InstallShield
www.installshield.com

What's new

The InstallShield Visual Debugger is now a part of the Integrated Development Environment (IDE), where it can be accessed directly from the menu and toolbar.

The debugger runs in two windows: the Control window, and the resizable, movable Code window.

These are separated so that you can enlarge the code window independently to see more of your code when setting breakpoints or when stepping into the code.

Because it is now part of the IDE, which contains script editing features, it is relatively direct to jump back into an editing window to apply fixes identified by running the debugger, recompile with the integrated compiler, and re-debug. All relevant windows stay open on your screen simultaneously.

The Control and Code windows, however, look much like their predecessor in previous versions of InstallShield, making the transition to the present windowed system smooth for previous users.

{button ,PI("",`comment')} Feedback

InstallShield
www.installshield.com

What the debugger does (and doesn't) do

The InstallShield Visual Debugger is a sophisticated diagnostic tool for diagnosing and analyzing flaws in your setup projects – especially those flaws called "run-time errors."

The Debugger is not an editor, nor is it a compiler. You edit and compile scripts in the InstallShield IDE. The debugger's two purposes are:

- By means of breakpoints, to slow down the execution of your setup scripts so that you can identify where in your sequence of commands run-time errors occur, and
- By means of its watch windows, to display the changing values of selected global and local variables.

When you develop setups with InstallShield, you compile your scripts by choosing Compile from the Build menu in the IDE. Frequently, as with all code, the compiler will find errors in the scripts, an you may need to make several editing and compiling passes before a script compiles successfully. But even after it compiles successfully, it may not run successfully. When this happens, you use the InstallShield Visual Debugger to locate and correct the problem

---

Planning ahead for debugging

To get the best results from using the debugger, you must be thinking about it as you create, and then, after compiling, edit your code.

You should anticipate weak spots in your script—places where you "think" you've got the proper sequence of commands, but aren't sure, places where there are complicated if…endif statements, places where a variable might change value several times and each change is critical to the script's progress. Many developers make actual paper notes of the functions within which these potential trouble-spots occur, so you can identify them quickly when you get to the final, debugging step.

You should work very hard to eliminate syntax errors—the kind the debugger helps find. However good a tool it is, the project that debugs "clean" the first time will be the fastest project you do.

When you compile, which you must do before debugging, check the warnings as well as the errors that the compiler produces. They may provide clues to which functions or locations in your script need to be traced with the debugger.

InstallShield
www.installshield.com

Preparing for debugging

1. Open or create your project in the InstallShield IDE.
2. Compile your project's script by choosing Compile from the Build menu.
3. If your script fails to compile, edit and recompile until it compiles successfully.
4. Begin your debugging session by choosing Debug Setup from the Build menu.

When the debugger starts, it always recompiles your script for its own information-gathering purposes, and it requires that the script compile successfully. Therefore, always ensure your script compiles before running the debugger.

InstallShield
www.installshield.com

Debugging practice session

1. Open a project that you know compiles successfully.
2. Choose Debug Setup from the Build menu.
3. When the debugger opens, note that the first statement of your script is highlighted in the code window.
4. Click the Step Into button several times and observe as the debugger steps through your script statement by statement.
5. At various points in stepping through your script, note the values of variables in the Watch window.
6. Click Exit to end the debugging session.

InstallShield
www.installshield.com

Types of errors the debugger will help you find

There are three main types of errors to look for when you are debugging:

Error	Description
Syntax errors	These are the simplest errors. They include spelling and punctuation errors, as well as errors in usage and statement structure. For example: n Misspelling a keyword n Forgetting to end a statement with a semicolon n Typing an if statement without an endif The InstallShield Script Compiler detects and flags syntax errors.
Run-time errors	These errors occur when you run your setup and it tries to perform an operation that is impossible to complete. For example, your setup may try to copy files to a subfolder you have not yet created. In this case, all the statements in the script may be syntactically correct, but the statement to create the subfolder is missing. There are other run-time errors you cannot always prevent. These errors occur only on your user's machine. An environment error is a typical example, and will occur if: n The floppy disk containing your setup is bad n There is not enough room on your user's hard disk and you get a "Disk full" error n Your user's machine runs out of memory
Program logic errors	Logic errors can be much harder to track down than syntax or run-time errors. For example, imagine that you build a path statement, but the result you get is incorrect. The path string consists of two or three variables, each of which receives its value from a different portion of the script. How can you tell where the bug is? In this case, you might have incorrectly initialized a variable, so you need to trace the value of each one. You might also discover that all your variables were correct, but you used the incorrect operator. With the InstallShield Visual Debugger, you can track, or even alter, the values of all the variables at any point in the setup process and quickly locate the module which is causing the problem.

InstallShield
www.installshield.com

Strategies for debugging routines

Assuming you are a careful programmer and your script compiled on the first try, you may be tempted to think that it contains no run-time errors. In this case, your first debugger run should be done without any breakpoints. Just click the Go button and watch your script do its job.

In the event, however, that it doesn't actually run as you intended, your next job will be to isolate the problems in the code that caused your problem.

One way to do this is to use the Step Into button until you see the problem occur in the executing script. This may be tedious, but it has the virtue of keeping the script in Break mode throughout, so that you will have an exact line number showing and selected in the debugger's code window when you switch to it from the setup window.

It might be a good idea to set a breakpoint at the top of the function in which that line falls, rather than at that line itself, then use the Go button to bring your setup to that function's top, and Step Into throughout the entire function, following critical variables along the way.

On the other hand, if you have doubts about your code in the first place, you could set some breakpoints early in your script from the very beginning, and then use the Go button to proceed full-speed to those, Stepping Into only until you are sure that the current part of your script is not going to fail. If several breakpoints are set, you will be running and pauses in spurts, and may be more reassured that you know exactly what is going on with all your variables along the way.

The choice is yours!

InstallShield
www.installshield.com

Using Breakpoints with Go, Step Into and Step Over

With at least one breakpoint set, you're ready to start executing your code. You have two choices of how to start:

- The Go button
- The Step Into button

The Go button will start your script, and allow it to run until it reaches a breakpoint. When you press Go, your screen will fill with the image you selected for your project's initial graphic, and the debugger will fall to a lower "layer" of your screen. You can call it back to the top at any time by pressing Alt+Tab.

If you want to step through your script one executable line at a time—or if you failed to set any breakpoints--you can start your actual run by pressing the Step Into button. For most cases, it is unnecessarily tedious to do this for the entire script, but extremely valuable to do it for complex sections.

After the execution of your script has paused, you can choose to proceed by using either the Step Into or Step Over buttons, depending on what lies ahead and how you wish to proceed at the moment. The greatest benefit of both of these buttons lies in using them in conjunction with the Watch window.

The Step Into button executes the code (it's intended, and named for stepping into functions) one step at a time as you click the button. In order to jump to the next logical break, such as the end of this function, or the end of an if… endif or while…endwhile loop, but not allow it to run uncontrolled beyond there, use the Step Over button.

InstallShield
www.installshield.com

The debugger's interface

The debugger's interface has several parts:

The control window
This is where you set variables and use the buttons. The control window is sizable and movable.

The code window
This is where your code is displayed for debugging, and where you can set breakpoints.

A set of buttons on the control window
Use the buttons to stop and start the execution of your code, to set breakpoints and to exit the debugger.

The Set Breakpoints dialog box, which is opened with the Break button.

{button ,PI("",`comment')}__Feedback

InstallShield
www.installshield.com

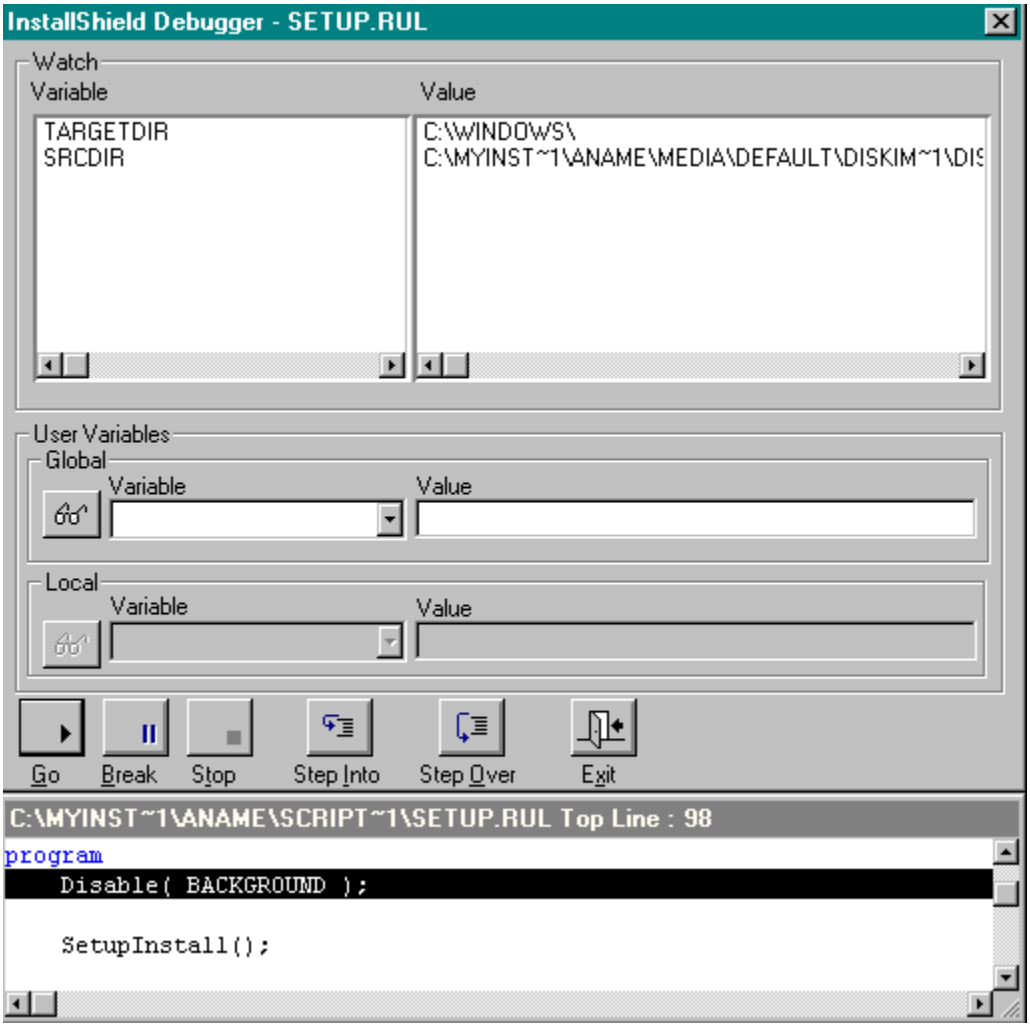
The control window

The control window is where you set variables and use the buttons. The control window is sizable and movable. Its title bar contains the name of your primary script (Setup.rul).

At the top of the control window is the Watch window, in which chosen variables are displayed with their current values.

Just below that, in the User Variables section, are the Global and Local boxes in which you designate which variables to watch, and in which you can manually reset values when your script is executing in Break mode.

{button ,PI("",`comment')} Feedback



{button ,PI("",`comment')} Feedback

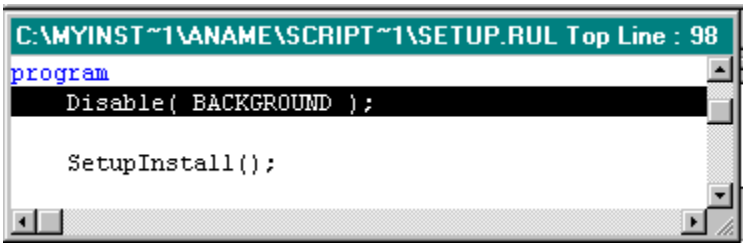
InstallShield
www.installshield.com

The code window

The Code window is the place where your code is displayed, and where you may set breakpoints by double-clicking. The code window is sizable and movable, although it initially comes up looking as if it's attached to the control window. By making it larger, you can see more of your code.

Note that the title bar of the code window contains your file's name and the line number of the uppermost displayed line. The only way to determine a line's number in this window is to scroll the desired line to the window top and then read the title bar. (Of course, you may count down from the topmost displayed line.) You cannot use the Go To keystroke to select a specific line by number, for instance to set a breakpoint there.

{button ,PI("",`comment')} Feedback



{button ,PI("",`comment')} Feedback

InstallShield
www.installshield.com

The debugger buttons

At the bottom of the control window, there are six buttons.

The Go button starts running your script from scratch, or releases the script from Break mode after it has been paused. It will run until it hits a breakpoint.

The Break button opens the Set Breakpoints dialog box. When breakpoints have been set, the small windows in the button appear colored.

The Stop button arbitrarily and absolutely cuts off the running of your script. It will have little use unless a setup "goes haywire" unexpectedly. Since your breakpoints will stop the script anyway, and if you leave the debugger to edit and recompile a script, it will always start again from its beginning.

The Step Into button executes the next available line of code only.

The Step Over button is useful if you've set one breakpoint too many at the beginning of a function. Use it to skip that function and resume running at full speed up to the next breakpoint.

The Exit button stops all processing at the next pause exits the debugger completely, exactly as the close box does. Breakpoints and variable watch settings are cleared at the end of each debugger session.

InstallShield
www.installshield.com

The Go button

Click the Go button to begin executing your script. Your setup keeps running until:

- The script reaches a breakpoint previously set
- You stop the setup by clicking the Stop button

If you click the Go button when the InstallShield Visual Debugger first opens and you do not stop it using one of the above methods, your setup will execute from the beginning to the end exactly as your user will see it.

To run the setup one statement at a time, use the Step Into button.

The Break button

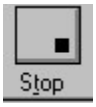
Click the Break button to open the Set Breakpoints dialog box

---

{button ,PI("",`comment')} Feedback

InstallShield
www.installshield.com

The Stop button



Click the Stop button to interrupt your script's running at the next possible stopping point in your script.

When you first open the debugger, the Stop button is disabled. When you click Go the Stop button is enabled. When the script pauses (usually when it is waiting for user input at a dialog box), you can use the Stop button to halt your script.

Use Alt+Tab to toggle focus onto the debugger window. If the Stop button is red you can click the Stop button to stop executing your script at the next statement. When you stop the setup this way, the debugger goes into break mode.

{button ,PI("",`comment')} Feedback

InstallShield
www.installshield.com

The Step Into button

Click the Step Into button to execute one statement at time. The debugger enters run time, but only until it executes one statement. As soon as the debugger reaches another executable statement, it enters Break mode.

If you want go through each statement of a user-defined function, click the Step Into button when you reach the function call.
When you click the Step Into button, the debugger loads the code for the function into the Source Code window and highlights the first executable line.
Once you have stepped into the function, you can step through it line-by-line using either the Step or Step Into button.

Do not click the Exit button to exit a function you have stepped into.
Doing so will cause the program to exit the script entirely.

You can follow the value of a local variable only after you use the Step Into button to enter a user-defined function.

Once you step into a function, you can open the Local combo box and see the values for the local variables in that function. The Eyeglass button next to the Local combo box becomes active. By clicking it, you can add local variables to the Watch window.

If the function is located in source outside the script, such as a DLL file, the title bar will display the name of the file containing the function.

When the current line is not a user-defined function, clicking the Step Into button has the same result as clicking the Step button.

InstallShield
www.installshield.com

The Step Over button

Use the Step Over button when your script is in Break mode, and it stops at a function that you do not want to check.

This usually occurs when there is an external function which you know is OK, and which does not set any variables critical to the rest of your script.

The Exit button

You can exit the debugger session by clicking the Exit button. This action closes the debug window and exits the setup.

When you exit a debugging session, any changes you have made during the session are not saved. When you start another session, none of your breakpoints, modification to variables, etc. will remain.

Color syntax highlighting

The InstallShield Visual Debugger uses color syntax highlighting to display your script in the Source Code window. Your script appears with each element of its syntax highlighted in a different color.

These syntax colors make it possible to track each component of your script visually as you debug. With color syntax highlighting, you can, for instance, quickly determine which functions are built-in and which are user-defined.

InstallShield's built-in color scheme is detailed under the Tools menu. Choose Options, then choose the Format tab.

Setting and deleting breakpoints

Breakpoints are arbitrary markers that determine where the debugger will pause the execution of your compiled script so that you can either step through the code one line at a time, or watch changes in variables as the execution progresses, or both.

Breakpoints may be set either for functions or for individual executable lines of code.

Breakpoints may be deleted by reversing the procedures used to set them.

{button ,PI("",`comment')} Feedback

The Set Breakpoints dialog box

The Set Breakpoints dialog box opens when you click the Break button

The Set Breakpoints dialog box has three text boxes:

- The Functions box is a combo box which you can scroll to choose among functions as breakpoint locations.
- The Set Break At box allows you to type in a target location, including a line number, if you happen to know it.
- The Current Break Points box displays those break point locations already set.

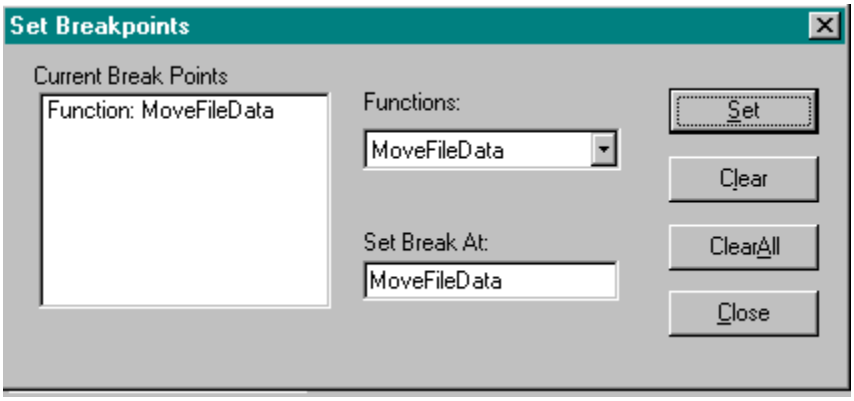
There are also four buttons

Set button
Use the Set button to move a name from the Functions box or a line number from the Set Break At box to the Current Break Points box, that is, to actually set it.

Clear button
Use the Clear button to remove a breakpoint from an item you have previously selected in the Current Break Points box.

ClearAll button
Use the ClearAll button to remove all breakpoints from the Current Break Points box.

Close button
Use the Close button (or the Close box on the title bar) to close the dialog box.



{button ,PI("",`comment')}_Feedback

To set a breakpoint for a function

With your project selected in the Project workspace:

- Click the Break button. The Set Breakpoints dialog box opens.
- Find the Functions box, and click its arrow to scroll to the function at which you'd like the execution to pause. All the functions used in the current script will appear in alphabetical order.
- Click your chosen function, then click the Set button to actually select it.

  Its name will be copied to the larger Current Breakpoints box and the line of code which calls it will appear colored red, with white type in the code window.

You may repeat this process to set multiple breakpoints in the Current Breakpoints box.

You can delete a breakpoint by selecting its name in the Current Breakpoints Window and pressing the Del key.

You cannot set a breakpoint at a built-in InstallShield function.

{button ,AL(`breakpoint for any executable line of code',0,`',`')} See also

{button ,PI("",`comment')} Feedback

To set a breakpoint for any executable line of code

With your the main debugger window open:

- Scroll the code window to display the line you wish to be the first line not executed when execution pauses
- Double-click that line. The line will appear colored red, with white type in the code window.

If you happen to have the Set Breakpoints dialog box open, you will notice that the line number of the breakpoint line will appear in the Current Breakpoints box.

Alternatively, if you happen to know the line number of the line where you want the breakpoint, you can:

- Click the Break button to open the Set Breakpoints dialog box.
- Click in the Set Break At box to activate it
- Type the number of the line where you want your breakpoint
- Press Enter or click the Set button.

Your line number will appear in the Current Breakpoints box. In the code box below, the line will appear colored red, with white type in the code window.

You can delete a breakpoint by selecting its name in the Current Breakpoints Window and pressing the Del key.

{button ,AL(`breakpoint for a function',0,`',`')} See also

{button ,PI("",`comment')} Feedback

Deleting a breakpoint

You can delete an existing breakpoint set at either a line or a function:

Using the Set Breakpoints dialog box

1. Select the line or function you want to delete in the Current Breakpoints box. The breakpoint appears highlighted.
2. Click the Clear button.

To clear every breakpoint, click the ClearAll button.

Using the Source Code window

1. Point to the line from which you want to remove the breakpoint.
2. Double-click the selected line of text. The text in the line changes to its previous syntax color highlighting.

Setting and deleting variables

The Watch window reports the current values of chosen variables at every breakpoint or at every executed step once the break mode is on. You can watch both global and local variables change.

You can watch several string and numeric variables and structure variables in the window at one time. You cannot watch lists or other complex variables in the debugger.

Once your script is running in break mode, you may delete some variables and add others, as appropriate.

Global variables

The system variables TARGETDIR and SRCDIR are used in so many functions that they appear by default in the Watch window when you start the debugger. If you wish, you can delete them from the Watch window.

To add other global variables:

1. Select the Global combo box by clicking its arrow
2. Point to the appropriate variable in the combo box, and click it to select it. Or, you can click in the combo box and type the variable name.

The current value of the variable appears in the cell to the right.

3. To place the selected variable in the Watch window, click the Eyeglass button on the left of the variable name.

Local variables

You can watch a local variable only while the script is executing the function in which it is declared. The Local combo box is active only when the debugger is executing a function and it can contain only those variables that are available within the current function. Once the debugger finishes executing that function, the value of all local variables returns to zero, and the box is emptied.

To add local variables to the Watch window:

1. Be sure (by putting a breakpoint at the "Begin" statement of the relevant function) that the script's execution will pause at the desired function. Local variables can only be set when the debugger is already in break mode.
2. Select the Local combo box by clicking its arrow.
3. Point to the appropriate variable in the combo box, and click it to select it. Or, you can click in the combo box and type the variable name.

The current value of the variable appears in the cell to the right.

4. To place the selected variable in the Watch window, click the Eyeglass button on the left of the variable name.

If your script contains only global variables, the Local combo box is unavailable.

Deleting either global or local variables

To delete a variable from the watch window:

1. Select the variable you want to delete in the watch window and click. The variable appears highlighted.
2. Press the Delete key on your keyboard.

Using the debugger with the script editor

The Visual Debugger neither identifies errors nor edits code. Only you can find errors that escaped the compiler, and only an editor can actually change the code.

When running the debugger, you may have the Script Editor window open, and it may contain the same file (probably Setup.rul) as does the debugger's code window. When you stop the debugger and go to edit your code, you must do so in the Script Editing window. If another file is there, or that window isn't open, you must set it up for editing the offending script.

When you feel you have made enough changes, you must

- Save the script with Ctrl+S.
- Compile the script (choose Compile from the Build menu).
- Reselect the debugger window.
- Adjust breakpoints and variables, if appropriate, to include the changes just made.
- Click Go or Step Into again to rerun the code in Debugger from scratch.

Actually watch the variables change

In order to see the value of a variable, you must be viewing the Watch window, but while your script is running, your application (and not the debugger) will have the focus on your screen.

Use Alt-Tab in alternation with the debugger's Step Into button to switch back and forth between the executing script and the debugger.

In order to trace precisely, it usually makes sense to use only the Step Into button to proceed through a function where you have variables being watched. If you Step Over the entire function, your variable will change, but it will be lost from the Watch window before you can see the change(s) in its value.

Watch return values from built-in functions

The system variable LAST_RESULT contains the value returned by the most recent call to a built-in InstallShield function. To watch the value of LAST_RESULT, do the following:

1. In the Global Variable combo box, select LAST_RESULT from the drop-down list box.
2. Click the eyeglass button to the right of the combo box. LAST_RESULT appears in the Watch window.

Choose variables to watch

Perhaps the biggest trade secret of swift, sure debugging is knowing which variables to choose to watch. You cannot watch them all. If you're debugging a script you've written or edited, most likely you know where you got creative…and those spots are more likely to have problems than others just because that material is new. But here are a few suggestions:

- Any variables you have initialized based on a count of objects in a list. Perhaps your list changed after the code was "finished"?
- Any predefined variables whose values you intended to redefine. Perhaps some other part of your script resets them? Or, conversely, some other part of the script expects them to have the default values?
- The last variable defined or changed in the script just before the point where it failed on a previous debugging session.
- Any variables on which the variables which failed might depend.
- Variables set by statements where you might have used the wrong operator, such as ">" instead of "<" or "+" instead of "-".

{button ,AL(`Modify the script that the Project Wizard generates D',0,`',`')}See also

{button ,PI("",`comment')}Feedback

Spot-check one variable

The Watch window is intended to monitor one or more variables continuously as the script executes. But You can also if you just need to check the value of a variable once, or from time to time, you do not need to add it to the Watch window.

You can spot-check the value of any variable by doing the following:

1. Select the Global or Local combo box the same way you would if you were adding a variable to the Watch window.
2. When the combo box drops down, click a variable. The variable appears in the cell to the left, with its current value in the cell to the right.

Manually change variables' values

Initially, when you put a global variable in the Watch window, its value is set by your script's initialization of that variable. A local variable is initially set at 0.

As your script executes, however, the value of the variable may change, and if so, the changed value will be reported in the Watch window, which you can see any time you are in break mode.

The ability to change the values of variables while the script is running allows you to test different possible scenarios. It also allows you temporarily to override the effects of a logic error in your script so that you can continue the current debugging run.

To change the value of a variable, proceed exactly as you would to set that variable in first place, except that instead of allowing the value which comes up in the right-hand text box to stand, you delete it and type your preferred value.

Set a breakpoint with my mouse

If the line at which you want to set a breakpoint is visible in the Source Code window, this is the quickest way to set the breakpoint:

1. Point to the line at which you want to set the breakpoint.
2. Double-click the selected line. The text in the line changes to white against a red background to mark the breakpoint.

Debug my setup on a different machine

You may need to debug your setup on a machine on which InstallShield has not been installed—for example, a machine running another operating system (such as Windows 3.1 or Windows NT 3.51) that your setup targets. To debug on a different machine, do the following:

1. Compile and build your setup on the machine on which InstallShield has been installed.
2. Select Debug Setup from the Build menu. This generates the Setup.dbg file before launching the Visual Debugger; once the Debugger is launched, you may immediately exit it.
3. Copy the following to the other machine:
   - The folders in <drive name>\My Installations\<project name>\Media\<build name>\Disk Images (or the <project name>\...\Disk Images subfolder of the project folder you specified in the Options dialog box's Project Location tab). These folders are named Disk1, ..., DiskN).
   - The file <drive name>\My Installations\<project name>\Script Files\Setup.dbg.
   - Isdbgi51.dll (32-bit Intel), Isdbga51.dll (32-bit Alpha), or Isdbg51.dll (16-bit), from your Windows system folder to the other machine's Windows system folder (32-bit) or from your Windows folder to the other machine's Windows folder (16-bit).
4. On the other machine, in DOS mode, switch to the <drive name>\My Installations\<project name>\Media\<build name>\Disk Images\disk1 folder and enter `setup -d` (or under Windows 3.x, `win <path of disk 1>\setup -d`) at the prompt.

Cope with problems using the debugger

If the debugger does not appear (choose Debug Setup from the Build menu), check for the presence of the following files:

- Setup.dbg—in your My Installations\<project name>\Script Files folder (or the <project name>\Script Files subfolder of the project folder you specified in the Options dialog box's Project Location tab)
- Setup.rul—in your My Installations\<project name>\Script Files folder
- Setup.ins—in your My Installations\<project name>\Script Files folder
- Setup.exe—in your My Installations\<project name>\Media\<build name>\Disk Images\Disk1 folder
- Isdbg51.dll—in your Windows system folder (16-bit Windows)
- Isdbgi51.dll—in your Windows system folder (32-bit Windows Intel)
- Isdbga51.dll—in your Windows system folder (32-bit Windows Alpha)

The last three of these are supplied and installed by InstallShield. Setup.rul is the master script you made for your project. Setup.exe and Setup.ins are created by the compilation process.

Displaying custom dialog boxes

One common technical support issues is the failure of custom dialog boxes to display. Building and debugging custom dialog boxes can be a sophisticated process, but you can often find the bug which prevents a custom dialog from displaying by making sure that all of the following conditions are true:

- The DLL that contains the dialog is on your shipping disk.
- Your setup is copying the DLL to the path and subfolder where your script expects to find it.
- The dialog you are loading is in the DLL.
- You are using the correct ID to address the dialog if the dialog is in the DLL.

{button ,AL(`Custom_Dialog_Boxes D',0,`',`')} See also

{button ,PI("",`comment')} Feedback

Running Scandisk

Windows 95 and DOS-based systems use VFAT and FAT, respectively. The VFAT and FAT file systems sometimes fall victim to extreme disk fragmentation. A disk is considered fragmented when a very large number of files are stored in a very large number of separate clusters or groups of clusters.

Highly fragmented disks push the limits of VFAT and FAT file management capabilities. InstallShield is a program that by its very nature is file transfer-intensive. When a target system hard disk is highly fragmented and a setup runs, there is the potential for file corruption and data loss when the file system on the fragmented disk cannot handle the intensive file transfer operations.

As a precaution against data loss during setups run on fragmented disks, run Scandisk on your target system hard disk before running setups to correct any hard disk fragmentation problems. You may also consider advising your users to run Scandisk before installing your application. You can display an AskYesNo dialog box early in the setup, offering users the chance to run Scandisk, after which they can restart the setup.

{button ,AL(`AskYesNo D',0,`',`')} See also

{button ,PI("",`comment')} Feedback

Creating program folders and icons

If you are having difficulty creating program folders (program groups in Windows 3.1 and NT) and icons, isolate the section of your script in which you create them and run it separately.

Use the SprintfBox function to display the parameters of your AddFolderIcon function. Make sure that all of the values passed to the parameters are valid and in the correct order.

{button ,AL(`AddFolderIcon D;SprintfBox D',0,`',`')} See also

{button ,PI("",`comment')} Feedback

Switching disks

Your script may encounter problems when it attempts to prompt the user to remove one disk and insert the next. For example, you might get a "File not found" error message at the point where the user is supposed to take out Disk 1 and insert Disk 2.

Usually when you have such a problem, it occurs because you left a file open on your diskette. DOS will raise this error message when it finds a new disk volume ID or file allocation table and cannot access the open file.

To avoid this problem, do *not* open files directly on the diskette during your setup. Do not use your information files, bitmap files, or DLLs directly from diskette. Instead, copy the file(s) to the target hard disk and access them from the hard disk.

If you encounter such a problem while debugging, check to make sure that you do not open any files before transferring them to the target disk.

Before you call us

Try to isolate the problem by determining from your user exactly what is wrong. Are many of your customers reporting the problem, a few, or only this customer? Here are some areas to check in diagnosing your customer's problem:

- Has the user tried to install your software more than once? If not, ask the user to install it again on the same system. If there is still a problem, ask the user to try installing it on another system — on several, if possible. It is unreasonable to establish conclusions using only one system.
- Is the problem related to the distribution media? If you ship both 3.5" disks and CD-ROM, ask the user to try both. This may help identify a disk problem or a drive problem. If the same problem occurs on both sets of disks, you have significantly narrowed down the problem. If you are confident there is nothing wrong with your setup, send the user another set of disks.
- Use brand-name distribution media whenever possible. If you had disk problems before, you know it does not pay to cut corners when you buy disks. A good disk costs only ten or fifteen cents more than a disk of marginal quality, but the difference between a happy customer and an annoyed one is all the difference in the world.
- If you determine that your customer's problem occurs on only one system, you need to find out how that system differs from others.
- Ask the customer how much memory and system resources are free. If the system resources are extremely low, it is possible some other program is using them. Ask the user to exit Windows to free system resources. If there is ample memory and system resources free after restarting Windows, ask the user to try the setup program again.
- Check what version of Windows the user is running. InstallShield typically needs a minimum of 2 MB of RAM on the system. However, it is possible that if you are using an extensive script and large bitmaps, you will require more RAM. If your customer is pushing these limits, they will need more RAM.
- Look at the Config.sys and Autoexec.bat files. Ask your customer to send you a copy of these files. If the Config.sys has many entries, and your customer is loading a lot of programs to specific locations, this could cause problems. Special memory managers and drivers with a lot of parameters can cause problems. Use only the most basic drivers. Ask the customer to remove as many drivers as possible from their Config.sys. In general, the more lines you find in the Config.sys, the greater the chance for conflict. Have them temporarily REM all unnecessary statements and reboot the system.

Display drivers

When a customer cannot *start* InstallShield properly, it is often because of their display driver.

The following occurrences usually indicate that the problem is a display driver:

- InstallShield starts up but nothing shows up on the screen
- When you try to display a bitmap, the system hangs
- When you try to fade the bitmap in, something goes wrong with the screen

The display driver executes the graphics commands for the Windows graphical user interface. When your customer has a display-driver problem, however, neither Windows nor InstallShield is running; control has been passed to the display driver. Each vendor who manufactures a video card also writes the display driver for the video card; the customer's display card is only as good as the customer's driver.

If your customer is running a non-standard display driver, ask your user to install the software again using a standard Microsoft Windows VGA driver. If necessary, the customer can always return to the non-standard driver after installing your software.

Determine the end user's monitor resolution. Extremely high-resolution monitors—1600 x 1280 or higher—may use specialized drivers that may not have been thoroughly tested.

Anti-virus programs

An anti-virus program can interfere with your customer's setup. Most anti-virus programs do two things:

- Prevent you from copying EXE, DLL, COM and other program files and data to your hard disk.
- Watch for illegal operations; for example, any attempt to write to the boot sector of your hard disk.

The problem is that anti-virus programs cannot distinguish between harmful and helpful software, so they can stop you from installing new software.

What the user can do

The Central Point and MS-DOS anti-virus programs offer you more than one level of protection. At one level, the program will lock any new EXE program file being copied to the hard disk, and will not allow the program to run until it has been scanned for viruses.

When this happens, your setup will look as if it is running smoothly. Your icons will be created in Program Manager, your dialog boxes will appear. Unless your user has selected the option to scan new EXE files for viruses, the anti-virus program will not allow your program to launch.

Before InstallShield can install your software on the user's hard drive, it copies the setup program from the floppy disk to the hard disk, and then launches the setup from the hard disk. If a user is running an anti-virus program, there is no way InstallShield or any setup can override it. Tell your customer to:

1. Open Autoexec.bat using a text editor.
2. Go to the beginning of each (there should not be more than one) line that starts an anti-virus program and type:

   `REM`
3. After REM, type a space.
4. Save Autoexec.bat.
5. Repeat Steps 1 - 4 for Config.sys.
6. Reboot the computer and try the setup again.

For most anti-virus programs, the default level of protection is much lower and does allow you to copy an EXE file to the hard disk. Most users will not have trouble installing your software. You may want to include a note in your user's guide or Readme file about anti-virus programs and how they can affect a setup.

InstallShield help is attempting to send you to the InstallShield Web site (www.installshield.com).

We welcome your suggestions for improving InstallShield help. To share your ideas, please visit http://www.installshield.com/talk/ or email doc@installshield.com.