Virtuoso Core Framework 1.0.0

Introduction To Virtuoso

Introduction To Virtuoso

Virtuoso is an application development workflow consisting of a framework and a content library of components. Virtuoso can be thought of as "after-market" No-Code which adds No-Code visual programming on top of existing programming workflows.

No-Code and Low-Code platforms generally offer great power in terms of quickly building useful applications at a fraction of the cost. However, they have many painful limitations:

> Vendor Lock-In: When you pick a No-Code platform to build products or tools for your business, you are locked in to that platform. Migrating to a different platform is either extremely difficult or impossible.
> Lack of Flexible: When you use a traditional No-Code or Low-Code platform, you are cut off from underlying build systems and native third generation languages (3GLs such as Java, C#, etc.) which are the source of power and flexibility.
> Lack of openness: While some No-Code and Low-Code platforms have basic plugin extensibility at best, developers have very limited ability to create content using normal Pro Code skillsets and programming experience they already possess.

Virtuoso solves all of these problems, combining the power that No-Code and Low-Code has to offer, with none of their drawbacks to the user. The end result is a native project, fully formed, with the underlying Pro Code 3GL environment and build systems completely intact. Indeed, you could build an application with Virtuoso and then completely uninstall it. Your project would still build and run like normal, as if developed from scratch using the conventional build system.

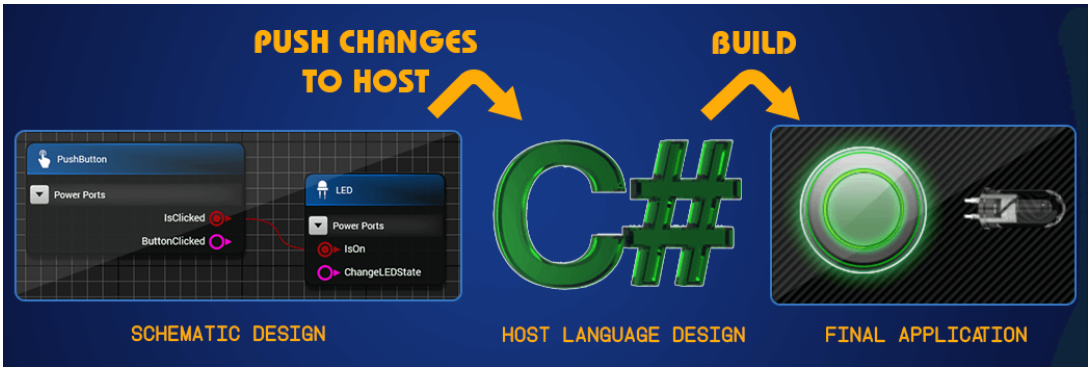
In addition to providing higher levels of abstraction, Virtuoso is focused on collaboration and composability, and includes wizards to allow developers to create No-Code components from their third-generation language (3GL) libraries. Many software solutions, such as AI, involve useful components that are only smaller parts of a larger problem that a user is trying to solve. These components or mini-solutions will continue to grow in sophistication and require more specialized knowledge, easier integration, and a digital economy. Virtuoso provides this composable software architecture, digital economy, and No Code composition of software solutions. And best of all, it's Core Framework (more on that later) and No-Code platforms are free.

So why would you actually pay money to use an inherently less powerful, flexible, and scalable technology?

The Core Framework is designed to accommodate the graphical development of an application, called the “host application”. After the application is designed graphically in the schematic editor, the design gets pushed to the host project, which is written in the language specific to that host, for example C#. Virtuoso handles creating and updating the host application according to the graphical design, essentially taking the graphical drawing and converting it to a corresponding application written in the host language.

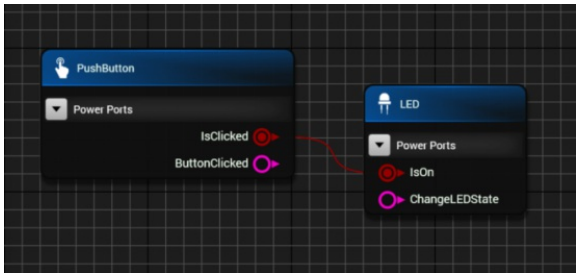
Developers can develop their application entirely in the Virtuoso schematic, as may be the case if the developer does not know the underlying host language. Or the developer can just use the schematic as a starting point for an application design, using the convenience of the graphical design and graphical library, and then extend the design with additional custom functionality written in the host language. Subsequent changes to the schematic design can be made and pushed to the host application without affecting the host code you have added yourself.

The figure below shows this general workflow. Virtuoso is used to create a host. Components are graphically added to the schematic editor and configured, with ports connected as necessary. The Core Framework then emits the host code into the project needed to implement the schematic, using the optimum design pattern specific to the host language. For example, for a WPF C#/.NET host, the schematic editor will produce a design implemented with a Model-View-View Model design pattern as a starting point, and the developer can from there add and extend the functionality of the host directly in C# and XAML.



This workflow is ideal for applications that can be naturally expressed declaratively as a set of components logically interacting with other components in some way. The components may only represent code execution of business logic, or the components may represent controls which the user interacts with on the application view. In the Virtuoso host design paradigm, there is thus a schematic layer to the design and a visual layer. Each schematic component may or may not have a corresponding visual control on the view for user interaction. The logical component is represented as a model or view model, and the visual component is represented as a view bound to the corresponding view model. For a component that has a visual element, there is always a one-to-one correspondence between the component’s view on the visual layer and the component’s view model on the schematic layer.

In the example below, the schematic on the left shows a push button component and an LED component. The components have a boolean output port and a boolean input port, respectively. The ports are connected, representing that when the push button is pressed (true), the LED should be on. The view on the right shows the corresponding user interface. After placing the components on the schematic editor, making the connection between the ports, and pushing the updates to the host, the user simply needs to position the push button and LED view components, build the project, and run.



The Virtuoso Core Framework is built to be completely extensible to new host languages, new port types, and new components. You can easily write so-called “port metadata” classes which define new port types and defines the code to be emitted to represent a logical connection between that port type and any other port type, for a given host language. You can then write new components to use those port types, and then use Virtuoso to quickly create applications specific to your application domain using a schematic design flow.

The Virtuoso Core Framework is free for development purposes, as described in the Core Framework End User License Agreement. You may also deploy applications built using the Core Framework for commercial purposes, in some cases free and in other cases subject to a 5% royalty on gross revenue, also as described in the Core Framework EULA.

To summarize, the Core Framework is designed for both power and flexibility. It allows applications to be developed graphically without requiring knowledge of the host language, using an ever-expanding content library of components. Its extensible architecture allows you to customize and extend the functionality and components of Virtuoso to enhance existing ecosystems or forge new ones. It also allows any host application designed initially in the schematic with Virtuoso components to be further developed by host language developers using the same application paradigm they are used to. Virtuoso components emitted to the host appear just like any normal class instance and can be interacted with in the same way.

Licensing

The Virtuoso Core Framework is free for development purposes, as described in the Core Framework End User License Agreement and the Virtuoso Pricing page. You may also deploy applications built using the Core Framework for commercial purposes, in some cases free and in other cases subject to a 5% royalty on gross revenue, also as described in the Core Framework EULA.

This documentation describes separate technology stacks and software for the Core Framework and No-Code host platforms. The Core Framework software consists of the common elements of visual programming, including the schematic editor itself and a scalable toolbox of components delivered as Montage packages. A specific No-Code host platform, such as the C# WPF desktop application host platform is built as a plugin extension of the Core Framework. While they are distinct software elements from a technical standpoint, they are both considered a part of the Core Framework, as are the Virtuoso component and port wizards, and they are all covered under the Core Framework End User License Agreement.

Host component libraries such as the Virtuoso.StandardLibrary package are licensed according to the Virtuoso Host Component End User License Agreement. Packages licensed under the host component EULA are free to use. The embedded hardware virtualization toolkit, Virtuoso.EmbeddedToolkit.VS2019, is an exception, however. See the Virtuoso Embedded Toolkit End User License Agreement for the licensing details, and the Virtuoso Pricing page for the latest pricing information.

Getting Started

You will need a Montage account to use Virtuoso. Creating an account is free and quick. You will need to decide whether you will use Montage as an individual or as a member of an organization. Any products you may purchase for no-code content will all be centrally managed in your account for you. This section provides basic instructions for getting started with Montage. For the complete instructions visit Montage's Getting Started section. Here, we take you straight to the Montage registration page, where you will need to register your account.

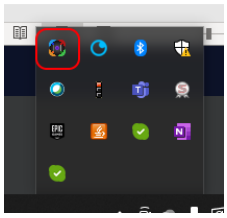
https://www.montage-software.com/register

Next, you will need to install the Montage Launcher, found here:

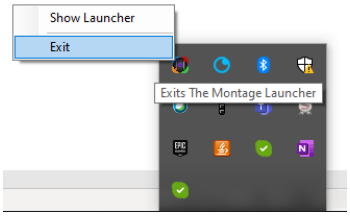
https://www.montage-software.com/MontageLauncher/Downloads

Note that you could also register using the Montage Launcher. Download the Montage Launcher Installer and then run it as administrator by right-clicking the installer and clicking "Run as administrator". Once this is installed, run the Montage Launcher.

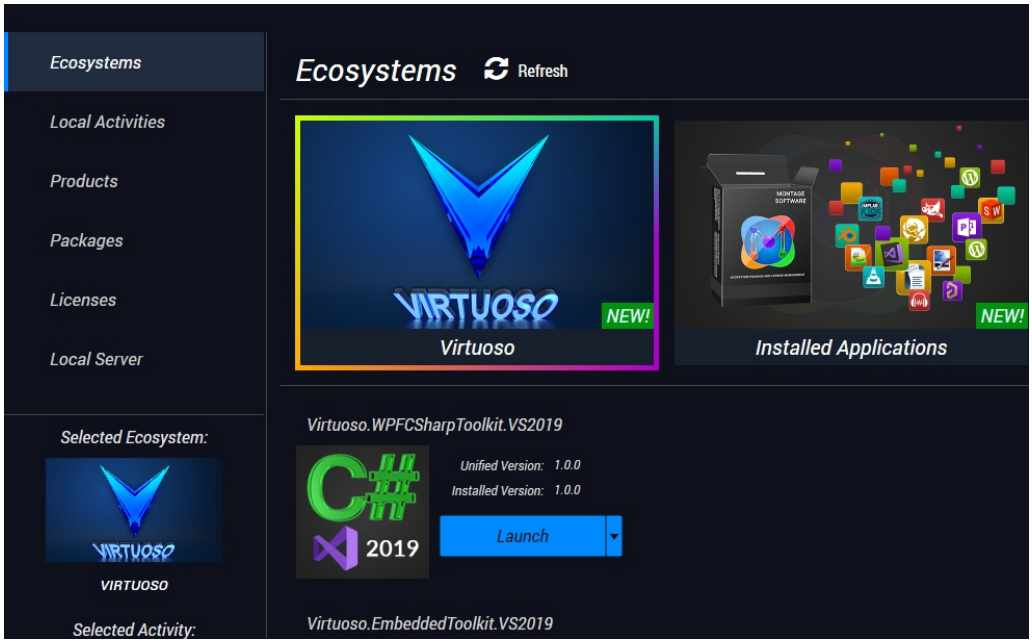
The Montage Launcher has a user interface for managing everything locally on your computer, and also runs a local server in the background to support Montage ecosystem requests. You will see the Montage Launcher icon in the task tray when it is running:



When the Montage Launcher is closed by clicking the "Close" button in the top left corner, it is still running its local server that allow it to support ecosystems that use it for services. To close the server as well, you need to right-click on the Montage task tray icon and click "Exit", as shown below. Or you can click "Show Launcher" if it is hidden and you want to see the Launcher UI, or "Hide Launcher" to hide it if it is visible.



Run the Launcher, and then you will either need to sign in or register. Once signed in, you will see several tabs on the left. Select the "Ecosystems" tab. You will see a list of Montage ecosystems along the top, make sure Montage is selected as shown below. The selected ecosystem determins which products and packages you will see in the "Products" and "Packages" tabs, thus narrowing the focus of your use of the Launcher, since Montage is designed to support any ecosystem.



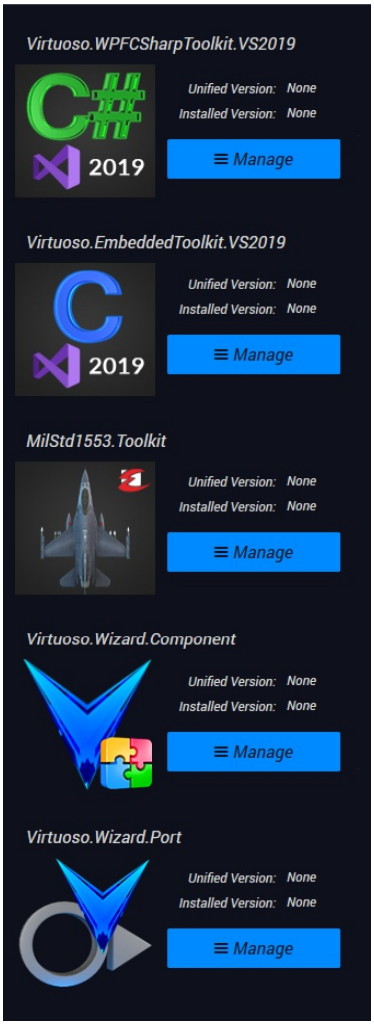
When you select the Virtuoso ecosystem, immediately below you will see the "bootstrap" packages listed by the ecosystem, which are commonly used to help bootsrap the use of the ecosystem. These packages are only provided here for convenience, and accessing them here is no different than accesing them under the "Packages" section.

As you use Virtuoso, you will come to understand that package dependency management is a core concept of both Virtuoso and Montage. Montage's job is to make sure that everything is installed and set up properly, even if you aren't familiar with the tools and don't know what needs to be set up.

You will also learn that with Montage everything is focused around an "Activity", which is like a project, but includes all tools and configurations needed for the project to run. The Activity is the entire dependency graph that describes what an Activity needs to run, for both packages inside the Activity's project(s), and on the system. This ensures portability of the Activity.

In addition to the installation issues of an Activity's graph, Montage also manages licensing or other issues related to the use of the Activity. As you install or attempt to use packages, you may find that you have an issue that needs to be resolved, such as accepting a license agreement or purchasing a product. Montage allows these issues to be resolved quickly and efficiently on behalf of the supported ecosystem.

To begin to understand package dependencies, consider the Virtuoso bootstrap packages shown below.



One thing we might eventually realize is that all of these packages are installed "Side-By-Side", meaning that only one version of the packages can be installed at a time. The Virtuoso.WPFCSharpToolkit.VS2019 package is the Virtuoso Core Framework No-Code platform for building C# WPF desktop applications. We could install that by itself, and with just that package, we would be able to create C# WPF desktop applications.

The Virtuoso.EmbeddedToolkit.VS2019 package provides content to allow embedded systems to be virtualized, and for embedded C/C++ applications to be hosted in a C# application. This package depends on the Virtuoso.WPFCSharpToolkit.VS2019 package, so we could install both by simply installing the Virtuoso.EmbeddedToolkit.VS2019 package.

Similarly, the MilStd1553.Toolkit package is a bundle package that delivers multiple Virtuoso content packages. This package depends on the Virtuoso.EmbeddedToolkit.VS2019 package, which in turn depends on the Virtuoso.WPFCSharpToolkit.VS2019 package. We can install all three by simply installing this package.

The Virtuoso.Wizard.Component and Virtuoso.Wizard.Port packages are tools used to build component packages and port packages, respectively. These are stand-alone packages and can be installed separately.

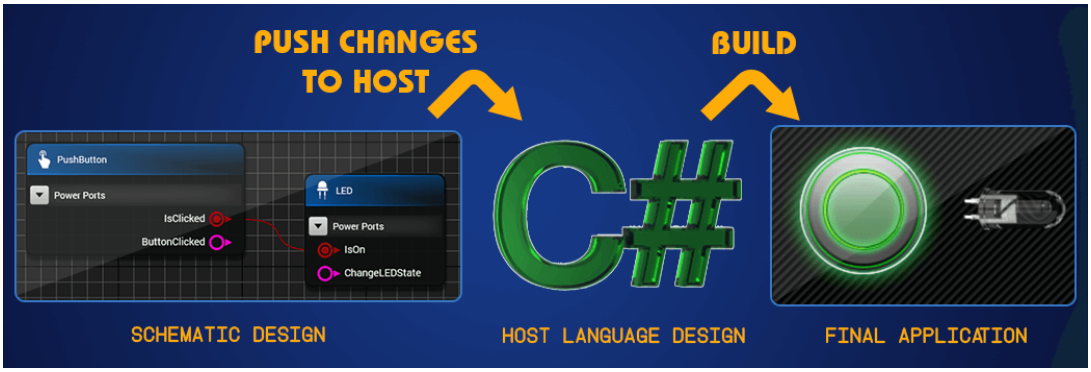
We will assume you are here to use Virtuoso for hardware virtualization, so we will install the Virtuoso.EmbeddedToolkit.VS2019 package. If you are just wanting to build C# WPF desktop applications or another platform, you can just select that paltform toolkit. Since this package depends on Visual Studio 2019, Montage will guide you through the installation of Visual Studio 2019 as well if it has not been installed.

The Virtuoso Design Workflow

As was mentioned, Virtuoso can be thought of as an "after-market" No-Code solution, which adds No-Code to any already-existing native workflow. The No-Code programming environment is a visual node-based visual programming environment, where components are dragged and dropped, configured, and connected to other components. This is the high-level design environment where citizen developers can build powerful applications without any programming experience. We call the application built by Virtuoso the "host application", as it notionally hosts the components of the high-level design, and indeed can create hierarchical compositions of multipe applications. The high-level design is then pushed to the host project, which is written in the language specific to the host, for example C#, and using the native build system, such as Visual Studio. Virtuoso handles creating and updating the host application according to the graphical design, essentially taking the graphical drawing and converting it to a corresponding application written in the host language.

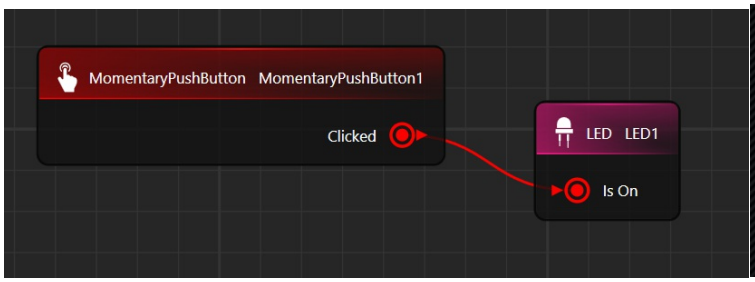
Developers can develop their application entirely in the Virtuoso schematic, as may be the case if the developer does not know the underlying host language. Or the developer can just use the schematic as a starting point for an application design, using the convenience of the graphical design and graphical library, and then extend the design with additional custom functionality written in the host language. Subsequent changes to the schematic design can be made and pushed to the host application without affecting any 3GL host code you have added yourself.

The figure below shows this general workflow. Virtuoso is used to create a host. Components are graphically added to the schematic editor and configured, with ports connected as necessary. Virtuoso then emits the host code into the project needed to implement the schematic, using the optimum design pattern specific to the host language. For example, for a WPF C#/.NET host, the schematic editor will produce a design implemented with a Model-View-View Model design pattern as a starting point, and the developer can from there add and extend the functionality of the host directly in C# and XAML.

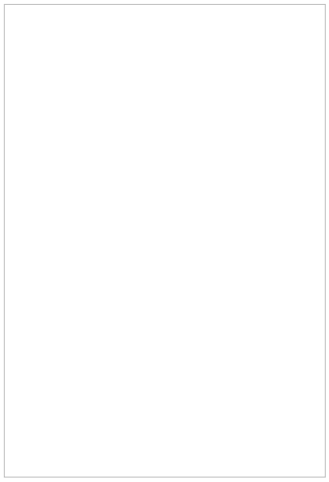


This workflow is ideal for applications that can be naturally expressed declaratively as a set of components logically interacting with other components in some way. The components may only represent code execution of business logic without any user interaction, or the components may represent controls which the user interacts with on the application view. Each schematic component may or may not have a corresponding visual control on the view for user interaction. For a C#/WPF application, the logical component is represented as a model or view model, and the visual component is represented as a view bound to the corresponding view model. For a component that has a visual element, there is always a one-to-one correspondence between the component's view on the visual layer and the component's view model on the schematic layer.

In the example below, the schematic on the left shows a push button component and an LED component. The components have a boolean output port and a boolean input port, respectively. The ports are connected, representing that when the push button is pressed (true), the LED should be on. The view on the right shows the corresponding user interface. After placing the components on the schematic editor, making the connection between the ports, and pushing the updates to the host, the user simply needs to position the push button and LED view components, build the project, and run. Watch it here.



Each component can contribute a property editor window and preview window to allow the component instances to be easily configured and previewed without programming. In the example below, different properties of the No-Code LED are easily configured from the property window, and previewed in the preview window.



These windows are developed by the component author with framework support, and are seamlessly integrated into the schematic editor for a unified and intuitive design experience. The property editor can handle all configuration in the embedded window, or can be used to launch more sophisticated applications used to easily configure the component. There is no limit to how much complexity a single component is able to represent.