SmartCheck™ Basics

NuMega™ SmartCheck™ 6

Automatic Run-Time Error Diagnosis for Visual Basic®

Windows® 95
Windows® 98
Windows NT®

Compuware® NuMega™

July 1998

Part number 0000-55-2748

Revision 3

Contents

1 The SmartCheck Solution

The need for software development teams to produce quality software is greater than ever before. The complexity of software has grown geometrically and the opportunity for problems to develop is immense. Software defects can cripple a product, cause lengthy schedule delays, and ultimately cost the engineer, the development team, and the company dearly.

Developers spend most of their debugging time tracking down and repairing elusive bugs that were introduced early in development. The quality assurance staff does the bulk of the feature testing late in the development process when the schedule allows little time. Unfortunately, testing at this late stage usually focuses only on the outward functionality of the product. Testers run a GUI regression test bed, achieve exit criteria, and declare the product ready for shipping.

All too often, the product still possesses many hidden bugs that conventional testing techniques failed to identify. These bugs create customer dissatisfaction and poor product reputation. Updates, patches, and other expensive, embarrassing retroactive fixes cost time and money that could be spent more profitably and creatively on product improvement and new product development.

Check Early, Check Often— Our Philosophy

The solution to this problem is simple and quality assurance circles have been aware of it for years. To increase software quality, developers must thoroughly test their code early in the development process. Bugs must be caught and resolved as they are introduced to avoid surprises during integration, quality assurance, beta testing, and production. Briefly stated, “check early, check often.”

Before NuMega debugging solutions, this was easier said than done. Most developers need to spend the majority of their time writing code if they are to release a product on schedule. Unfortunately, few developers have the time or resources to test their products thoroughly as they develop them.

SmartCheck provides the solution to this dilemma. SmartCheck automates the crucial process of error-detection and analysis, identifies elusive bugs that are beyond the reach of traditional debugging and testing techniques, and adds little or no time to the development process.

Industry figures show that 50 percent of the development effort on an average project is spent on debugging. Regularly using SmartCheck will significantly reduce the amount of time needed to debug your applications.

The Benefits of Using SmartCheck

SmartCheck is the first run-time debugging tool for Visual Basic to provide clear, detailed analysis of program errors and detailed tracking and logging of program events. It automatically detects and diagnoses Visual Basic runtime errors and translates vague error messages into exact problem descriptions. As such, it serves as an invaluable development tool for both novice and experienced Visual Basic developers.

SmartCheck addresses the most hard-to-solve conditions encountered by Visual Basic developers:

- Fatal run-time errors that are cryptic and hard to solve.
- Problems that result from a specific sequence of events.
- Incorrect Windows API Usage from Visual Basic.
- Bad values passed to built-in Visual Basic functions.
- Hard-to-solve value coercions.
- Errors in components, such as ActiveX controls, used by your program

The rest of this section describes how SmartCheck addresses each of these conditions.

Fatal Runtime Error Analysis

When a runtime error occurs, Visual Basic only reports the general category of the error, and prompts you to acknowledge the error message. In contrast, SmartCheck performs detailed analysis of fatal runtime errors, providing specific information on the cause, and possible solutions, and pinpointing the line in source code at which the error occurred. For example, the Visual Basic runtime error “429 - ActiveX component can’t create object” represents many different error conditions that SmartCheck detects and reports. SmartCheck can report a specific key that is missing from the registry, or identify a specific DLL that failed to load. The cryptic Visual Basic runtime error message could be transformed into “Component Creation Failure: ActiveX DLL c:\dll\mycomp.dll not found.”

Error And Event Tracking and Logging

Windows is an event-driven environment in which much of your program is executed in response to Windows messages and other events. SmartCheck logs events as they occur, so you can see a complete history of events that led to a problem. SmartCheck logs the following Visual Basic events:

- Error messages, including advanced analysis of Visual Basic runtime errors
- API calls you make from Visual Basic, including argument information
- Form and control creations for Visual Basic
- Object interfaces including setting and using control properties and methods
- Object events, such as click or load
- Visual Basic built-in functions, including arguments
- Value coercions
- Handled runtime errors, including the error handler, and where it returns

You can also use SmartCheck to view system API calls, Windows messages and hooks, and output debug strings, all of which can prove especially valuable when you are analyzing your C, C++ or Delphi components, or when you need to drill deeper into Visual Basic.

Windows API Checking

When doing Visual Basic development, you will often need to use Windows API calls to perform functions Visual Basic can’t handle directly. Because API calls are built and documented for C and C++ developers, you may pass parameters incorrectly or fail to check return codes when making Windows API calls. When these calls fail, Visual Basic does not provide any information on how or why they failed. SmartCheck finds and highlights incorrect usage or failure of Windows API calls.

Visual Basic Argument Checking

Passing invalid arguments to functions can result in vague Visual Basic error messages, such as the well-known “Error 5 - Invalid procedure call or argument”, which burdens you with the effort of pinpointing the cause of the error. This effort can be especially time-consuming when an error occurs on a complex line of code in which many Visual Basic keywords are used, and many arguments are passed. SmartCheck tracks arguments and returns from Visual Basic functions, and can identify the specific value that caused an error, as well as provide details on the value that was expected.

Value Coercion Detection

Visual Basic allows you to use variants, that is, variables that do not have explicitly specified data types. A variant’s data type may change automatically, at runtime, according to the context in which it is used. *Incorrect* use of variants can result in particularly hard-to-find errors, because value coercions may occur without your intention.

In addition, Visual Basic performs automatic type adjustments of non-variants, which can result in lost information, as in this example:

```
dim d as double
dim i as integer
...
i = d
```

SmartCheck detects and reports value coercions as they happen. You can use this information to isolate errors and, in general, to assess your use of variants. For example, after reviewing the SmartCheck information on value coercions you may choose to declare the data types of certain variables explicitly to eliminate excessive conversions, and thus improve program performance.

Error Detection in Third-party Components

You may often use ready-made components, such as ActiveX controls, to incorporate special capabilities into your Visual Basic projects. Because these components are often purchased, rather than being developed in-house, you may not have the source code for them. However, since these components are critical to your application, you need to ensure their correct use, and even be able to detect problems in them.

Like all NuMega products, SmartCheck checks third-party components, and includes NuMega's award-winning error checking for C/C++ and Delphi components. If you do have the source code for any component, SmartCheck will pinpoint the line in the source code at which an error occurs.

Unlike ordinary heap checkers that are limited to finding common memory errors, SmartCheck is a sophisticated error-detection tool that validates the latest windows APIs including ActiveX, DirectX, OLE, COM, and ODBC. Additionally, SmartCheck detects errors in executable files, dynamic link libraries, third-party modules, and OLE components. SmartCheck also pinpoints static, stack, and heap errors, as well as memory and resource leaks, in the C, C++ and Delphi components you use with your Visual Basic applications.

Flexible Environment

SmartCheck provides a flexible environment. You can:

- Set SmartCheck options and launch SmartCheck from Microsoft Visual Basic. When SmartCheck starts, it automatically compiles and starts running your program.
- Start SmartCheck as an independent application, rather than launching it from Visual Basic.
- Start SmartCheck from a command line or automate a series of tests from a batch file.

Windows Compliance Assurance

To assure your program's ability to run on all Win32 variants, SmartCheck creates compliance reports that identify calls specific to Windows NT 4.0, Windows NT 5.0, Windows 95, Windows 98, Windows CE 2.0, and Win32s. SmartCheck also displays your program's use of Visual Basic built-in functions, support routines, ANSI C functions, and ANSI C Extensions.

Customer Assistance

For Non-technical Issues

NuMega Customer Service is available to answer any questions you might have regarding upgrades, serial numbers and other order fulfillment needs. Customer Service is available from 8:30am to 5:30pm EST, Monday through Friday. Call:

- In the U.S. and Canada: 888-283-9896
- International: +1 603 578 8103

For Technical Issues

NuMega Technical Support can assist you with all your technical problems, from installation to troubleshooting.

Before contacting technical support please read the relevant sections of the product documentation and the ReadMe files.

You can contact Technical Support by:

E-Mail	Include your serial number and send as many details as possible to Tech@numega.com
World Wide Web	Submit issues and access our support knowledge base at www.numega.com. Go to Support.
Telephone	Telephone support is available as a paid* Priority Support Service from 8:30am to 5:30pm EST, Monday through Friday. Have product version and serial number ready. In the U.S. and Canada, call: 888 NUMEGA-S International customers, call: +1 603 578 8100 * Technical Support handles installation and setup issues free of charge.
Fax	Include your serial number and send as many details as possible to 603 578 8401

Before contacting Technical Support, please obtain and record the following information:

- Product/service pack name and version
- Product serial number
- System configuration: operating system, network configuration, amount of RAM, environment variables, and paths
- Name and version of your compiler and linker and the options you used in compiling and linking
- Problem details; settings, error messages, stack dumps, and the contents of any diagnostic windows
- If the problem is repeatable, the details of how to create the problem

Where to Go From Here

This manual provides an overview of SmartCheck and explains how to use its most commonly-used features. These include:

- Checking code
- Viewing data
- Configuring error detection and reporting
- Checking compliance

For detailed information, see the SmartCheck online Help. To access a list of SmartCheck Help topics from Visual Basic, click Help, then click SmartCheck Help Topics.

2 Checking and Analyzing Programs

SmartCheck makes checking code so easy and convenient that every member of your development team can use SmartCheck daily to test his or her code. SmartCheck analyzes your executable image, DLLs, and OCXs as they execute, so you do not need to recompile or relink your program. Simply run the program under SmartCheck, which works in the background, and automatically detects all the categories of errors described here.

Error	SmartCheck Error Detection
Visual Basic Runtime Errors	Many Visual Basic runtime error messages are general descriptions that encompass a range of error conditions. SmartCheck analyzes Visual Basic runtime errors and provides specific, detailed explanations of the conditions that caused them.
Procedure and Function Failures	SmartCheck detects errors that result from incorrect use of API functions and built-in Visual Basic functions. It reports specific causes of function failures, such as the use of incorrect data types, or out-of-range values, as arguments to functions.
Interface Method Failures	SmartCheck detects invalid arguments and return codes for OLE interface methods.
Leaks	SmartCheck detects memory and resource leaks. Memory leaks occur when memory is allocated, but never freed. SmartCheck detects memory leaks caused by Windows memory allocation functions, such as HeapAlloc, GlobalAloc, and LocalAlloc, and standard C and C++ allocation routines including malloc and new. Resource leaks occur when windows specific resources, such as HMENU, HKEY, and HCURSOR, are allocated by your program, but not released back to the system. Resource leaks can consume excess memory and degrade system performance. Note that SmartCheck tracks *memory and resource* allocations specifically as they pertain to the Windows API and to components written in languages such as C or C++. This specific type of tracking does not apply directly to programs or components written in Visual Basic.

Error	SmartCheck Error Detection
Memory	SmartCheck detects overwriting and underwriting of memory in dynamically allocated memory, local or stack memory, and global or static memory.
Pointer	Bad pointers frequently cause errors. To help you eliminate them, SmartCheck checks for: Operations on null pointers. Operations on pointers that do not point to valid data. Attempts to free handles without unlocking them.

As described in the following sections, you can run SmartCheck:

- from the Visual Basic Development Environment
- as a stand-alone application
- from the command line

Starting SmartCheck from Visual Basic

You can set SmartCheck options and launch SmartCheck from within Visual Basic. When you launch SmartCheck, it automatically compiles and starts running your program.

To start checking a program with SmartCheck from within Visual Basic:

1. On the File menu, click Open Project, then find and open the project you want to check.
2. On the Project menu, click Project Properties, then click the Compile tab. Make *sure* these options are selected:
   - ◊ Compile to Native Code
   - ◊ No Optimization
   - ◊ Create Symbolic Debug Info
3. On the DevPartner menu, click Run with SmartCheck.
4. Use options in the SmartCheck dialog to either review and change Visual Basic project properties before proceeding, or start SmartCheck.

   When SmartCheck starts, if necessary, it automatically compiles and starts running your program.

   From this point on, you are working within the SmartCheck environment, and program checking occurs exactly as described from step 5 onward in the next section.

Checking Programs With SmartCheck

To use SmartCheck as a stand-alone application:

1 Click the Windows Start button, and then point to Programs. Point to the folder that contains SmartCheck, and then click SmartCheck.

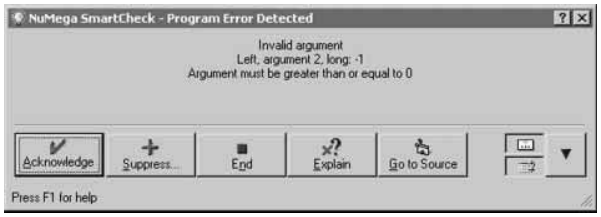
2 On the File menu, click Open.

3 Select the file you want to load and click Open.

4 On the Program menu, click Start.

SmartCheck displays the Program Results window and starts your program. The Program Results window displays the errors and events SmartCheck detects.

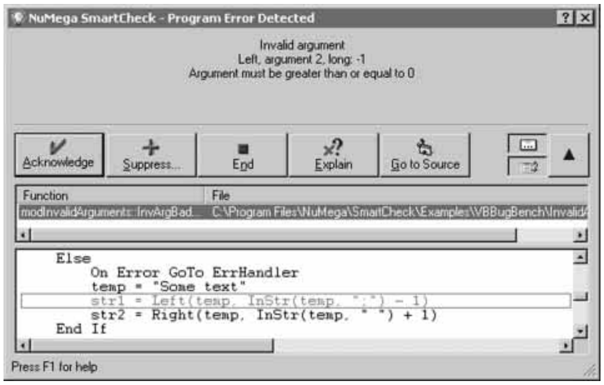
5 As you use your application, SmartCheck works in the background. When SmartCheck detects an error, it displays detailed information about it.


NuMega SmartCheck - Program Error Detected

Invalid argument
Left, argument 2, long: -1
Argument must be greater than or equal to 0

You can click:

- Acknowledge to continue checking your program.
- End to stop your program.
- Explain to view a detailed explanation of the error, with suggested solutions.
- The arrow (the rightmost) button to view source code and call stack information.


NuMega SmartCheck - Program Error Detected

Invalid argument
Left, argument 2, long: -1
Argument must be greater than or equal to 0

Function	File
modInvalidArguments::InvArgBad...	C:\Program Files\NuMega\SmartCheck\Examples\VBBugBench\Invalid

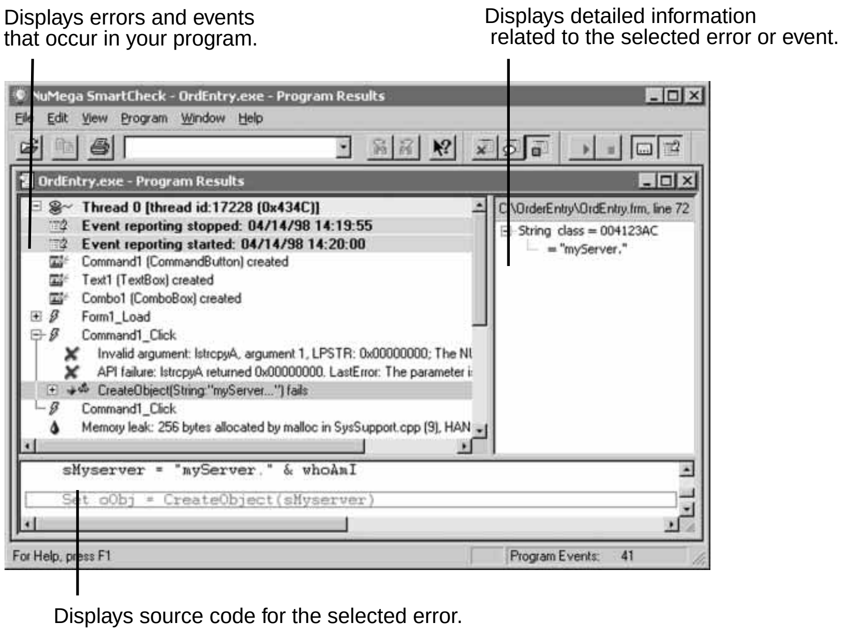
```
    Else
        On Error GoTo ErrHandler
        temp = "Some text"
        str1 = Left(temp, InStr(temp, ";") - 1)
        str2 = Right(temp, InStr(temp, " ") + 1)
    End If
```

You cannot suppress terminal errors, such as Visual Basic run-time errors.

- Suppress if you do not want SmartCheck to report the error again. SmartCheck then lets you choose the scope for which it suppresses the error (within the funciton, within the source file, within the EXE or DLL, or anywhere it occurs) and lets you add a remark. You can also save suppression information for future runs of the program. See *Suppressing Errors* on page 16.

- Go to Source to go directly to the source code that caused this error, in Visual Basic. (This option is only available if you started SmartCheck from Visual Basic.) After you review or change your project, you can either return to the active SmartCheck session or start a new SmartCheck session. See *Going to Source Code in Visual Basic* on page 15.
- Report Errors Immediately, , to have SmartCheck finish checking your program without pausing to report each subsequent error.
- Event Reporting, , to turn event reporting off or on. This option allows you to capture information relevant to testing your program, while eliminating the overhead of unnecessary event reporting. When this option is off, SmartCheck reports only terminal errors. See *Turning Event Reporting On and Off* on page 13.

When you are done checking your program, use the data in the Program Results window to analyze your program. For example, click an error in the Program Results window to display the line of code in which SmartCheck detected the error. See *Viewing the Results of Your Error Detection Session* on page 14.


Displays errors and events that occur in your program.
Displays detailed information related to the selected error or event.
NuMega SmartCheck - OrdEntry.exe - Program Results
File
Edit
View
Program
Window
Help
OrdEntry.exe - Program Results
For Help, press F1
Program Events: 41
Displays source code for the selected error.

```
Thread 0 [thread id:17228 (0x434C)]
    Event reporting stopped: 04/14/98 14:19:55
    Event reporting started: 04/14/98 14:20:00
    Command1 (CommandButton) created
    Text1 (TextBox) created
    Combo1 (ComboBox) created
    Form1_Load
    Command1_Click
        Invalid argument: lstrcpyA, argument 1, LPSTR: 0x00000000; The NU
        API failure: lstrcpyA returned 0x00000000. LastError: The parameter i
        CreateObject(String:"myServer...") fails
    Command1_Click
    Memory leak: 256 bytes allocated by malloc in SysSupport.cpp (9), HAN
```

```
C:\OrderEntry\OrdEntry.frm, line 72
String class = 004123AC
    = "myServer."
```

```
    sMyserver = "myServer." & whoAmI

    Set oObj = CreateObject(sMyserver)
```

Starting SmartCheck From the Command Line

Start SmartCheck from the command line when you want to:

- Pass a file to SmartCheck to open at initialization.
- Automate a series of tests from a batch file.

You can use the SMARTCHK command with .SCX, .SCE, and .EXE files as follows:

```
SMARTCHK [foo.scx]
```

```
SMARTCHK [foo.sce]
```

```
SMARTCHK [[arg1 arg2] foo.exe [app_arg1 app_arg2]]
```

(SmartCheck assigns .SCX as the default extension for Compliance report files and .SCE as the default extension for program results files.)

SmartCheck provides these optional switches.

Switch	Description
/B logfile	Run SmartCheck in batch mode. All operations are executed with no user input required. The results are saved in "logfile." This switch overrides /L, /M, and /S.
/L	Disable start-up splash screen.
/M	Start SmartCheck minimized.
/S	Disable immediate error reporting.
/W<dir>	Specify the working directory. The directory path must immediately follow the /w argument. Do not use a space to separate the directory path from the argument.

Turning Event Reporting On and Off

You can turn SmartCheck event reporting on and off while executing your program. This lets you capture information relevant to your testing and debugging tasks, while eliminating the overhead of unnecessary event reporting. It is especially useful when debugging applications with many initialization events, such as database applications, or applications that run continuously, such as servers.

On the Program menu, select Event Reporting to turn reporting on or off.

When reporting is on, SmartCheck captures and reports all events specified by the current Error Detection and Reporting settings for your project. When reporting is off, SmartCheck captures and displays only terminal errors. When Event Reporting is turned back on from the off state, SmartCheck correctly displays event nesting levels relative to the point at which Event Reporting was restarted.

Viewing the Results of Your Error Detection Session

SmartCheck places a wealth of information at your fingertips, including:

- Arguments passed to Visual Basic built-in functions and API functions.
- The line in the source code in which SmartCheck detected an error. In addition to viewing the code in the SmartCheck source pane, you can go directly to the line of code in the Visual Basic code window.
- The error's corresponding call stack.
- The source code for any function in the call stack.
- The point at which memory is allocated (for errors that involve a memory block that is allocated elsewhere).
- Online Help for the error.

Locating Errors

Some errors may be nested under the events in which they occurred, and therefore not immediately visible under Program Results. Here is a convenient way to locate and review the errors:

1. On the View menu, select Show Errors and Leaks only.
2. Click on an error message that you want to examine.
3. On the View menu, select Show Errors and Specific Events to view the sequence of events that led to the error.

Displaying Source Code and Call Stack Data

Click an error to display the following:

- The source code in which SmartCheck detected the error.

  SmartCheck highlights the line that contains the error by framing it and displaying it in red.
- The error's call stack.

  SmartCheck lists each function in the stack, the file in which the function is located, and the line on which the function is found.

  A statement of the form functionname!number indicates that no debug information is present for that function. functionname identifies the module (that is, the dll or EXE) that contains the function. The number following the exclamation point identifies the relative offset of the function in that module.

Using the Call Stack

The stack frame lets you display the source code for any function in the stack. This is useful for seeing the events that led to the error. If the error involves a memory block that is allocated elsewhere, the stack frame also lets you view the point at which the memory is allocated.

To view a particular function, click the corresponding function in the stack. If the error deals with memory that was allocated either from the heap or from earlier on the call stack, you can choose one of the following before selecting a function:

- Location of Error

  Lists the functions that led to the error.
- Point of Allocation

  Lists the functions where memory is allocated.
- Point of Deallocation

  Lists the functions where memory is freed.

Going to Source Code in Visual Basic

You can go directly from an error or event to the associated line of source code in Visual Basic. After reviewing your project in Visual Basic, you can either return to the active SmartCheck session or restart the SmartCheck session. Be aware that:

- Go to Source is active only if you started the SmartCheck session from Visual Basic.
- If you make changes to your Visual Basic project, then return to SmartCheck, your source files may be out-of-synch with the debug information in the active SmartCheck session. After making changes to your project, you should generally re-compile it, then run the updated version of your program under SmartCheck.
- The code must be from a module that is contained in the current Visual Basic project.

To go to source code:

1. Click on the error or event in the Program Results window.
2. Click the right mouse button, and then click Go to Source. SmartCheck opens the source file in Visual Basic, and then locates and highlights the relevant line of source code.

After reviewing your project:

1. On the DevPartner menu, select Run with SmartCheck.
2. In the SmartCheck dialog, you can either:
   - ◊ Click Return to SmartCheck to return to the active SmartCheck session.
   - ◊ Click Run with SmartCheck to recompile and restart the session.

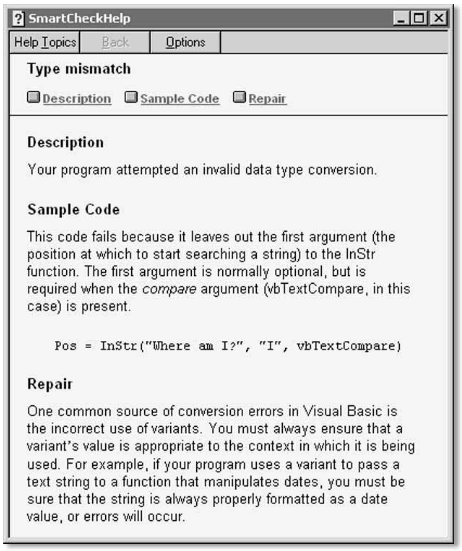
Displaying Help for the Error

SmartCheck provides the following assistance for each type of error it detects:.

- A complete description of the error.
- Sample error code.
- Suggestions for correcting the error

To display Help for a particular error, do the following:

1. Click the error on which you need Help.
2. Click the right mouse button, and then click Explain.



Type mismatch

Description

Your program attempted an invalid data type conversion.

Sample Code

This code fails because it leaves out the first argument (the position at which to start searching a string) to the InStr function. The first argument is normally optional, but is required when the *compare* argument (vbTextCompare, in this case) is present.

```
    Pos = InStr("Where am I?", "I", vbTextCompare)
```

Repair

One common source of conversion errors in Visual Basic is the incorrect use of variants. You must always ensure that a variant's value is appropriate to the context in which it is being used. For example, if your program uses a variant to pass a text string to a function that manipulates dates, you must be sure that the string is always properly formatted as a date value, or errors will occur.

Suppressing Errors

You can suppress an error while you check your program or after you analyze it in the Program Results window. Suppressed errors:

- Are ignored by the Report Errors Immediately command, that is, SmartCheck does not display the Program Error Detected window for suppressed errors.
- Can either be hidden from view or displayed in a grayed-out state under Program Results. On the View menu, select Suppressed Errors to hide or show suppressed errors.

You may want to suppress an error if it is generated by code from a third-party DLL, an OCX or another developer, or if your code handles it properly.

You cannot suppress terminal errors, such as Visual Basic run-time errors.

To suppress an error, do the following:

1. Click the error you want to suppress.
2. Click the right mouse button, and then click Suppress.
3. Select one of the following suppression scope options:
   - Suppress this Error Only When it Occurs in This Function
   - Suppress this Error Only When it Occurs in This Source File
   - Suppress this Error Only When it Occurs in This EXE or DLL

- Suppress this Error Regardless of Where it Occurs

4 If you want SmartCheck to suppress the error automatically the next time you check the program, select Save Suppression Information. Otherwise, SmartCheck only suppresses this error when you display the results of this error detection session.

5 If you want to add a notation to the error you are suppressing, add a remark in the text box.

Suppression information for the current module is stored in a .SUP file in the directory of the executable file being debugged.

SmartCheck adds the suppressed error to the list it maintains in the Error Suppression tab within the program settings. See *Customizing Error Suppression Settings* on page 25 for information about removing errors from the suppression list.

Changing the Results View

By default, the Program Results window displays errors, threads, leaks and Visual Basic events. However, you can change the type of data it displays by changing its view. On the View menu, click one of the following to change the Results view:

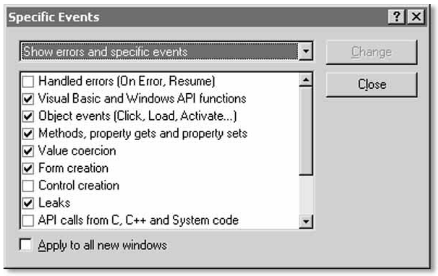
- Show Errors and Leaks Only

  Displays errors, threads, and leaks.
- Show All Events

  Displays errors, threads, leaks and all events.
- Show Errors and Specific Events

  Displays errors, threads, and specific types of events that you select. Click Specific Events on the View menu, then select the events you want to view from the available list, as shown in the sample illustration.

The Error Detection and Event Reporting program settings determine the type of errors and events that SmartCheck detects and reports. See Chapter 3: Customizing Error Detection and Reporting on page 21.



Show errors and specific events

- [ ] Handled errors (On Error, Resume)
- [x] Visual Basic and Windows API functions
- [x] Object events (Click, Load, Activate...)
- [x] Methods, property gets and property sets
- [x] Value coercion
- [x] Form creation
- [ ] Control creation
- [x] Leaks
- [ ] API calls from C, C++ and System code

- [ ] Apply to all new windows

The Program Results window uses the following icons to represent errors and events:

Icon	Event Type	Description
✗	Errors	SmartCheck detected an error in your program.
💧	Resource, Memory and OLE Leaks	SmartCheck detected a memory, resource, or interface method leak. The message describes the leak.
~	Thread-Starts	SmartCheck detected the creation of a thread.
✎	Session Information	SmartCheck identifies the current session and the system on which it is running.
📝	Event Reporting	SmartCheck inserts a statement in the event log each time you enable or disable event reporting.
⇅	Thread Context Switch	SmartCheck detected that your program has switched from one thread to another.
◆◆	Visual Basic Intrinsics and Local API Calls	SmartCheck logs this event each time your Visual Basic program uses an intrinsic (built-in) Visual Basic function such as MsgBox, Mid$ and CurDir. Local API calls are calls you make from Visual Basic to external libraries such as WIN32API.
◆ ◆ OLE	API Calls from C, C++ and System code OLE Method Calls from system code	SmartCheck detects when an API or OLE call is made on behalf of your program (as opposed to API calls that you make explicitly in your Visual Basic code). Note that a running Visual Basic program makes many calls to the Windows API behind-the-scenes. By default, SmartCheck does not report these behind-the-scenes calls, since knowing about them is generally not useful. However, it does, by default, report all explicit API calls you make within your Visual Basic code and add them to the event log, tagged with the icon for "Visual Basic Intrinsics and Local API Calls".
↗ ↙ ◆	Object Property Get Object Property Let Object Method Call	These events represent setting and referencing properties and calling methods of your Visual Basic controls. Examples include: setting Command1.Caption = "Hello"; testing the value of Command1.Caption; Command1.SetFocus. When a property "let" occurs, SmartCheck shows the new value being assigned; when a property "get" occurs, SmartCheck shows the line of code in which the property was retrieved (though not the value being retrieved). While actions involving controls are reported, note that method calls on external objects (such as "Excel.Worksheet") will not be seen. SmartCheck supports all built-in Visual Basic controls, and all the controls that come with the Visual Basic 5.0 Enterprise and Visual Basic 6.0 Enterprise editions (at the time of the printing of this manual). SmartCheck can also "see into" third-party OCXs that use IDispatch::Invoke to communicate with Visual Basic. (SmartCheck does not log events fired by controls that use the "dual interface" mechanism).

Icon	Event Type	Description
	Visual Basic Form Creations Control Creations	These events represent the creation of Visual Basic forms and controls. Note that after either of these events occurs, the control exists although no user-written code has been executed for the control yet. Typically, when a new form loads, you will see a form creation event, followed by events that represent the creation of the controls contained within the form. It is important to realize that the reporting of a form creation event is distinct from the reporting of a call to a user-written Form_Load procedure.
	Object Events	These events represent user-written handlers that execute when specific actions are invoked. For example, if you write a "Click" handler for the Command1 button, SmartCheck reports an event with the text "Command1_Click". These events are specific to control-related handlers.
	Visual Basic Value Coercions	SmartCheck logs this event each time Visual Basic performs a value coercion, that is, converts a variant's value from one data type to another. For example, if you pass the string "123" to a function that expects a numeric value, Visual Basic converts the string to a numeric type. SmartCheck logs both the input value and type and the output value and type. SmartCheck coercion reporting is particularly valuable for less obvious coercions automatically performed by Visual Basic. For example, if you pass a variable of type "Single", with a value of 123.456, to a function that expects an integer parameter, Visual Basic "silently" truncates the value to 123.
	Visual Basic Error Handlers and Resumes	SmartCheck tracks any runtime errors you handle in your Visual Basic code. It shows you when the error occurs, the error handler statement that executes and the statement at which your program resumes after the error. Error handlers and resumes are filtered out by default.
	Window and Dialog Messages	SmartCheck adds a Message event to the event log when the program processes a dialog or Windows message.
	Hooks	SmartCheck adds a Hook event to the event log when the program processes a Windows hook call. The function name and arguments are included on the line.
	Comments	SmartCheck adds a Notes event to the event log when your program makes a call to OutputDebugString.

Viewing Version Information

SmartCheck collects version information and the complete path and file name of every module loaded by your program. On the View menu, select Version Info to display this information. On the File menu, select Print to print the displayed report.

Version Info is only available when the Program Results window is active.

Printing or Saving Your Results

On the File menu, click Print to print the contents of the Program Results window.

To save the results of your error-detection session for later viewing:

1. If your application is running, quit your application.
2. On the File menu, click Save As.
3. Enter a file name and select the location in which you want to save the file.

   By default, SmartCheck saves the file in the directory that contains the executable.

3 Customizing Error Detection and Reporting

SmartCheck provides a series of program settings that let you determine how it detects and reports errors and events. These program settings control the following:

Program Setting	Description
Error Detection	Determines the types of errors SmartCheck detects and reports.
Reporting	Determines if SmartCheck collects, analyzes and reports handled Visual Basic runtime errors, collects and reports data about calls your program makes to libraries and Windows APIs, and sets event reporting on by default.
Files to Check	Determines the modules SmartCheck checks.
Error Suppression	Determines if SmartCheck reports errors in specific libraries and instances.
Program Info	Determines the program search path and directory SmartCheck uses to locate your program files and establishes program parameters to pass as command-line arguments.

Customizing Program Settings

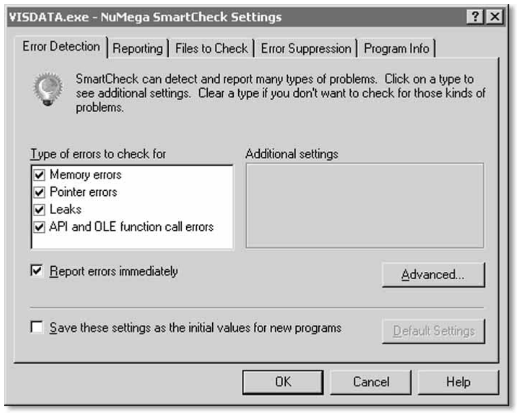
To modify the program settings:

1. Do one of the following:
   - If you are using Visual Basic, click DevPartner, then click SmartCheck Settings.
   - If you are using the SmartCheck application, click Program, then click Settings.
2. Click the tab for the settings group you want to modify.

   The sections that follow highlight the settings for each of these tabs.
3. When you finish modifying the settings, click OK to save your changes.

Customizing Error Detection Settings

The Error Detection settings determine how SmartCheck detects and reports errors.



VISDATA.exe - NuMega SmartCheck Settings

SmartCheck can detect and report many types of problems. Click on a type to see additional settings. Clear a type if you don't want to check for those kinds of problems.

Type of errors to check for

- [✓] Memory errors
- [✓] Pointer errors
- [✓] Leaks
- [✓] API and OLE function call errors

Additional settings

[✓] Report errors immediately

[ ] Save these settings as the initial values for new programs

Type of Errors to Check For

SmartCheck provides a default settings configuration that is designed to get you up and running as quickly as possible, while addressing conditions that are generally of most concern to Visual Basic developers. You can specify a unique error detection scheme, as needed, for each program you run under SmartCheck. When you click on an error category in this list, SmartCheck displays any sub-settings associated with the selected category under **Additional Settings**.

For information about controlling the events SmartCheck reports, see *Customizing Reporting Settings* on page 23.

Report Errors Immediately

Determines if SmartCheck automatically displays the Program Error Detected window each time it encounters an error in your program. Displaying the Program Error Detected window is useful for seeing errors in context. If you prefer, clear this setting to check your program without interruption. SmartCheck always maintains a log of your error-detection session, so you can see your program's errors and events at your convenience.

Save these settings as the initial values for new programs

Depending on your development environment, you may want to permanently modify your changes to the Error Detection settings. Select this setting to apply your modifications to all subsequent programs you check with SmartCheck.

Advanced Settings

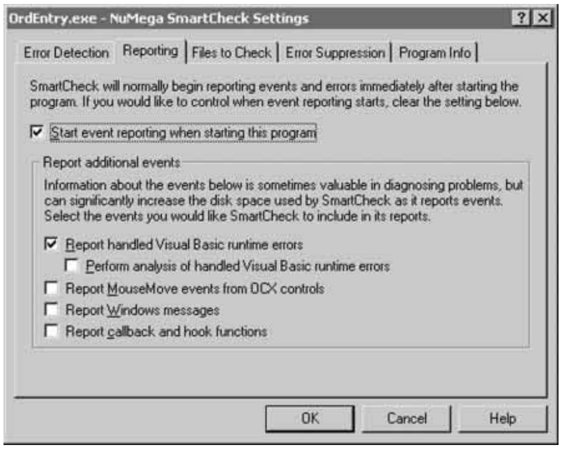
Additional error detection settings are available on the Advanced Settings dialog. To review all the Advanced settings, bring up the Advanced Settings dialog in SmartCheck and use the context-sensitive help to obtain popup definitions for specific items. Several settings that are important when addressing particular testing needs are mentioned here:

- **Report errors even if no source code is available:** It is generally assumed that you will be testing your own programs, for which source code is present. Therefore this setting is disabled by default. However, there may be instances in which you need to check for errors in third-party components, for which you do not have source code. In these instances, you need to select this setting before checking your program.
- **Performance Optimizations:** SmartCheck provides three settings that are specifically related to performance optimization, **Cache program events** and **Defer program results until program terminates**, which are disabled by default, and **Suppress system API and OLE calls**, which is enabled by default. The default settings generally result in the most efficient SmartCheck performance. However, if you are testing a database application, you may improve performance by enabling **Cache program events** and **Defer program results until program terminates.** If you are attempting to debug a hard-to-solve problem, and want to track system API and OLE calls, you should clear the setting to **Suppress system API and OLE calls**, then re-run your program.

Customizing Reporting Settings

Reporting settings instruct SmartCheck whether or not to:

- Set event reporting on by default when the given project is loaded
- Collect, analyze and report handled Visual Basic runtime errors in your program
- Collect the Windows messages your program sends and receives



SmartCheck will normally begin reporting events and errors immediately after starting the program. If you would like to control when event reporting starts, clear the setting below.

[✓] Start event reporting when starting this program

Report additional events

Information about the events below is sometimes valuable in diagnosing problems, but can significantly increase the disk space used by SmartCheck as it reports events. Select the events you would like SmartCheck to include in its reports.

[✓] Report handled Visual Basic runtime errors
[ ] Perform analysis of handled Visual Basic runtime errors
[ ] Report MouseMove events from OCX controls
[ ] Report Windows messages
[ ] Report callback and hook functions

Use event reporting to solve the following problems:

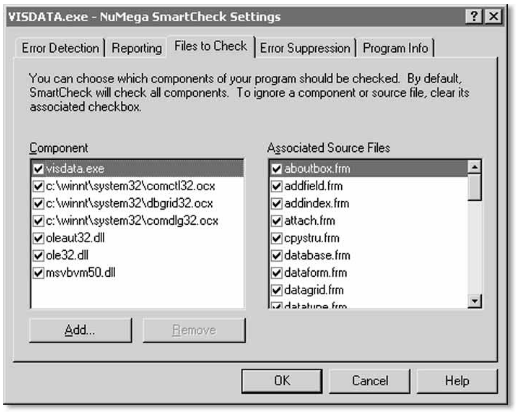
Problem Area	Suggested Analysis
Handled Visual Basic runtime errors	Diagnose hard-to-solve conditions or errors, that are not adequately addressed by your program's error-handling.
Sequences	Examine messages and how your program responds to them. For instance, did the messages come in the order you expected? Check the API calls your program made in response to messages.
Performance	Look for indications of wasted time. For instance, is your program painting a window twice in succession on two different messages? Your program may be making hundreds or even thousands of unnecessary memory allocations or file reads. You can block these allocations into a few big operations to improve performance.
Threads	Look at the thread-switching and thread interaction. This helps you debug multi-thread problems with semaphores in critical sections.
API failures	Look at the arguments passed to APIs. When pointers are passed, trace the data to which they point.

Files To Check

SmartCheck automatically checks all the source files for your program and its related static and run-time DLLs and OCXs. However, you might want to check only a specific portion of your code. For example, you might want to limit error detection to a specific module or source files that comprise a module.

To limit the code SmartCheck checks, clear the modules or source files you do not want to check.

Note Four of the listed files, MSVBVM50.DLL, MSVBVM60.DLL, OLE32.DLL, and OLEAUT32.DLL, *must remain selected* for SmartCheck to work properly.


VISDATA.exe - NuMega SmartCheck Settings

You can choose which components of your program should be checked. By default, SmartCheck will check all components. To ignore a component or source file, clear its associated checkbox.

Component	Associated Source Files
✓ visdata.exe	✓ aboutbox.frm
✓ c:\winnt\system32\comctl32.ocx	✓ addfield.frm
✓ c:\winnt\system32\dbgrid32.ocx	✓ addindex.frm
✓ c:\winnt\system32\comdlg32.ocx	✓ attach.frm
✓ oleaut32.dll	✓ cpystru.frm
✓ ole32.dll	✓ database.frm
✓ msvbvm50.dll	✓ dataform.frm
	✓ datagrid.frm
	✓ datatype.frm

Excluding Dynamically Loaded Modules from Checking

Before you check a program for the first time, the Modules list box displays only the DLLs and OCXs loaded at program startup. When checking a program, SmartCheck tracks and records all dynamically loaded modules. Subsequently, each time you display the Modules and Files tab, you will see a list that has been updated to include all the modules recorded by SmartCheck during the previous “run”. To exclude dynamically loaded modules from checking:

- Before SmartCheck has updated the list, you can add the modules to the list yourself, then clear their associated check boxes.
- After SmartCheck has updated the list, simply review the list and clear the modules or related source files that you do not want to check.

Customizing Program Information Settings

Use the Program Info settings to establish the following for your program:

- Command Line Arguments
- Working Directory
- Temporary event file directory. (This option is useful if your system drive is nearly full, since SmartCheck event files are sometimes quite large.)
- Source File Search Path

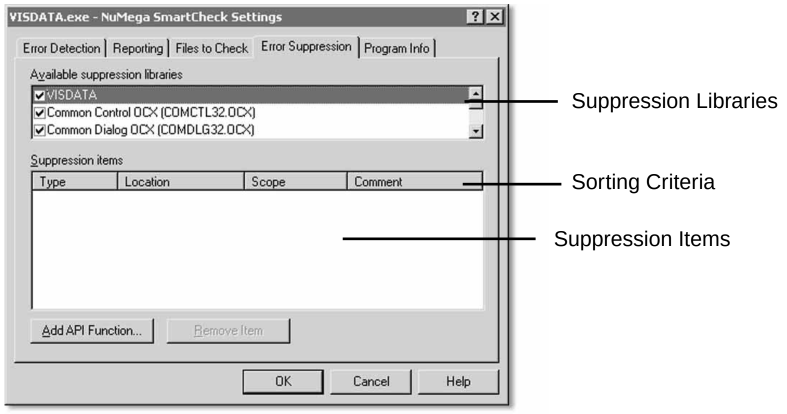
Customizing Error Suppression Settings

Each time you suppress an error, SmartCheck adds the suppression information to the suppression library. SmartCheck uses the suppression library to determine which errors to suppress for future runs of the program. In addition to the individual program libraries, SmartCheck supplies suppression libraries for common development environments, OCXs and DLLs, including Visual Basic, MFC, OWL, and the Delphi VCL.

As your work progresses, you may want to delete a suppression item, or you may want SmartCheck to check your program without referring to the library. Perform the actions in the following table to view and modify libraries and suppression items.

To	Do this
Display the list of suppression items for a particular library	Select the name of the library
Enable or disable a library	Select or clear the checkbox for the library
Sort the suppression items in a library	Click a sorting criterion above the list of items
Delete a suppression item	Select the item from the list, and then click Remove

Note: You can only delete suppression items from your program library. You cannot delete suppression items from the supplied system module libraries.


VISDATA.exe - NuMega SmartCheck Settings

Available suppression libraries
- ☑ VISDATA
- ☑ Common Control OCX (COMCTL32.OCX)
- ☑ Common Dialog OCX (COMDLG32.OCX)

Suppression items

Type	Location	Scope	Comment

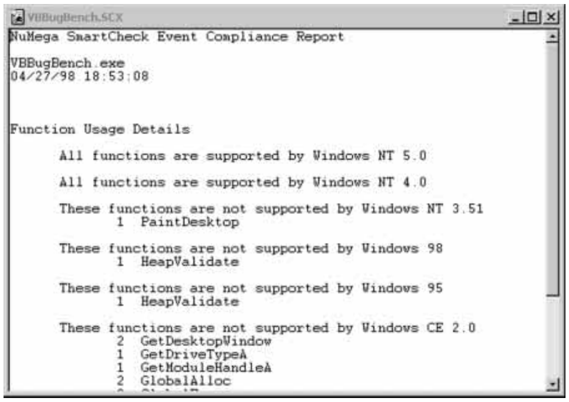
4 Checking Compliance

Microsoft provides a collection of 32-bit application programming interfaces (APIs) called Win32. Win32 is implemented under Windows NT, Windows 95, Windows 98 and Windows CE 2.0. A portion of Win32 is also implemented on Windows 3.1 as Win32s.

Although many of the APIs within Win32 support these multiple versions of Windows, some are platform specific. You can unknowingly use a call or set of calls that are available on one platform, but not another.

To assure that your program is compliant across the various Windows platforms and Win32s, SmartCheck provides compliance reports that check your program's APIs. Use these reports to determine if your program's APIs are available on all Win32 platforms or just a subset. Additionally, SmartCheck categorizes your program's use of C Run-Time Library calls into ANSI and non-ANSI and provides lists of Visual Basic functions and support routines used by your program.

The following example illustrates a compliance report.



```
NuMega SmartCheck Event Compliance Report

VBBugBench.exe
04/27/98 18:53:08


Function Usage Details

      All functions are supported by Windows NT 5.0

      All functions are supported by Windows NT 4.0

      These functions are not supported by Windows NT 3.51
             1  PaintDesktop

      These functions are not supported by Windows 98
             1  HeapValidate

      These functions are not supported by Windows 95
             1  HeapValidate

      These functions are not supported by Windows CE 2.0
             2  GetDesktopWindow
             1  GetDriveTypeA
             1  GetModuleHandleA
             2  GlobalAlloc
```

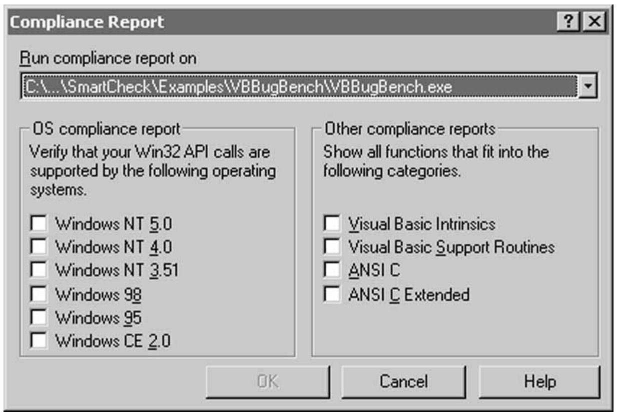
Checking API Compliance

The best strategy for ensuring API compliance is to attend to it as you design a program. Otherwise, finding and fixing API compliance-related problems can be a long and tedious task, especially if you postpone the job until after a program is completed. You can use SmartCheck compliance checking to produce either a Program Compliance or Event Compliance report:

- Program Compliance checks compliance for all functions to which your EXE file refers. This report does not check API calls made by DLLs to which the EXE points and does not check API calls made from Visual Basic code. Use the Event Compliance report to address these two issues.
- Event Compliance uses the results of your error-detection session and covers only those WIN32 , C Run-Time Library and Visual Basic functions that were called when you ran the program. This report includes calls made by DLLs to which your program points.

To Produce a Program Compliance Report

To produce a Program Compliance report:


Compliance Report

Run compliance report on: C:\...\SmartCheck\Examples\VBBugBench\VBBugBench.exe

OS compliance report
Verify that your Win32 API calls are supported by the following operating systems.
- [ ] Windows NT 5.0
- [ ] Windows NT 4.0
- [ ] Windows NT 3.51
- [ ] Windows 98
- [ ] Windows 95
- [ ] Windows CE 2.0

Other compliance reports
Show all functions that fit into the following categories.
- [ ] Visual Basic Intrinsics
- [ ] Visual Basic Support Routines
- [ ] ANSI C
- [ ] ANSI C Extended

1. Open the program for which you want to produce the report.
2. On the View menu, click Compliance Report.
3. In the Compliance Report window, at Run compliance report on, select the .EXE.
4. Select the operating systems, Visual Basic categories and versions of C to be addressed by the report.
5. Click OK.

To Produce an Event Compliance Report

To produce an Event Compliance report:

1. Make sure event reporting is enabled.
2. Check the program, using all the functions you want included in the compliance report.
3. On the View menu, click Compliance Report.
4. In the Compliance Report window, at Run compliance report on, select the Program Results item from the list.
5. Select the operating systems, Visual Basic categories and versions of C to be addressed by the report.
6. Click OK.

Index