# Comprehensive Debugging Guide for ASP.NET Framework

## Debugging

**Debugging** is the systematic process of identifying and resolving errors (bugs) in your application to ensure it functions as expected.

## Essential Debugging Methods

* **Breakpoints**: Pause code execution at specific points to examine the application state
* **Logging**: Record application behavior for post-execution analysis
* **Exception Handling**: Implement try-catch blocks to gracefully manage runtime errors
* **Debugging Windows**: Utilize Visual Studio’s specialized windows to monitor variables and execution flow
* **Step Execution**: Navigate through code line-by-line to understand execution paths

## Application Execution Modes

### Debug Mode

* Provides detailed run-time information
* Allows breakpoint functionality
* Enables access to full debugging toolset
* Maintains symbol files for variable inspection
* Typically, slower performance due to additional overhead

### Release Mode

* Optimized for performance and efficiency
* Removes debugging information
* Reduces final application file size
* Applies code optimizations
* Limited debugging capabilities

## Advanced Breakpoint Types

### Standard Breakpoint

* Pauses execution at a specific line of code
* Set by clicking in the margin or pressing F9

### Tracepoint

* Functions like breakpoint but logs a message instead of pausing
* **How to set**: Right-click breakpoint → Actions → Add message
* Useful for tracking execution flow without interrupting it

### Temporary Breakpoint

* Automatically removed after being hit once
* **How to set**: Use Breakpoint Settings window or Debug → New Breakpoint → Temporary Breakpoint
* Perfect for one-time checks without modifying breakpoint configuration

### Dependent Breakpoint

* Activates only after another specific breakpoint is hit
* **How to set**: Right-click breakpoint → Conditions → Add dependency
* Helps control complex debugging flows
* For set this we can do:

A screenshot of a computer error

AI-generated content may be incorrect.

### Function Breakpoint

* Targets a specific function rather than a line of code
* **How to set**: Debug → New Breakpoint → Function Breakpoint (Shortcut: Ctrl + D, F)
* Useful when you need to debug a method regardless of where it’s called from

A screenshot of a computer

AI-generated content may be incorrect.

## Conditional Breakpoints

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

### Conditional Expression

* Breakpoint triggers only when specified condition evaluates to true
* **Example**: counter == 10 pauses only when counter equals 10
* **Options**:
  + **Is True**: Breaks when expression evaluates to true
  + **When Changed**: Breaks when expression value changes between hits
* Ideal for debugging specific scenarios without excessive stopping

### Hit Count

* Breakpoint activates after being hit a specified number of times
* **Example**: Setting hit count to 5 will pause execution on the 5th occurrence
* Excellent for debugging loops or repetitive calls

### Filter

* Restricts breakpoint to specific threads, processes, or machines
* **Example**: ThreadID == 5 only breaks on thread 5
* Essential for multi-threaded and distributed application debugging

## Critical Debugging Windows

### Output Window (Ctrl + Alt + O)

* Displays debug logs, build messages, and exception details
* Captures Debug.WriteLine() and Console.WriteLine() output
* Shows application-level messages during execution

### Autos Window (Ctrl + Alt + V, A)

* Automatically displays variables used in current and previous lines
* Updates dynamically as you step through code
* Provides quick access to relevant variables without manual configuration

### Locals Window

* Shows all variables in the current scope/method
* Updates as execution context changes
* More comprehensive than Autos but limited to current context

### Watch Window (Ctrl + Alt + W, 1-4)

* Allows monitoring specific variables or expressions
* Persists across debugging sessions
* Supports custom expressions and evaluations
* Multiple watch windows (1-4) available for organizing different watch categories

### Call Stack Window (Ctrl + Alt + C)

* Displays the hierarchy of function calls leading to current position
* Enables navigation to any point in the call chain
* Essential for understanding execution flow and context

### Immediate Window

* Allows executing code commands during debugging
* Useful for testing expressions or modifying variables on-the-fly
* Accessible via Debug → Windows → Immediate

## Conditional Compilation

Conditional compilation enables including or excluding code sections based on defined preprocessor symbols.

### Key Directives:

* **#define**: Declares a symbol for conditional compilation
* **#if, #elif, #else, #endif**: Controls which code blocks compile
* **#pragma warning**: Manages compiler warnings

### Common Symbols:

* **DEBUG**: Automatically defined in Debug configuration
* **TRACE**: Present in both Debug and Release by default
* **Custom symbols**: Can be defined in project properties

### Example:

### **A screenshot of a computer program AI-generated content may be incorrect.**

### Use Cases:

* Environment-specific code
* Feature toggles
* Platform-specific implementations
* Diagnostic code in development builds

## Visual Studio Debugger Experience

* The current execution line is highlighted in **yellow**
* Variable values can be viewed by hovering over them during a debugging session
* IDE provides real-time information about application state

## Essential Debugging Shortcuts

| **Action** | **Shortcut** | **Description** |
| --- | --- | --- |
| Start Debugging | **F5** | Launch application with debugger attached |
| Start Without Debugging | **Ctrl + F5** | Runs application without debugger |
| Stop Debugging | **Shift + F5** | Terminates the current debugging session |
| Step Over | **F10** | Executes current line and moves to next line |
| Step Into | **F11** | Enters method calls to debug their internals |
| Step Out | **Shift + F11** | Completes current method and returns to caller |
| Toggle Breakpoint | **F9** | Adds/removes breakpoint at current line |
| Clear All Breakpoints | **Ctrl + Shift + F9** | Removes all breakpoints in solution |
| Breakpoint Window | **Ctrl + D, B** | Opens window to manage all breakpoints |

## Pro Debugging Tips

* **Use Debug.Assert()** to validate assumptions during development
* **Create custom visualizers** for complex objects to improve debugging experience
* **Use conditional breakpoints** to avoid unnecessary interruptions
* **Leverage Edit and Continue** to modify code during debugging sessions
* **Use the Exception Settings window** to control which exceptions pause execution
* **Configure symbol loading** for better debugging of external libraries
* **Try remote debugging** for server deployments and distributed applications
* **Use logging frameworks** (like Serilog or NLog) for production diagnostics