Microsoft Visual Studio

What is Microsoft Visual Studio?

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that expand the functionality at almost every level—including adding support for source control systems (like Subversion and Git) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Azure DevOps client: Team Explorer).

Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML, and CSS. Support for other languages such as Python, Ruby, Node.js, and M among others is available via plug-ins. Java (and J#) were supported in the past.

The most basic edition of Visual Studio, the Community edition, is available free of charge. The slogan for Visual Studio Community edition is "Free, fully-featured IDE for students, open-source and individual developers".

As of 8 November 2021 the current production-ready Visual Studio version is 2022, with older versions such as 2013 and 2015 on Extended Support, and 2017 and 2019 on Mainstream Support.

Architecture

Visual Studio does not support any programming language, solution or tool intrinsically; instead, it allows the plugging of functionality coded as a VSPackage. When installed, the functionality is available as a Service. The IDE provides three services: SVsSolution, which provides the ability to enumerate projects and solutions; SVsUIShell, which provides windowing and UI functionality (including tabs, toolbars, and tool windows); and SVsShell, which deals with registration of VSPackages. In addition, the IDE is also responsible for coordinating and enabling communication between services. All editors, designers, project types and other tools are implemented as VSPackages. Visual Studio uses COM to access the VSPackages. The Visual Studio SDK also includes the Managed Package Framework (MPF), which is a set of managed wrappers around the COM-interfaces that allow the Packages to be written in any CLI compliant language. However, MPF does not provide all the functionality exposed by the Visual Studio COM interfaces. The services can then be consumed for creation of other packages, which add functionality to the Visual Studio IDE.

Support for programming languages is added by using a specific VSPackage called a Language Service. A language service defines various interfaces which the VSPackage implementation can implement to add support for various functionalities. Functionalities that can be added this way include syntax coloring, statement completion, brace matching, parameter information tooltips, member lists, and error markers for background compilation.[15] If the interface is implemented, the functionality will be available for the language. Language services are implemented on a per-language basis. The implementations can reuse code from the parser or the compiler for the language. Language services can be implemented either in native code or managed code. For native code, either the native COM interfaces or the Babel Framework (part of Visual Studio SDK) can be used. For managed code, the MPF includes wrappers for writing managed language services.

Visual Studio does not include any source control support built in but it defines two alternative ways for source control systems to integrate with the IDE. A Source Control VSPackage can provide its own customised user interface. In contrast, a source control plugin using the MSSCCI (Microsoft Source Code Control Interface) provides a set of functions that are used to implement various source control functionality, with a standard Visual Studio user interface. MSSCCI was first used to integrate Visual SourceSafe with Visual Studio 6.0 but was later opened up via the Visual Studio SDK. Visual Studio .NET 2002 used MSSCCI 1.1, and Visual Studio .NET 2003 used MSSCCI 1.2. Visual Studio 2005, 2008, and 2010 use MSSCCI Version 1.3, which adds support for rename and delete propagation, as well as asynchronous opening.

Visual Studio supports running multiple instances of the environment (each with its own set of VSPackages). The instances use different registry hives (see MSDN's definition of the term "registry hive" in the sense used here) to store their configuration state and are differentiated by their AppId (Application ID). The instances are launched by an AppId-specific .exe that selects the AppId, sets the root hive, and launches the IDE. VSPackages registered for one AppId are integrated with other VSPackages for that AppId. The various product editions of Visual Studio are created using the different AppIds. The Visual Studio Express edition products are installed with their own AppIds, but the Standard, Professional, and Team Suite products share the same AppId. Consequently, one can install the Express editions side-by-side with other editions, unlike the other editions which update the same installation. The professional edition includes a superset of the VSPackages in the standard edition, and the team suite includes a superset of the VSPackages in both other editions. The AppId system is leveraged by the Visual Studio Shell in Visual Studio 2008.

Features

1. Code editor

Visual Studio (like any other IDE) includes a code editor that supports syntax highlighting and code completion using IntelliSense for variables, functions, methods, loops, and LINQ queries. IntelliSense is supported for the included languages, as well as for XML, Cascading Style Sheets, and JavaScript when developing web sites and web applications. Autocomplete suggestions appear in a modeless list box over the code editor window, in proximity of the editing cursor. In Visual Studio 2008 onwards, it can be made temporarily semi-transparent to see the code obstructed by it. The code editor is used for all supported languages.

The Visual Studio Code Editor also supports setting bookmarks in code for quick navigation. Other navigational aids include collapsing code blocks and incremental search, in addition to normal text search and regex search. The code editor also includes a multi-item clipboard and a task list. The code editor supports code snippets, which are saved templates for repetitive code and can be inserted into code and customized for the project being worked on. A management tool for code snippets is built in as well. These tools are surfaced as floating windows which can be set to automatically hide when unused or docked to the side of the screen. The Visual Studio code editor also supports code refactoring including parameter reordering, variable and method renaming, interface extraction, and encapsulation of class members inside properties, among others.

2. Debugger

Visual Studio includes a debugger that works both as a source-level debugger and as a machine-level debugger. It works with both managed code as well as native code and can be used for debugging applications written in any language supported by Visual Studio. In addition, it can also attach to running processes, monitor, and debug those processes. If source code for the running process is available, it displays the code as it is being run. If source code is not available, it can show the disassembly. The Visual Studio debugger can also create memory dumps as well as load them later for debugging. Multi-threaded programs are also supported. The debugger can be configured to be launched when an application running outside the Visual Studio environment crashes.

The Visual Studio Debugger allows setting breakpoints (which allow execution to be stopped temporarily at a certain position) and watches (which monitor the values of variables as the execution progresses). Breakpoints can be conditional, meaning they get triggered when the condition is met. Code can be stepped over, i.e., run one line (of source code) at a time. It can either step into functions to debug inside it, or step over it, i.e., the execution of the function body isn't available for manual inspection. The debugger supports Edit and Continue, i.e., it allows code to be edited as it is being debugged. When debugging, if the mouse pointer hovers over any variable, its current value is displayed in a tooltip ("data tooltips"), where it can also be modified if desired. During coding, the Visual Studio debugger lets certain functions be invoked manually from the Immediate tool window. The parameters to the method are supplied at the Immediate window.

3. Designer

Visual Studio includes a host of visual designers to aid in the development of applications. These tools include:

1. Windows Forms Designer

The Windows Forms designer is used to build GUI applications using Windows Forms. Layout can be controlled by housing the controls inside other containers or locking them to the side of the form. Controls that display data (like textbox, list box and grid view) can be bound to data sources like databases or queries. Data-bound controls can be created by dragging items from the Data Sources window onto a design surface. The UI is linked with code using an event-driven programming model. The designer generates either C# or VB.NET code for the application.

2. WPF Designer

The WPF designer, codenamed Cider, was introduced with Visual Studio 2008. Like the Windows Forms designer it supports the drag and drop metaphor. It is used to author user interfaces targeting Windows Presentation Foundation. It supports all WPF functionality including data binding and automatic layout management. It generates XAML code for the UI. The generated XAML file is compatible with Microsoft Expression Design, the designer-oriented product. The XAML code is linked with code using a code-behind model.

3. Web designer/development

Visual Studio also includes a web-site editor and designer that allows web pages to be authored by dragging and dropping widgets. It is used for developing ASP.NET applications and supports HTML, CSS and JavaScript. It uses a code-behind model to link with ASP.NET code. From Visual Studio 2008 onwards, the layout engine used by the web designer is shared with the discontinued Expression Web. There is also ASP.NET MVC support for MVC technology as a separate download and ASP.NET Dynamic Data project available from Microsoft.

4. Class designer

The Class Designer is used to author and edit the classes (including its members and their access) using UML modeling. The Class Designer can generate C# and VB.NET code outlines for the classes and methods. It can also generate class diagrams from hand-written classes.

5. Data designer

The data designer can be used to graphically edit database schemas, including typed tables, primary and foreign keys and constraints. It can also be used to design queries from the graphical view.

6. Mapping designer

From Visual Studio 2008 onwards, the mapping designer is used by LINQ to SQL to design the mapping between database schemas and the classes that encapsulate the data. The new solution from ORM approach, ADO.NET Entity Framework, replaces and improves the old technology.

4. Extensibility

Visual Studio allows developers to write extensions for Visual Studio to extend its capabilities. These extensions "plug into" Visual Studio and extend its functionality. Extensions come in the form of macros, add-ins, and packages. Macros represent repeatable tasks and actions that developers can record programmatically for saving, replaying, and distributing. Macros, however, cannot implement new commands or create tool windows. They are written using Visual Basic and are not compiled. Add-Ins provide access to the Visual Studio object model and can interact with the IDE tools. Add-Ins can be used to implement new functionality and can add new tool windows. Add-Ins are plugged into the IDE via COM and can be created in any COM-compliant languages. Packages are created using the Visual Studio SDK and provide the highest level of extensibility. They can create designers and other tools, as well as integrate other programming languages. The Visual Studio SDK provides unmanaged APIs as well as a managed API to accomplish these tasks. However, the managed API isn't as comprehensive as the unmanaged one. Extensions are supported in the Standard (and higher) versions of Visual Studio 2005. Express Editions do not support hosting extensions.

Visual Studio 2008 introduced the Visual Studio Shell that allows for development of a customized version of the IDE. The Visual Studio Shell defines a set of VSPackages that provide the functionality required in any IDE. On top of that, other packages can be added to customize the installation. The Isolated mode of the shell creates a new AppId where the packages are installed. These are to be started with a different executable. It is aimed for development of custom development environments, either for a specific language or a specific scenario. The Integrated mode installs the packages into the AppId of the Professional/Standard/Team System editions, so that the tools integrate into these editions. The Visual Studio Shell is available as a free download.

After the release of Visual Studio 2008, Microsoft created the Visual Studio Gallery. It serves as the central location for posting information about extensions to Visual Studio. Community developers as well as commercial developers can upload information about their extensions to Visual Studio .NET 2002 through Visual Studio 2010. Users of the site can rate and review the extensions to help assess the quality of extensions being posted. An extension is stored in a VSIX file. Internally a VSIX file is a ZIP file that contains some XML files, and possibly one or more DLL's. One of the main advantages of these extensions is that they do not require Administrator rights to be installed. RSS feeds to notify users on updates to the site and tagging features are also planned.

Game development with Visual Studio

The Unity engine integrates into one unparalleled platform to create 2D and 3D games and interactive content. Create once and publish to 21 platforms, including all mobile platforms, WebGL, Mac, PC, and Linux desktop, web, or consoles. Use powerful cross-platform tools to make your interactive content run beautifully on any device.

1. Debug in Visual Studio

World-class debugging for Unity games in Visual Studio

Visual Studio brings a premium debugging experience to the Unity game engine. Identify issues quickly by debugging your Unity games in Visual Studio—set breakpoints and evaluate variables and complex expressions. You can debug your Unity game that is running either in the Unity Editor or in the Unity Player, or even debug an external managed DLL in your Unity Project.

2. Enhance productivity

Full featured IDE for Unity

Write code more productively by leveraging all the productivity features that Visual Studio has to offer, such as IntelliSense, refactoring, and code browsing capabilities. Customize the coding environment exactly the way you want–choose your favorite theme, color, fonts, and all the other settings. In addition, use the Unity Project Explorer to navigate and create your Unity scripts–no more switching back and forth between multiple IDEs. Create Unity script methods inside Visual Studio quickly by using the Implement MonoBehaviours and the Quick MonoBehaviours Wizards.