Visual Basic Setup Kit

The Visual Basic Setup Kit provides tools to create setup programs for your Visual Basic applications.

Overview

The Visual Basic Setup Kit provides the tools necessary to write professional setup programs using Visual Basic. The Setup Kit allows you to:

- Pre-install the Visual Basic runtime .DLL on your customer's system.
- Install files using a single API function call.
- Install files into the appropriate Windows directories (\WINDOWS, \WINDOWS\SYSTEM) regardless of the actual directory names.
- Use Windows version stamping resources to determine if a file should be copied to the customer's machine.
- Create Windows Program Manager groups and icons for your application.
- Create a customized look and feel for your setup program.

The tools for the Setup Kit are located in \VB\SETUPKIT\KITFILES

Components of the Visual Basic Setup Kit

The Visual Basic Setup Kit is installed in the SETUPKIT directory beneath the Visual Basic Professional Toolkit directory.

When you create your setup program with the Visual Basic Setup Kit you will work with the following files:

SETUP.EXE

This is what your customer runs to install your application. SETUP.EXE is provided by the Setup Kit. It performs the following:

1. Copies VER.DL_ and SETUPKIT.DL_ into the Microsoft Windows directory and renames them VER.DLL and SETUPKIT.DLL, respectively. If there is a newer version of the file already there, it is not overwritten.
2. Copies VBRUN100.DL_ into the Microsoft Windows directory and renames it VBRUN100.DLL. If a copy of VBRUN100.DLL is already in that directory, the file is not copied. SETUP.EXE does not check to see if there is a copy of VBRUN100.DLL elsewhere on the user's system.
3. Copies SETUP1.EXE into the Microsoft Windows directory and renames it to the name specified in the file SETUP.LST.
4. Executes SETUP1.EXE (or whatever it has been renamed) from the Microsoft Windows directory.

Note SETUP.EXE is a pre-install program. It is necessary to pre-install dynamic-link libraries (DLLs) on your customer's hard disk because otherwise Microsoft Windows may try to load a DLL from a floppy drive. If this happens and the user removes the floppy disk, Microsoft Windows asks the user to re-insert the disk that contains the DLL. This can confuse users. A pre-install program such as SETUP.EXE solves this problem.

SETUP1.EXE

This is a Visual Basic application that you customize. The source files for SETUP1.EXE are provided by the Setup Kit. You use these source files as a template for creating your own setup program. It is named SETUP1.EXE so users won't try to execute it from a floppy disk. SETUP.EXE copies SETUP1.EXE into your customer's Microsoft Windows directory and renames it to the filename you specify in SETUP.LST. SETUP.EXE then runs this file and installs your software.

Note Do not confuse SETUP.EXE with SETUP1.EXE. They are different files which perform different functions.

SETUP.LST

SETUP.LST is a one line text file that you create. It contains a filename. SETUP.EXE looks in this file to determine what the name of your setup program is so it can be installed properly.

SETUP1.BAS

SETUP1.BAS is a collection of Visual Basic functions. It is supplied by the Setup Kit. SETUP1.BAS uses VER.DLL and contains easy-to-use functions that:

- locate the Microsoft Windows and Microsoft Windows \SYSTEM subdirectories
- copy files (with or without version checking)
- create Microsoft Windows Program Manager Groups and Icons

TESTAPP.EXE

This is a sample application that you can use to test your setup program.

VBRUN100.DL_

VBRUN100.DL_ is actually VBRUN100.DLL. You rename VBRUN100.DLL to VBRUN100.DL_ when you copy it to the setup disk. VBRUN100.DLL is supplied by Visual Basic 1.0. It is required by any application that is written in Visual Basic 1.0. Use the .DL_ extension on the setup disk so that Microsoft Windows will not try to use it while the setup disk is in the floppy drive. When it is installed it is copied to the customer's machine as VBRUN100.DLL

VER.DL_

VER.DL_ is actually VER.DLL. You rename VER.DLL to VER.DL_ when you copy it to the setup disk. VER.DLL is provided by the Setup Kit. It determines the location of the customer's Microsoft Windows and Microsoft Windows \SYSTEM subdirectories and detects the version of files stamped with a Windows version stamp. Use the .DL_ extension on the setup disk so that Windows will not try to use it while the setup disk is in the floppy drive. When it is installed, it is copied to the customer's machine as VER.DLL

SETUPKIT.DL_

SETUPKIT.DL_ is actually SETUPKIT.DLL. You rename SETUPKIT.DLL to SETUPKIT.DL_ when you copy it to the setup disk. SETUPKIT.DLL is provided by the Setup Kit. It is used by SETUP1.BAS for various functions and also is used as an interface to VER.DLL. When installed, it is copied to the customer's machine as SETUPKIT.DLL.

How to Create SETUP1.EXE

SETUP1.EXE is a program that you create to setup your application. The best way to create this program is to modify the source code of the sample included with the Setup Kit (SETUP1.MAK).

The source code for the sample SETUP1.EXE is located in \VB\SETUPKIT\SETUP1. The sample installs the file TESTAPP.EXE, which is located in \VB\SETUPKIT\TESTAPP.

TESTAPP.EXE is a small application that uses the 3D controls, as well as the Common Dialog control. Therefore, in order to run correctly, TESTAPP.EXE requires that THREED.VBX, CMDIALOG.VBX, and COMMDLG.DLL are properly installed on the user's system. This is a typical setup scenario for applications created with the Professional Toolkit.

Follow the instructions in the example to create a floppy disk that uses the sample SETUP1.EXE to install TESTAPP.EXE.

The sample SETUP1.EXE demonstrates how to ask the user for the source and destination drives, ask for installation options, validate paths, create nested directories, get available disk space, copy files (with and without version checking), create program manager icons, and more.

Note To customize the sample source code so it installs your application instead of TESTAPP.EXE, you only need to change code in the Form_Load procedure of the SETUP1.FRM module.

SETUP1.MAK includes SETUP1.BAS which is a collection of useful subprograms.

See Also

How to Use the Setup Kit

Components of the Setup Kit

How to Create a Setup Disk for TESTAPP.EXE

Follow these steps to create a floppy disk that uses the Setup Kit to install TESTAPP.EXE:

1. Copy the Setup Kit files to a blank floppy:

File	Location
SETUP.EXE	\VB\SETUPKIT\KITFILES
SETUP.LST	\VB\SETUPKIT\KITFILES
SETUP1.EXE	\VB\SETUPKIT\SETUP1
SETUPKIT.DLL	\VB\SETUPKIT\KITFILES
VER.DLL	\VB\SETUPKIT\KITFILES
VBRUN100.DLL	\VB or the Microsoft Windows directory

2. Rename the Setup Kit DLLs (that are now on the floppy drive).

Current Name	New Name
SETUPKIT.DLL	SETUPKIT.DL_
VBRUN100.DLL	VBRUN100.DL_
VER.DLL	VER.DL_

For example:

```
RENAME    A:SETUPKIT.DLL  A:SETUPKIT.DL_
```

You must rename the above DLLs with a DL_ extension so Microsoft Windows does not try to use these DLLs while the setup disk is in the floppy drive.

3. Copy the Test Application's files to the floppy.

File	Location
TESTAPP.EXE	\VB\SETUPKIT\TESTAPP
THREED.VBX	Microsoft Windows \SYSTEM subdirectory
CMDIALOG.VBX	Microsoft Windows \SYSTEM subdirectory
COMMDLG.DLL	Microsoft Windows \SYSTEM subdirectory

Do not rename COMMDLG.DLL. It is not part of the Setup Kit. It is one of the files required by TESTAPP.EXE.

The floppy is now complete. To install TESTAPP.EXE, take the floppy to another machine. Insert the disk in drive A. From the Windows Program Manager (or File Manager) chose Run from the File menu and type "a:setup".

When you create diskettes for your applications, the steps will be the same, but you will:

1. Substitute your application files for the TESTAPP.EXE Files.
2. Create your own SETUP1.EXE (by modifying the source to this SETUP1.EXE)
3. Optionally change SETUP.LST. (See How to Use the Setup Kit)

How to Use the Setup Kit

This topic describes the entire procedure in detail. For a quick example of how to create a setup program using the provided sample code, see How to Create a Setup Disk for TESTAPP.EXE.

When you finish the following procedure, you should have a floppy disk containing the following files:

SETUP.EXE

SETUP1.EXE

SETUP.LST

VER.DL_

VBRUN100.DL_

** Application files you want installed on you customer's machine.

1 Determine What Files You Need to Distribute

Each control shipped with the Visual Basic Professional Toolkit requires the appropriate .VBX file. Some require additional .DLLs or other files. For details about the files required by each custom control, see Creating, Running, and Distributing Executable (.EXE) Files in the Introduction section of the Custom Control Reference.

2 Determine Where to Install the Files on Your Customer's Machine

Microsoft recommends that all .VBX and .DLL files that have the potential to be used by more than one application be installed in your customer's Microsoft Windows \SYSTEM subdirectory. This applies to all the .VBXs and .DLLs included with the Professional Toolkit.

Your application's executable (.EXE) file should be placed in its own directory along with any files that will only be used by your application, such as data files and bitmaps.

3 Write Your Setup Program (SETUP1.EXE)

4 Prepare Your Distribution Diskettes

Take a blank, formatted floppy disk and attach a printed label that reads as follows:

Disk1: Setup Disk
Insert this disk in drive **A:**
In the File Manager, choose Run from the File menu
Type **a:setup** and press Enter.

This label then provides all the instructions your customer will need to setup your application. If you cannot fit that much information on the diskette, simply label it, Disk 1: Setup Disk, and document the rest of the setup procedure in your documentation. Make as many diskettes that your application requires plus the files required by the Setup Kit. The first disk, (disk 1) will be referred to as the *Setup Disk*. Label the other disks appropriately so the user can insert them in the correct order.

5 Copy SETUP1.EXE to the setup disk.

SETUP1.EXE is the executable file you created in step 3.

6 Create SETUP.LST and Copy it to the Setup Disk

SETUP.LST is a text file that contains the filename that SETUP.EXE uses to rename SETUP1.EXE when it installs SETUP1.EXE into your customer's Windows directory. SETUP.LST can be more than one line, but only the first line will be used. Microsoft recommends that you do not pick generic names such as SETUP.EXE since it is likely that another application already uses that name. The following are examples of valid filenames:

ACMESET.EXE
MYAPPSET.EXE
CLNDRSET.EXE
MYSETUP.EXE

An example of a valid SETUP.LST file is located in \SETUPKIT\KITFILES.

Copy SETUP.LST to the setup disk.

7 Copy SETUPKIT.DL_, VER.DL_, and VBRUN100.DL_ to the Setup Disk

VER.DLL, SETUPKIT.DLL, and VBRUN100.DLL are required files. VBRUN100.DLL is provided by Visual Basic 1.0. SETUPKIT.DLL and VER.DLL are included with the Setup Kit. They are located in the \SETUPKIT\KITFILES subdirectory. To eliminate the possibility of Microsoft Windows loading and using these .DLLs while they are on a floppy drive, the Setup Kit requires that you rename them to SETUPKIT.DL_, VER.DL_ , and VBRUN100.DL_, respectively.

Copy SETUPKIT.DLL to the setup disk and rename it SETUPKIT.DL_.

Copy VER.DLL to the setup disk and rename it VER.DL_.

Copy VBRUN100.DLL the setup disk and rename it VBRUN100.DL_.

8 Copy SETUP.EXE to the Setup Disk

SETUP.EXE is the pre-install program supplied by the Setup Kit. This is the file your customers will run to install your application.

Copy SETUP.EXE to the Setup Disk. At this point in the procedure, your Setup Disk should contain these files:

- SETUP.EXE
- SETUP1.EXE
- SETUP.LST
- SETUPKIT.DL_
- VER.DL_
- VBRUN100.DL_

9 Copy Your Application Files to the Setup Disk

Copy all the files you determined your application required in step #1 to the Setup Disk. If all of the files dont fit, copy the remaining files onto additional diskettes. Make sure to label the diskettes as you make them.

10 Compare Your Disk Layout to Your Setup Application

Now that your files are all copied onto the distribution diskettes, you may discover that your original disk layout estimate was incorrect. Perhaps files had to be placed on different diskettes than you first thought, or perhaps you used more diskettes than you anticipated.

Review the code in your setup program to make sure the code agrees with the actual layout. If it does not, change either the code or the layout of the disks so that they match.

11 Test Your Distribution Diskettes

To test your setup program, place the setup disk in drive A: and run SETUP.EXE. After your application has been installed, check your hard disk to verify that SETUPKIT.DLL, VER.DLL, and VBRUN100.DLL have been copied to your Microsoft Windows subdirectory and that the remaining files required by your program have been copied correctly.

Be sure to test the installation of your application on other systems as well as your own. Your system may have residual .DLLs, .VBXs, or other files that allow your application to work on your system even though it would fail on other systems.

SETUP1.BAS API Reference

SETUP1.BAS is a collection of functions designed to handle the tasks that a setup program performs. For an example of how to use SETUP1.BAS, see the sample program (SETUP1.MAK) in the \SETUPKIT\SETUP1 subdirectory.

SETUP1.BAS contains these functions:

CenterForm

CopyFile

CreatePath

CreateProgManGroup

CreateProgManItem

FileExists

GetDiskSpaceFree

GetDrivesAllocUnit

GetFileSize

GetWindowsDir

GetWindowsSysDir

IsValidPath

PromptForNextDisk

SetFileDateTime

UpdateStatus

CopyFile Function

Action

Copies a file from a source to a destination. Optionally checks for version and date stamps, and will not copy if the source file is older than the destination. Returns integer representing success or error.

Syntax

CopyFile (*SourcePath$*, *DestPath$*, *FileName$*, *Older%*) **As Integer**

Remarks

The CopyFile function returns -1 if the copy was successful, or if the function successfully determined not to copy the file because the source was older. CopyFile returns 0 if there is an error during its operation.

This function requires STATUS.FRM to be in the project.

The CopyFile function uses these arguments

Argument	Description
SourcePath$	The string describing the complete path of the file to be copied. Wildcards are not allowed. A drive letter must be specified, followed by a colon, followed by the directory path.
DestPath$	The string describing the complete drive, path, and filename of the destination path. Wildcards are not allowed. A drive letter must be specified, followed by a colon, followed by the directory path.
FileName$	A valid DOS filename describing the file to be copied.
Older%	If True (-1) CopyFile will use VER.DLL to compare version and date stamps and if the source file is older, CopyFile will not copy the file. If False (0) CopyFile will copy the file.

Example

```
Result% = CopyFile ("A:\README.TXT","C:\MYAPP\README.TXT",0)
If Result% = 0 then
   MsgBox "Error Copying File README.TXT"
End If
```

GetWindowsDir Function

Action

Returns the current Microsoft Windows directory.

Syntax

GetWindowsDir () As String

Remarks

The GetWindowsDir function returns the path to the current Windows directory.

Example

```
windir$ = GetWindowsDir ()
```

GetWindowsSysDir Function

Action

Returns the current Microsoft Windows \SYSTEM subdirectory.

Syntax

GetWindowsSysDir () As String

Remarks

The GetWindowsSysDir function returns the path to the current Windows \SYSTEM subdirectory.

Example

```
winsysdir$ = GetWindowsSysDir ()
```

CreateProgManGroup Subprogram

Action

Creates a new Microsoft Windows Program Manager Group

Syntax

CreateProgManGroup (*FormName* **As Form**, *GroupName$*, *GroupPath$*)

Remarks

This subprogram establishes a DDE connection with the Program Manager and creates a new Microsoft Windows Program Manager Group.

Important *FormName* must contain a label control named Label1. Label1 is used for DDE operations.

The CreateProgManGroup subprogram uses these arguments

Argument	Description
MyForm	The form containing the label used to establish a DDE connection with the Program Manager.
GroupName$	The string containing the title of the Program Manager Group to be created.
GroupPath$	The string containing the filename of the new Program Manager Group.

Example

```
CreateProgramManagerGroup Form1, "My Cool Apps", "MYAPPS.GRP"
```

CreateProgManItem Subprogram

Action

Creates a new Microsoft Windows Program Manager Item

Syntax

CreateProgManItem (*FormName* **As Form**, *CmdLine$*, *IconTitle$*)

Remarks

This subprogram establishes a DDE connection with the Program Manager and creates a new Windows Program Manager Item.

Important *FormName* must contain a label control named Label1. Label1 is used for DDE operations.

The CreateProgManItem subprogram uses these arguments

Argument	Description
MyForm	The form containing the label used to establish a DDE connection with the Program Manager.
CmdLine$	The string containing the DOS command to execute when the item/icon is activated.
IconTitle$	The string containing the caption of the Program Manager Icon to be created.

Example

```
CreateProgramManagerIcon Form1,"C:\MYAPP.EXE","My First Application"
```

CreatePath Function

Action

Creates the specified path on the user's disk.

Syntax

CreatePath (**ByVal** *DestPath$*) **As Integer**

Remarks

The first three characters in *DestPath$* must contain a drive letter, followed by a "**:**". The remaining characters of the string contain the desired path, if any.

The CreatePath function returns -1 if the path is created successfully, otherwise 0 is returned.

This function is more useful than MKDIR because it can create paths more than one level deep.

Example

```
NewDir$ = "C:\MYAPPDIR\TEST\DEEP\FILES\"
result% = CreatePath (ByVal NewDir$)
If result then
   Print NewDir$ + " created"
Else
   Print NewDir$ + " not created"
Endif
```

CenterForm Subprogram

Action

Moves the specified form to slightly above the screen's center.

Syntax

CenterForm (*MyForm* **As Form**)

Remarks

Use this subprogram to display dialog boxes centered on the screen. After calling this routine, use the **Show** method to display the form.

Example

```
' Load the form as centered
CenterForm frmChoose
' Display the form as modal
frmChoose.Show 1
```

GetFileSize Function

Action

Returns the size, in bytes, of a specified file.

Syntax

GetFileSize (*FileName$*) **As Long**

Remarks

FileName$ must contain a full path name.

Example

```
MyFile$ = "C:\APPS\APPDATA\FORMS.DAT"
filesize& = GetFileSize (MyFile$)
```

IsValidPath Function

Action

Determines if a specified path is valid.

Syntax

IsValidPath (*Path$*, *DefaultDrive$*) **As Integer**

Remarks

This function returns -1 if the path specified by *Path$* is valid, otherwise 0 is returned.

Argument	Description
Path$	The path to be checked.
DefaultDrive$	The drive to use if no drive is specified in *Path$*.

This function reconstructs *Path$* so that it is in the format "*Drive*:*Dir*\...\". If *Path$* doesn't contain a drive letter, *DefaultDrive$* is prepended to *Path$*.

DefaultDrive must be in the format "*Drive*:".

UpdateStatus Subprogram

Action

Updates the installation status bar in STATUS.FRM.

Syntax

UpdateStatus (*FileLen&*)

Remarks

Use this function to provide the user with a visual cue while the program is copying files or performing some other lengthy process. This subprogram requires STATUS.FRM to be in the project.

Example

```
FileLength& = LOF(1)
UpdateStatus FileLength&
```

FileExists Function

Action

Determines if a specified file already exists.

Syntax

FileExists (*FileName$*) **As Integer**

Remarks

This function returns -1 if the file specified by *FileName$* already exists, otherwise it returns 0. *Filename$* must contain the full pathname of the file you want to verify.

Example

```
result% = FileExists ("C:\MYAPP\MYFILE.DAT")
If result% Then
   Call DoIt
Else
   Call DontDoIt
End If
```

GetDiskSpaceFree Function

Action

Returns the amount of free disk space on a drive.

Syntax

GetDiskSpaceFree (*Drive$*) **As Long**

Remarks

This function returns the amount of available disk space, in bytes, for the drive specified in *Drive$*. *Drive$* must contain a drive letter, followed by a colon ("C:"). It can optionally contain a full pathname ("C:\MYAPPS\COOLAPP\APP.DAT").

Example

```
freespace& = GetDiskSpaceFree ("C:\MYAPPS")
filesize& = GetFileSize ("MYAPP.DAT")

If freespace& > filesize& Then
   Call CopyFile (ByVal Source$, ByVal Dest$, ByVal FileName$, VerFlag%)
Else
   MsgBox "Not enough disk space"
End If
```

GetDrivesAllocUnit Function

Action

Returns the disk allocation unit for a drive.

Syntax

GetDrivesAllocUnit (*Drive$*) **As Long**

Remarks

This function returns the disk allocation unit for the drive specified in *Drive$*. *Drive$* must contain a drive letter, followed by a colon ("C:"). It can optionally contain a full pathname ("C:\MYAPPS\COOLAPP\APP.DAT").

SetFileDateTime Function

Action

Sets the destination file's date and time to the source file's date and time.

Syntax

SetFileDateTime (*SourceFile$*, *DestinationFile$*) **As Integer**

Remarks

This function returns -1 if the procedure was successful, otherwise 0 is returned.

This function is used by the **CopyFile** function.

Example

```
result% = SetFileDateTime (SourceFile$, DestFile$)
```

PromptForNextDisk Function

Action

Prompts the user to enter another disk.

Syntax

PromptForNextDisk (*DiskNum%*, *FileToLookFor$*) **As Integer**

Remarks

This function returns -1 if the correct disk is inserted, otherwise 0 is returned.

Argument	Description
DiskNum%	The number of the disk to prompt for.
FileToLookFor$	The filename is used to verify that the user inserted the disk specified by *DiskNum%*.

This function first looks for *FileToLookFor$*. If this file is not found, a dialog box is displayed which asks the user to insert the disk numbered *disknum%*. If the user inserts the correct disk, -1 is returned. If the user selects the dialog's Exit button, 0 is returned. If the user inserts the wrong disk, the dialog is redisplayed.