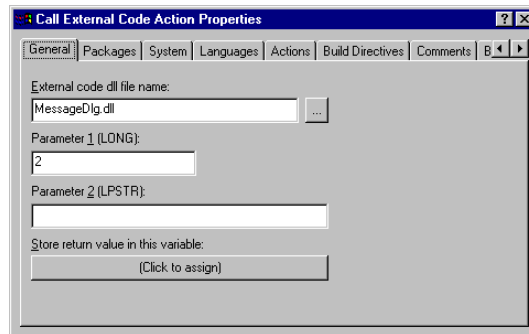
Note that we entered “1” for Parameter 1. This displays the dialog. Parameter 2 contains the text we want the message to display.

Next, the installer uses a Message Box action to display a Message Box. This action simply pauses the installer so that it will display our primary dialog until the user clicks “OK.”



**Illustration 16-2: Message Box**

To close our message, we used another Call External Code action. This calls the same code used to display the message, but uses different parameters. Double-click this action item to display its properties.



**Illustration 16-3: Call External Code Action Properties General Tab**

Here we entered “2” for Parameter 1. This closes the dialog. We left Parameter 2 blank, because we won’t display text in this instance.

# Index

---

## Symbols

%DefProgramFolder% 11-2  
 %ProductName% 4-1 to 4-2, 5-2, 8-2, 14-4  
 %ProgramFilesDir% 1-2 to 1-3, 4-1, 5-2  
 %SrcDir% 11-5  
 %SystemDir% 5-3  
 %SystemRoot% 9-1, 9-5  
 %TargetDir% 2-1, 4-1, 9-4, 11-1  
 %TempDir% 6-3, 12-2  
 %UninstallString% 6-3  
 %WinDir% 4-1, 4-5, 5-2  
 .BAT file name extension 6-3  
 .ini type format 12-1

## A

Active Web Installers 1-1, 14-1  
 Add Files actions 4-1, 4-3, 8-6, 9-1 to 9-2  
 Add Folder actions 5-1  
 Add Items menu 4-3  
 Add/Remove Programs 6-1  
 AND logic 7-4 to 7-5  
 Archive menu 14-2, 15-1 to 15-2  
 AutoPlay 11-1, 11-3

## B

Background Window 11-5  
 Batch Build 15-2  
 Batch files 6-1, 6-3 to 6-4  
 Build Installer 14-4, 15-2  
 Build Targets
 

- Add a new build target 14-2
- Build Targets 14-1 to 14-2, 15-1 to 15-2

## C

Call External Code actions 9-1 to 9-2, 9-5, 13-1 to 13-2, 16-1 to 16-2  
 CD-ROM installer 11-1  
 Command line 6-1  
 Compact setup 14-2  
 Custom packages 7-1  
 Custom Screen 10-7  
 Custom Screen actions 10-1, 10-4, 10-7  
 Custom Setup 7-1, 14-2

## D

Default install directory 1-2, 4-2  
 Desktop Folder 5-2 to 5-3  
 DisplayName string 6-1  
 DOS commands 6-3  
 Download Sites 14-1 to 14-2  
 Duplicate files 10-9

## E

Edit Screen 10-4  
 Edit Text File actions 6-1, 6-3  
 Environment variables 9-1  
 ERROR\_ACCESS\_DENIED error 13-1

## F

File groups 14-1 to 14-3  
 File menu 4-2, 11-5, 14-4, 15-2  
 File name field 9-5, 11-2  
 Find action 1-1, 2-2  
 Find actions 1-1 to 1-2, 1-4 to 1-5, 2-1, 2-3  
 Finished Message screen 4-1, 4-5

**G**

Goto actions 7-1 to 7-2, 7-5, 8-1 to 8-2, 8-5

**H**

Hide 'Starting Setup. Please wait...' dialog 11-5

Hide progress dialog 11-5

**I**

Icon key 11-5

Include uninstaller 11-5

Increment Variable Item actions 12-1, 12-3

Information icon 2-3

Input Box actions 7-1 to 7-2, 7-5, 8-1 to 8-2, 8-5

Install File actions 2-4

Installer File Name field 15-2

Installer Properties 4-1 to 4-2, 5-2, 8-2, 11-5

Internet Explorer 10-1

Internet menu 14-2

**K**

Key 3-2, 6-1 to 6-2, 9-3, 11-3

Key code 7-1 to 7-3, 8-1

**L**

Label actions 7-1 to 7-2, 8-1 to 8-2

Label field 8-5

**M**

Message Box actions 2-1, 2-3, 7-1, 7-4, 8-1, 8-4, 10-4, 10-7, 16-2

Move actions 2-1 to 2-4

**N**

Netscape Navigator 10-1

**P**

Packages 7-1, 7-5, 14-1

Parameter 13-1

Parameter 1 16-1 to 16-2

Parameter 2 16-1 to 16-2

Params field 4-5, 6-4, 9-5

Percent signs 1-2, 2-1, 3-2, 4-1, 5-2, 6-2, 7-2, 8-2, 9-3, 10-1, 11-1, 12-2, 14-4

Plug-in 10-1, 10-8

Preferences file 12-1

Pre-Screens Items package 1-1, 1-3, 1-5, 3-2, 7-5, 8-6, 11-3

Product name field 4-1, 8-2

Program Files folder 1-2

Program group 11-2

Program Item actions 11-1

Programs folder 11-2

Properties dialog

Actions tab 1-3, 2-3 to 2-4, 6-3 to 6-4, 7-4 to 7-6, 8-5 to 8-6, 9-5, 10-8, 11-4

Advanced tab 4-3 to 4-4, 5-2 to 5-3, 10-9, 14-3

General tab 1-2, 4-3 to 4-4, 6-2 to 6-4, 7-2, 7-5, 9-4 to 9-5

Packages tab 1-1, 9-2

**R**

Radio buttons 10-4 to 10-5

Radio Item dialog 10-5

Read Ini File Entry actions 12-1 to 12-2

Read Registry Entry actions 3-1 to 3-4, 6-1 to 6-2, 9-1, 9-3, 10-1 to 10-2, 11-1

Readme file 4-1

Registration Information screen 3-1, 3-4, 3-6

Registry 1-2, 3-2, 4-1, 5-2, 6-1, 9-1, 9-3, 9-5, 10-1, 11-1, 11-3

Registry key 13-1

Replace file 2-2

Root directory 11-5

Root key 6-2, 9-3, 11-3

Run Application actions 6-1, 6-4, 9-1 to 9-2, 11-1, 11-4 to 11-5

Runtime variables 1-2, 2-1, 3-2, 4-1, 5-2, 6-2, 7-2, 8-2, 9-3, 10-1, 11-1, 12-2, 14-4

**S**

Screen Editor 10-4

Screens menu 1-1, 1-3, 3-1, 4-2, 4-5, 5-2, 8-6, 11-5, 14-2

Select Components screen 7-1

Select Install Directory screen 1-1, 1-3 to 1-4, 4-1 to 4-3, 5-2, 9-3

Select Install Type screen 7-1, 14-1 to 14-2

Set Tab Order 10-5

Set Variable actions 1-2 to 1-3, 10-1, 10-7

Shadow files 10-1, 10-9

Shortcut actions 5-1

Show this dialog 3-2, 4-5, 14-2

Stop actions 11-1, 11-4

String value 9-1

Sub-key 6-1

Support Files package 9-2, 13-1, 16-1

**T**

Target directory 1-2, 2-1 to 2-2, 2-4, 4-1, 9-1, 9-3, 11-2

TargetDir 1-2 to 1-3

Test Variable Item actions 6-1 to 6-2, 7-1, 7-3, 8-1, 8-3, 9-1, 9-4, 10-1 to 10-3, 10-7, 11-1, 11-3

Typical Setup package 3-4, 8-6, 11-5, 14-2

**U**

Uninstall 6-4

Uninstall log file 12-1

Uninstaller 6-1

---

**A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**

---

Uninstaller tab 11-5  
UninstallString 6-1 to 6-2  
UserName 3-2

**V**

Variables 3-1, 8-2, 10-1, 10-5

**W**

Web Installers 1-1  
Windows directory 4-1  
Windows NT 9-1  
Write Ini File Entry actions 12-1 to 12-3  
Write Registry Entry actions 3-1, 3-4 to 3-6, 9-1, 9-4, 11-1 to 11-2