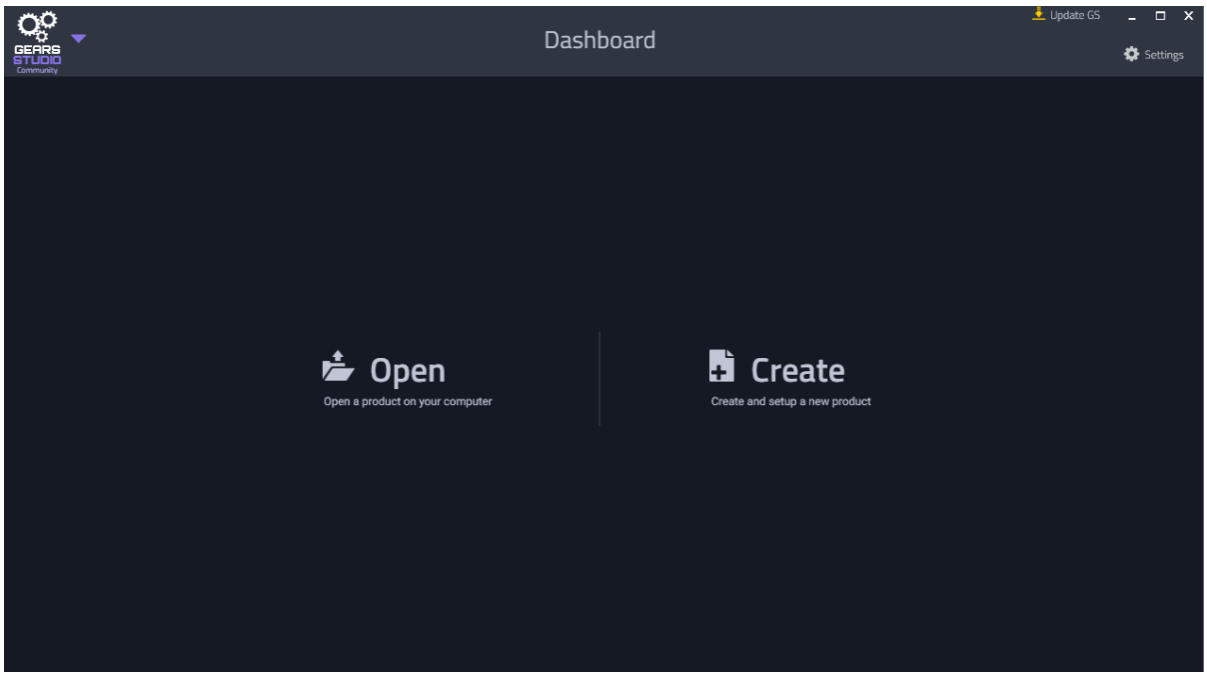
# Gears Studio

Community Edition

Version 18.2.1 WIP



# Documentation Legal Notice

## Customer Support

For specific support with Gears Studio:

- Post an issue at https://gitlab.com/bisimulations/GearsStudio/issues.

**Note:** Gears Studio outputs a log file to the installation folder called `GearsStudioLog.txt`. When reporting issues, include this file - it is overwritten each time Gears Studio starts.

"Known Issues" (page 70) describes issues known at the time of release and potential workarounds.

# Contents

# 1. Gears Studio Overview

Gears Studio is a development environment to facilitate the development of products, components, and APIs within a framework intended to avoid common pitfalls and to automate common tasks encountered in software development.

Bohemia Interactive Simulations provides a baseline unlicensed product, "Gears Studio Community Edition" (page 8), that provides everything required to streamline the development of Gears Products. "Gears Studio Professional Edition" (page 8) is an extended product, requiring a license, that adds support for code signing and JFrog Artifactory integration.

Gears Studio runs Professional Edition if the appropriate WIBU license is detected. Otherwise, it defaults to Community Edition. After installing Gears Studio, you can check the edition by looking under the icon in the top left of the application.

**Gears Studio Edition:** You are reading the **Gears Studio Community Edition** documentation.

Gears Studio provides the following features:

| Feature | Community Edition | Professional Edition |
|---|---|---|
| Create components from a template through a guided process. | ✔ | ✔ |
| Edit APIs with real-time error reporting. | ✔ | ✔ |
| Version APIs to facilitate collaborative development. | ✔ | ✔ |
| Manage groups of components as products. | ✔ | ✔ |
| Edit all components from a single Microsoft Visual Studio solution. | ✔ | ✔ |
| Build server support for continuous integration and unit testing. | ✔ | ✔ |
| Integration with JFrog Artifactory to version and distribute products, components, and APIs. | | ✔ |
| Clone component source code to the product Development Directory. | | ✔ |
| Sign components to allow them to be loaded by secure applications. | | ✔ |
| Technical support. | | ✔ |

Read the following documentation to familiarize yourself with Gears Studio:

- Review the "Gears Development Framework" (page 9).
- After deployment, start Gears Studio for the first time to perform "Gears Studio Setup" (page 14).

  For information about Gears Studio deployment, see the Gears Studio Deployment Guide.
- To learn the essentials of Gears Studio development, see "Getting Started with Gears Studio" (page 17).
- To navigate the user interface, see "Gears Studio UI Overview" (page 33).

Use Gears Studio to manage your development:

- "Developing Products" (page 46)
- "Developing Components" (page 49)
- "Developing APIs" (page 58)

Keep Gears Studio and your settings up-to-date:

- "Configure Gears Studio Settings" (page 64)
- "Update Gears Studio" (page 68)

Review "Known Issues" (page 70) to understand the current limitations of Gears Studio.

## 1.1. Gears Studio Community Edition

Gears Studio Community enables developers to create and edit Gears Components and manage them as products. Guiding them through the process of setting up a new component, real-time compilation to facilitate API development, and generating all the necessary boilerplate code required for a component to operate in the Gears framework.

Gears Studio Community also provides versioning functionality for APIs to enable developers to cooperate without affecting their individual development. Developers can work with static versions and update to later versions when ready.

Gears Studio Community Edition does not support the following features:

- Integration with JFrog Artifactory to version and distribute products, components, and APIs.
- Clone component source code to the product Development Directory.
- Sign components to allow them to be loaded by secure applications.
- Technical support.

To update to Gears Studio Professional Edition and gain access to these features, contact our sales department at sales@bisimulations.com.

## 1.2. Gears Studio Professional Edition

Gears Studio Professional adds the ability to sign a Gears Component DLL with a trusted certificate. Signing a Gears Component DLL with a trusted certificate does not encrypt the DLL. Signed Gears Components still export functions like any other DLL. Applications that support code signing only load components signed with the proper trusted certificates.

Gears Studio Professional also integrates JFrog Artifactory into Gears Product development. JFrog Artifactory is a specialized file server that stores and helps to manage binaries. This enables the publishing of APIs, components, and products to JFrog Artifactory Server to be shared with other developers. Libraries, SDKs, and other resources can also be published to JFrog Artifactory Server to enable developers to import them using Gears Studio.

# 2. Gears Development Framework

The Gears framework and Gears Studio were created to help prevent common development pitfalls and to automate common tasks encountered in software development.

Gears uses a component-based architecture to promote rapid development by building applications from self-contained systems that communicate using formally defined interfaces. This framework enables functionality to be reused and avoids the complexity of tightly coupled systems.

Gears provides the advantages of a component-based architecture while Gears Studio automates tasks to facilitate its implementation.

Review the following concepts to understand how Gears Studio manages the development hierarchy:

- "Product Concepts" (page 9)
- "Component Concepts" (page 10)
- "API Concepts" (page 10)
- "Development Files and Folders" (page 11)

## 2.1. Product Concepts

Products are a way to organize, group, and manage components that are related. They are made up of a series of components that all have their individual purposes and areas of responsibility. For example networking, camera, and animation components. When combined, these components work together to form the product.

Review the following concepts before "Developing Products" (page 46).

- "Product Directories" (page 9)

### 2.1.1. Product Directories

Gears Product files exist in two distinct directories, the Application Directory and the Development Directory, in order to separate the runtime product from the product source code.

When you create or import a product, you must specify local Application and Development directories.

- **Application Directory**

  The Application Directory contains the executable and component binaries that form the runtime product.

  Components that are compiled in your Code Editor are deployed directly to this directory in an Active state, ready for product execution. For more information, see "Component States" (page 10).

- **Development Directory**

  The Development Directory contains all the source control repositories and dependencies for the components, ready to use when working with the code. When you create or import a product, this must be an empty directory.

  Components that are imported from a remote repository using Source Control are deployed to this directory, ready for development.

  The Development Directory also contains all the files and metadata that Gears Studio uses to manage your product development. For more information about Active and Inactive states, see "Component States" (page 10).

For more information about Package and Source Mode, see "Component Concepts" (page 10).

## 2.2. Component Concepts

A Gears Component consists of a Dynamic-Link Library (DLL) and resource files. The DLL provides the component services. Every Gears Component implements the Component API to communicate with the underlying Gears framework, as well as custom APIs to provide its services.

Review the following concepts before "Developing Components" (page 49).

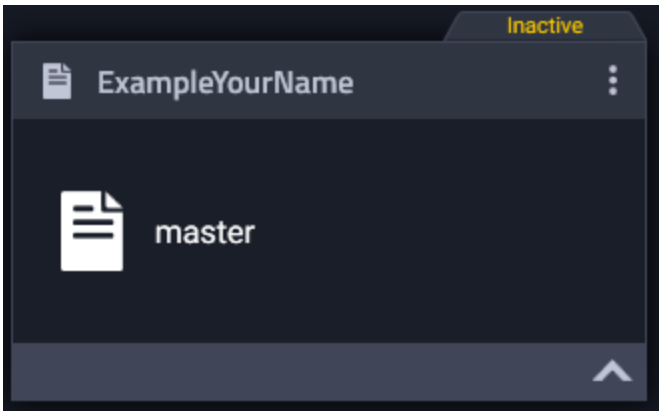
- "Component States" (page 10)

### 2.2.1. Component States

Component States indicate whether the component has binaries deployed in the Application Directory, ready to load when the product runs.

- **Inactive**

  Initially, newly created components do not have component binaries deployed to the Application Directory and are marked as **Inactive**, indicated by a tab on the "Component Card" (page 38).

  When a component is built in the Code Editor, the binaries deploy to the Application Directory and the Inactive indication is removed.

  Components with deployed binaries can be switched to an Inactive State. In this cases, the component is not loaded by the product application at runtime.

- **Active**

  Active components have been built in the Code Editor and their binaries deployed to the Application Directory. The product application loads active components when executed.

  Before executing a product, ensure that all the components you require are set to Active.

Build components in your Code Editor to make them Active and then use "Component Tools" (page 39) to manage component states.

## 2.3. API Concepts

An API is a well defined set of functions and data structures that enforce the policies and procedures for accessing a service.

Components in the Gears framework communicate using APIs. These APIs must conform to a very specific set of rules. For instance, the API must adhere to certain C Programming language standards. Using Gears Studio to create APIs ensure that these rules are followed.

Review "API Versioning" (page 11) before "Developing APIs" (page 58).

### 2.3.1. API Versioning

Use API versioning to provide backwards compatibility for APIs that may already be in use in delivered products.

Gears Studio supports the following choices for API versioning:

- **Revisions**

  Use Revisions for APIs that are still in development and have not been widely released or adopted.

  Revisions are not backwards compatible, and you can only implement a single major version of an API. For example, you cannot implement both `1-rev1` and `1-rev2`, and you must choose between them.

  Revisions provide more flexibility in what you can change; you are completely revising the previous version of the API.

- **Major Versions**

  Use Major Versions to make API changes when an API has already been released and adopted when you want to maintain the previous release.

  Major versions are completely backwards compatible, you can implement both versions 1 and 2 of an API to maintain the old functionality, while also adding new options. For example, version 1 could enable text to be displayed and version 2 could enable the ability to specify the color of the text. Version 1 continues to work, it just has more limited functionality.

  Due to backwards compatibility restrictions, Major Versions are more limited than Revisions. You cannot change existing typedefs, enums, or constants when making a new Major Version, because it would break previous versions if you did so. You can of course add new items of these types.

It is best practice to create a new revision or major version before making changes to an API.

## 2.4. Development Files and Folders

Gears Studio manages all component source code and product metadata in the Development Directory for the product.

Gears Studio generates Code Solutions that separate or partition the files developers use to modify their code from the behind the scenes details that Gears Studio uses.

The Development Directory contains the following folders:

- A named folder for each component the product uses.

  See "Component Folders" (page 11).

- A `.gearsstudio` folder that contains the metadata used by Gears Studio and the dependencies required to build the product.

  See "Project Files and Property Sheets" (page 13).

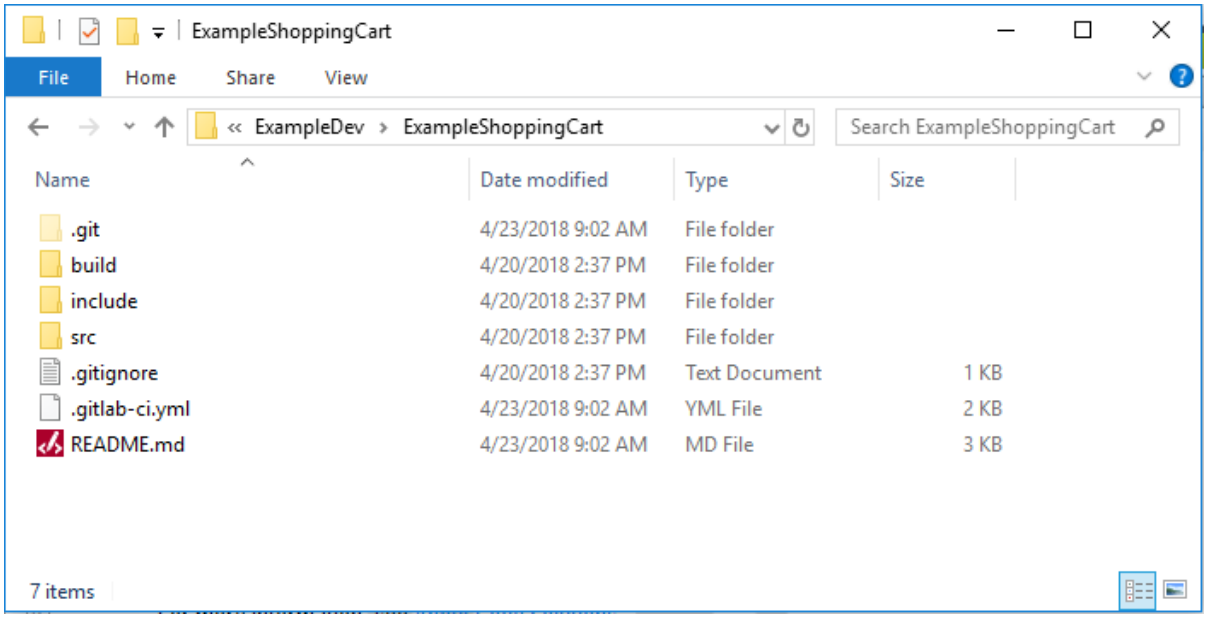
#### 0.0.1. Component Folders

Each component folder contains the Code Solution for the component generated by Gears Studio.

The files and folders are created while "Creating Components" (page 49) and updated when "Generating Components" (page 53) and "Using Code Editors" (page 54).

The code solution for each component is contained within a folder matching the name of the component.




Each component code solution initially contains the following folders:

- **build** - Contains files related to building the component.
- **include** - Contains header files and interface definition files.
- **src** - Contains source files, split into "Component Projects" (page 12).
- **.git** - Source control folder.

Another folder appears when the Gears Component is built for the first time.

- **bin** - Contains the component binaries and the debug database files.

Additional folders may be manually created and placed in the root directory of a Gears Component.

- **res** - The contents of this folder are copied to the root of the Application Directory.
- **runtime** - The contents of this folder are copied to the component folder in the Application Directory.
- **testdata** - The contents of this folder are copied alongside the UnitTest executable.

If created, these optional folders supplement the Gears development process by allowing a developer to incorporate external resources.

For example, if a component needs a preconfigured database to perform its service, the developer would:

- Manually create a folder named `runtime` in the root directory of the component.
- Copy the database file into the runtime folder.
- In code, access the database resource by combining the `component_directory` path (provided during the component startup phase) and the file name of the database.

Unit testing often requires test files that contain both valid and erroneous data. These files go in the `testdata` folder. When the UnitTest project invokes its post-build event commands, the `testdata` folder and it contents are copied alongside the UnitTest executable. Accessing the UnitTest files from the UnitTest source code looks something like this:

```
const char* test_file_path = ".\\testdata\\my_test_file.txt";
```

### 2.4.1. Component Projects

Gears Studio creates three separate projects to manage the source code:

- ***ComponentName***

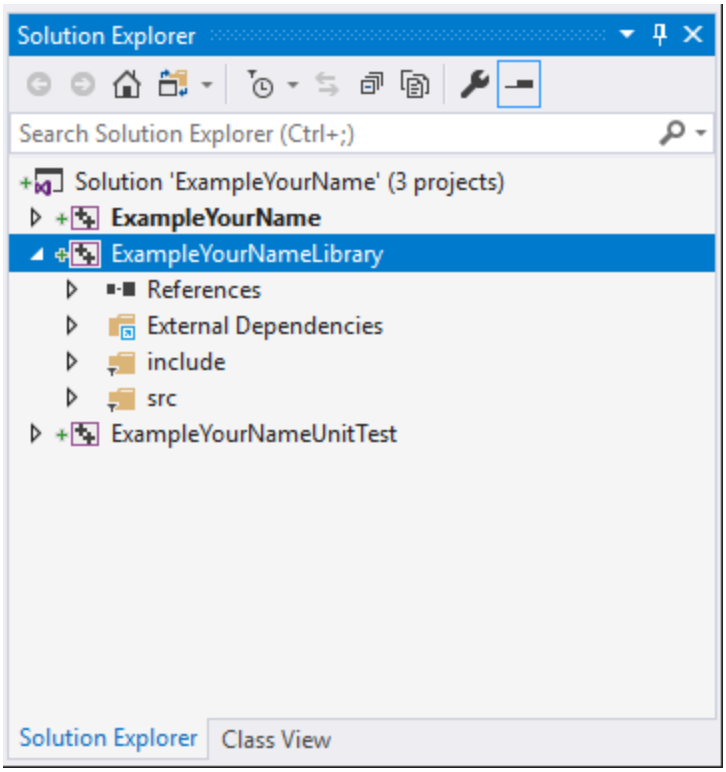
  This project creates the DLL loaded by your application at runtime. The code in this project is automatically generated by Gears Studio and should not be modified. Instead, modify code in the **Library** project that it links to.

- ***ComponentName*Library**

  This project contains the actual working component. It contains the logic for implemented APIs and makes calls to external APIs. Use this project to modify your component.

- ***ComponentName*UnitTest**

  This project contains unit tests to ensure the quality of your code. Initially there is a stub test that always passes. This project also links to the **Library** project to simplify testing your code.




## 2.4.2. Project Files and Property Sheets

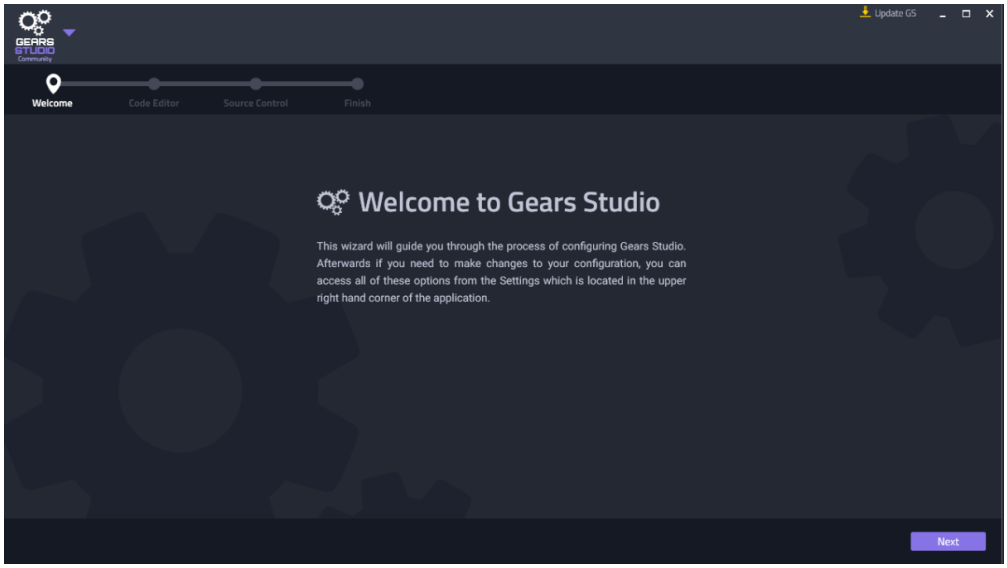
The `.gearsstudio` folder contains data used by Gears Studio to manage your products, dependencies, and code solutions. For example, the additional include directories that projects need in order to compile external API files. The files in this folder should not be manually modified.

Also, there are post-build events that execute when a project is built in the Code Editor. Both the Component and UnitTest projects have commands that run after a successful compilation of the source code. These commands are responsible for deploying component binaries, external resources, and unit test data files to the appropriate Application Directory paths.

These project settings are established using property sheets. Property sheets provide a way for Gears Studio to expand on the code project settings, without actually modifying the project file itself. This setup enables developers to modify their project files freely because the risk of breaking a setting that Gears Studio needs to work correctly is minimal.

# 3. Gears Studio Setup

When you start Gears Studio for the first time, the Setup Wizard opens to configure your development environment.




This process integrates Gears Studio with your development tools:

- Setting your default Code Editor.
- Setting your Source Control Client.

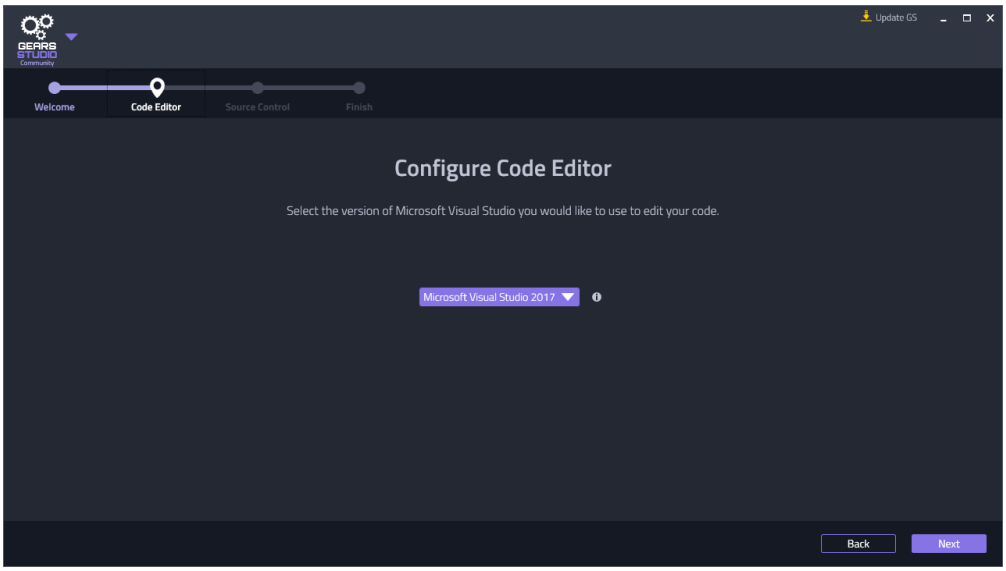
**Follow these steps:**

1. After Gears Studio installation, start Gears Studio for the first time.

   Gears Studio opens the Setup Welcome Page.

2. Click **Next** to set your preferred Code Editor.

   Gears Studio displays the Configure Code Editor page.




3. Use the drop-down to select your preferred Code Editor.

   Gears Studio supports the following Code Editors:

- Microsoft Visual Studio 2010
- Microsoft Visual Studio 2012
- Microsoft Visual Studio 2013
- Microsoft Visual Studio 2015
- Microsoft Visual Studio 2017

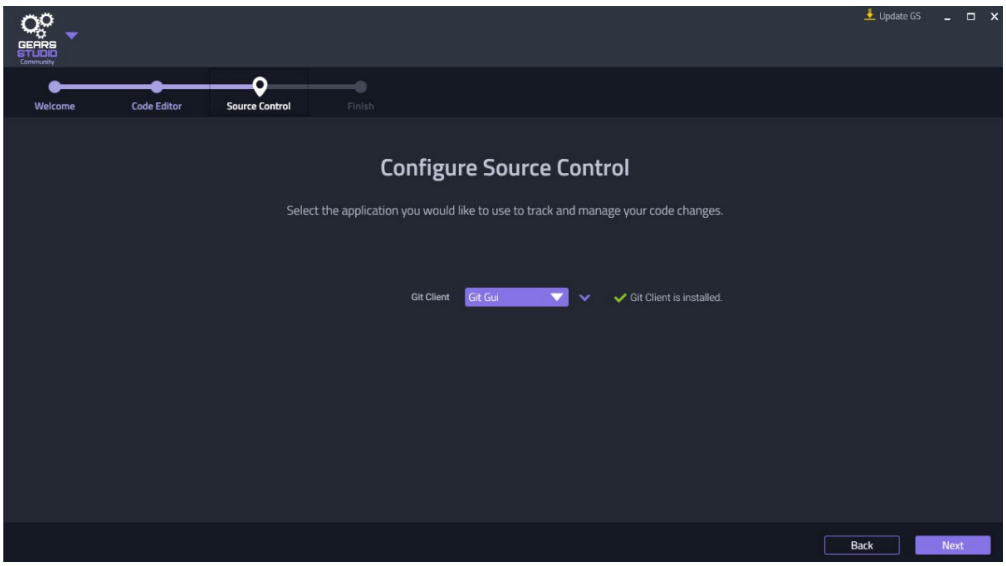
Gears Studio uses the Code Editor setting to build the appropriate Code Solutions and opens the specified Code Editor when required.

**Note:** The None option allows the user to select a Code Editor at a later time. When the none option is selected, Gears Studio will not build solution and project files when the Generate button is pressed and will not launch a code editor when the Code Editor button or Super Solution buttons are pressed.

Build Solution and Project files when the Generate button is pressed.

Launch a Code Editor when the Code Editor or Super Solution buttons are pressed.

4. Click **Next** to open the Configure Source Control page.

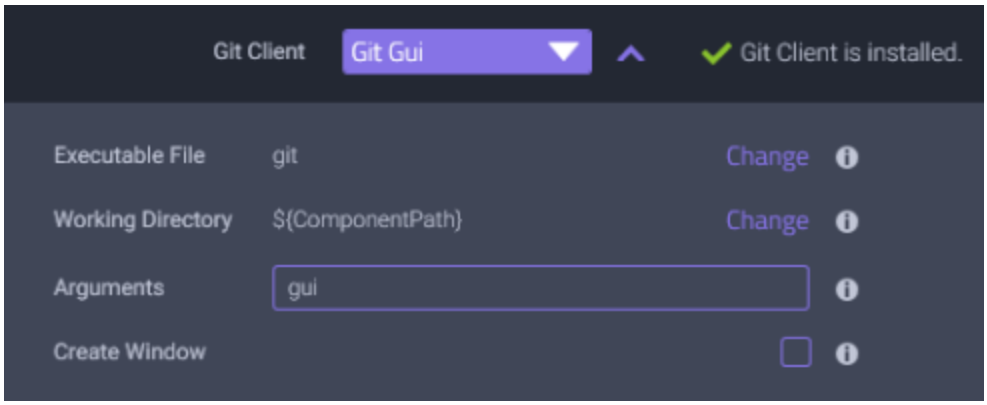



5. Use the **Git Client** drop-down to select your preferred source control client.

   The **Git Client is installed** message indicates whether Gears Studio can detect the selected client.

   To configure advanced settings, **follow these steps**:

   1. Click the **Arrow Icon** to expand the advanced settings.



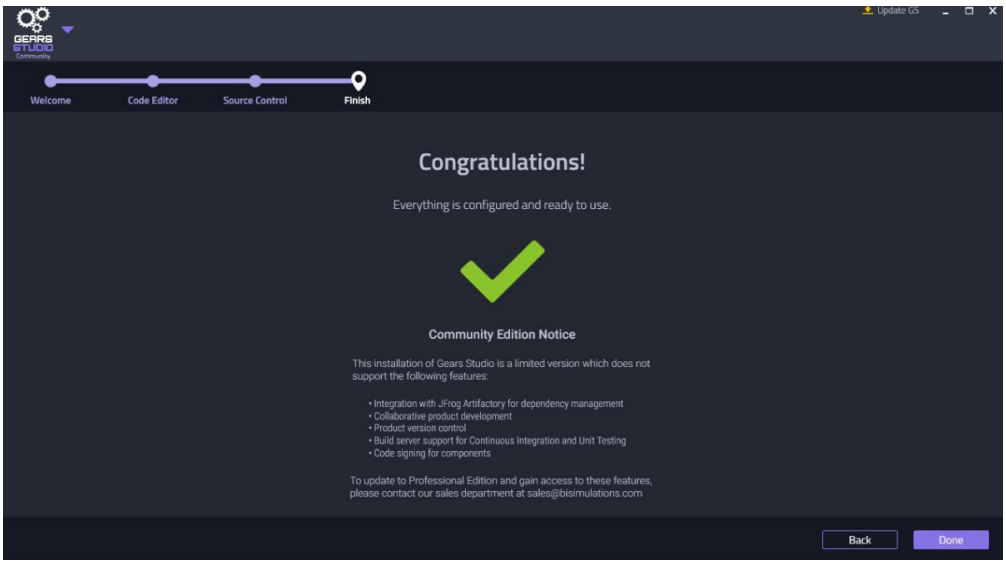
2. To configure the **Executable File**, click **Change**, browse for your source client executable and click **OK**.
3. To configure the **Working Directory** that contains the source control client executable, click **Change**, browse for the directory, and click **OK**.
4. To launch the source control client with specific arguments, modify the **Arguments** input:

   If the argument variable `${ComponentPath}` is specified in the argument list, it is replaced with the path to the Component being opened.

   For example: `--open "${ComponentPath}"`
5. Select **Create Window** to specify that the source control client should launch in a new window.

Gears Studio operations that launch a source control client on behalf of the user launch the selected source control application.

6. Click **Next** to open the Finish Page.




7. Click **Done** to exit the Setup Wizard.

Gears Studio is ready to use.

To change any of these settings later, see "Configure Gears Studio Settings" (page 64).

# 4. Getting Started with Gears Studio

This chapter uses working examples to guide you through the basic principles of using Gears Studio. We strongly advise following this tutorial before using Gears Studio for your own development.

**Follow this process:**

1. "Set Up the Example Product" (page 17)
2. "Set Up the Example Component" (page 19)
3. "Using the Example Code Solution" (page 24)
4. "Editing the Example Component" (page 27)
5. "Using the Example Super Solution" (page 30)

After completing this example process you should be ready to start developing your own Gears Products, Components, and APIs.

Review the "Gears Studio UI Overview" (page 33) next and then start "Developing Products" (page 46).

The following additional topics provide reference material for recurring development tasks:

- "Example Build Server Setup" (page 56)

## 4.1. Set Up the Example Product

Use the following process to set up a new example product, **ExampleMart**:

1. "Create the Example Product" (page 17)
2. "Import the Example Component" (page 18)

Complete this process and then continue to "Set Up the Example Component" (page 19) to add functionality to the product.

For more information about developing products, see "Developing Products" (page 46).

### 4.1.1. Create the Example Product

The initial step in the example is to create a new product called **ExampleMart**.

**Follow these steps:**

1. In the Gears Studio Dashboard, click **Create**.

   The Name Product panel opens.

2. In the Name Product panel, input **ExampleMart**.

   The Set Directories panel opens.

3. In the **Set Directories** panel, set the Development Directory.

   Click **Change**, browse for a folder, and click **OK**.

   Gears Studio uses this folder to store your code for the product and it must be empty.

   The default directory is: `<Gears Studio Installation>/Products`.

4. In the **Set Directories** panel, set the Application Directory.

   Select **Custom**, click **Change**, browse to the `ExampleMart` folder in the installation directory, and click **OK**.

Gears Studio uses this folder to store the component binaries and product application.

5. Click **Create**.

Gears Studio creates the ExampleMart product and opens the "Product Page" (page 35).

For more information, see "Creating Products" (page 46).

### 4.1.2. Import the Example Component

The next stage is to add an example component, **ExampleShoppingCart**, to the product.

Bohemia Interactive Simulations provides this example component in a remote repository.

**Follow these steps:**

1. From the "Product Page" (page 35), click **Development** in the Product Toolbar.

   The Development Directory opens in a File Explorer window.

2. Use your preferred Source Control Client to clone the remote repository for the ExampleShoppingCart component from:

   https://gitlab.com/bisimulations/ExampleShoppingCart

   The local repository copy should exist in the root of the Development Directory.

3. Use your preferred Source Control Client to switch to the **1.0.0** tag for ExampleShoppingCart.

   This avoids using later versions that may be created by other developers following this example.

4. In the Product View, click the **Add Icon**  and select **Import**.

   The Import Component panel opens.

5. Select the synchronized **ExampleShoppingCart** component from the drop-down, and click **Done**.

Gears Studio adds the ExampleShoppingCart component and refreshes the Product View.

For more information see, "Importing Components" (page 52).

### 4.1.3. Run the Example Product

Use Gears Studio to locate the product executable and run it.

**Follow these steps:**

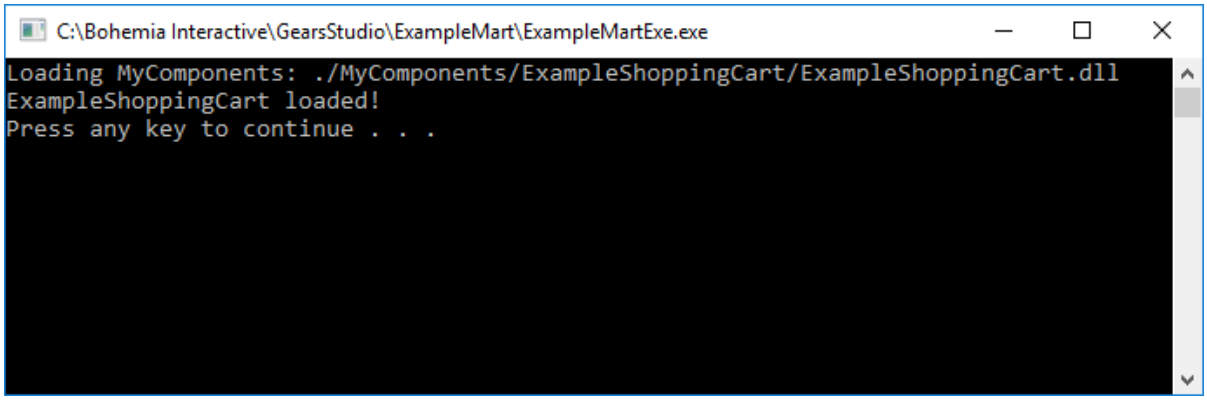
1. From the "Product Page" (page 35), click **Application** in the Product Toolbar .

   The Application Directory opens in a File Explorer window.

   Open the `MyComponents` folder to verify that the ExampleShoppingCart component is present.

2. Run `ExampleMartExe.exe`.

A console window opens with text stating that the ExampleShoppingCart is loaded.




There are no other components to interact with, so the product does nothing else at this point.

Press any key to close the console window.

## 4.2. Set Up the Example Component

Create a new component to expand the functionality of the example product.

**Follow this process:**

1. "Access the Create Component Window" (page 19)
2. "Create the Example Component" (page 20)
3. **Optional:** "Create an Example API" (page 22)
4. "Generate the Example Component" (page 23)

Complete this process and then continue to "Using the Example Code Solution" (page 24) to work with your code.

For more information about creating components and APIs, see "Creating Components" (page 49) and "Creating APIs" (page 58).

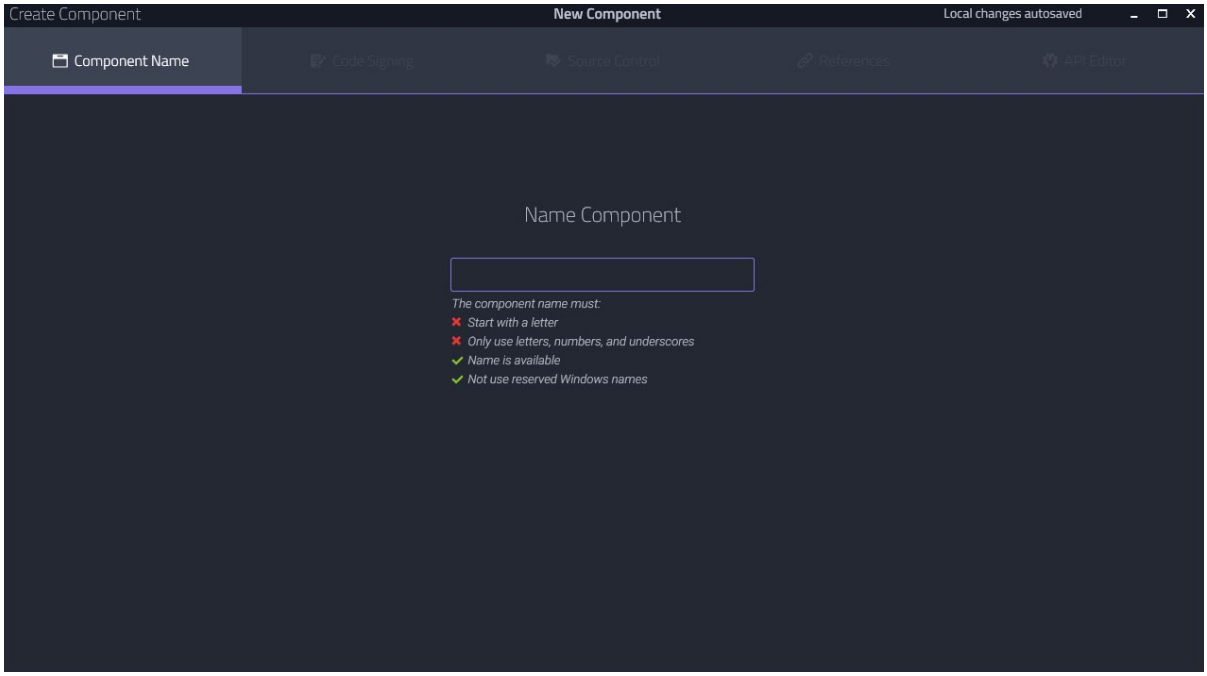
### 4.2.1. Access the Create Component Window

The initial step opens the "Component Window" (page 41) in Create Component mode.

**Follow these steps:**

1. Click the **Add** icon and select **Create**.

   The "Component Window" (page 41) opens in Create Component mode.

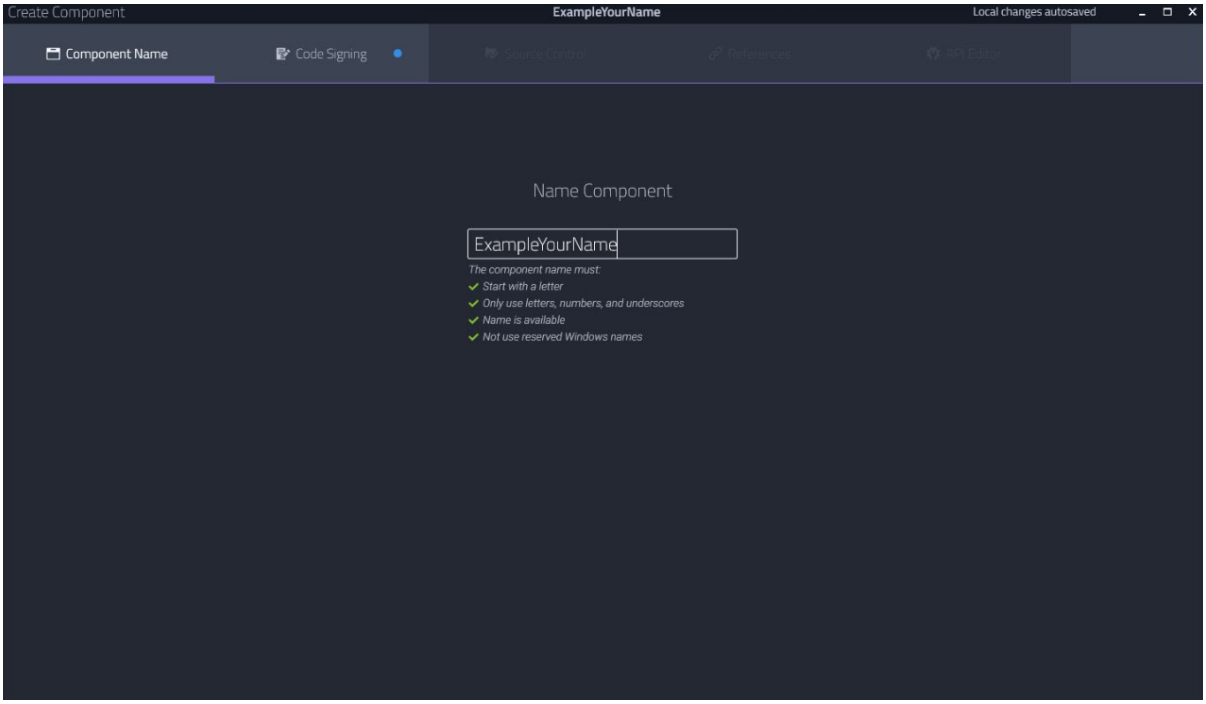



## 4.2.2. Create the Example Component

The "Component Window" (page 41) contains a set of tabs to guide the component creation process.

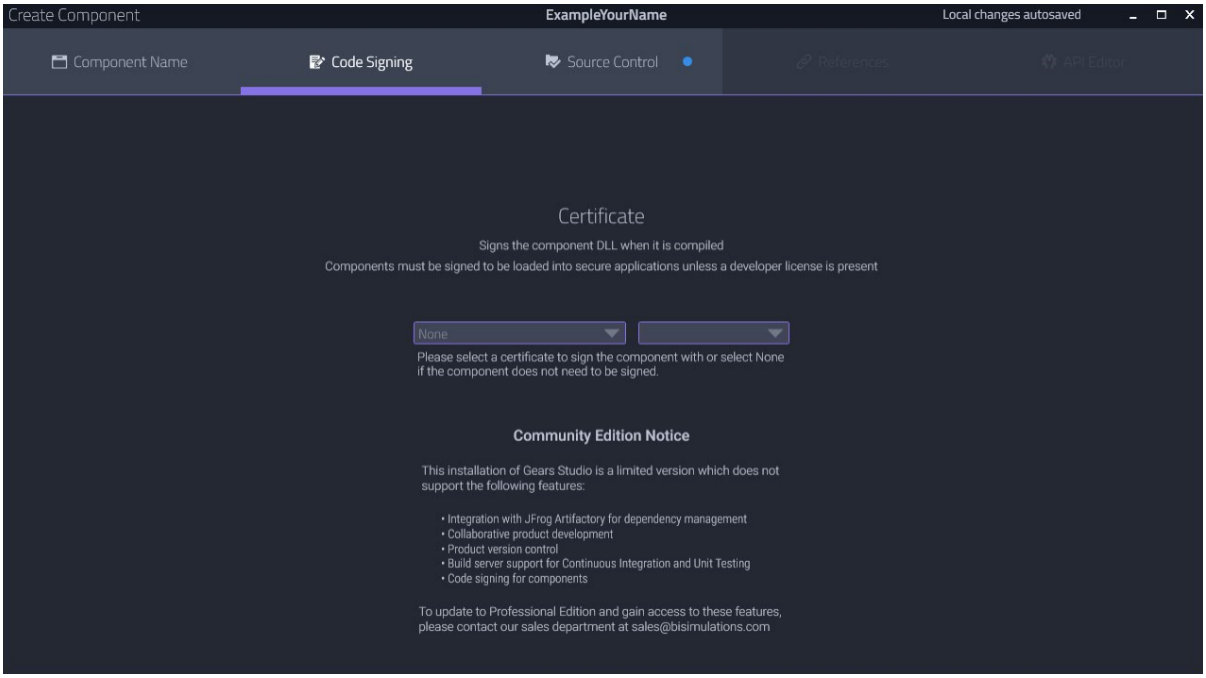
**Follow these steps:**

1. In the **Component Name** tab, input a unique name.



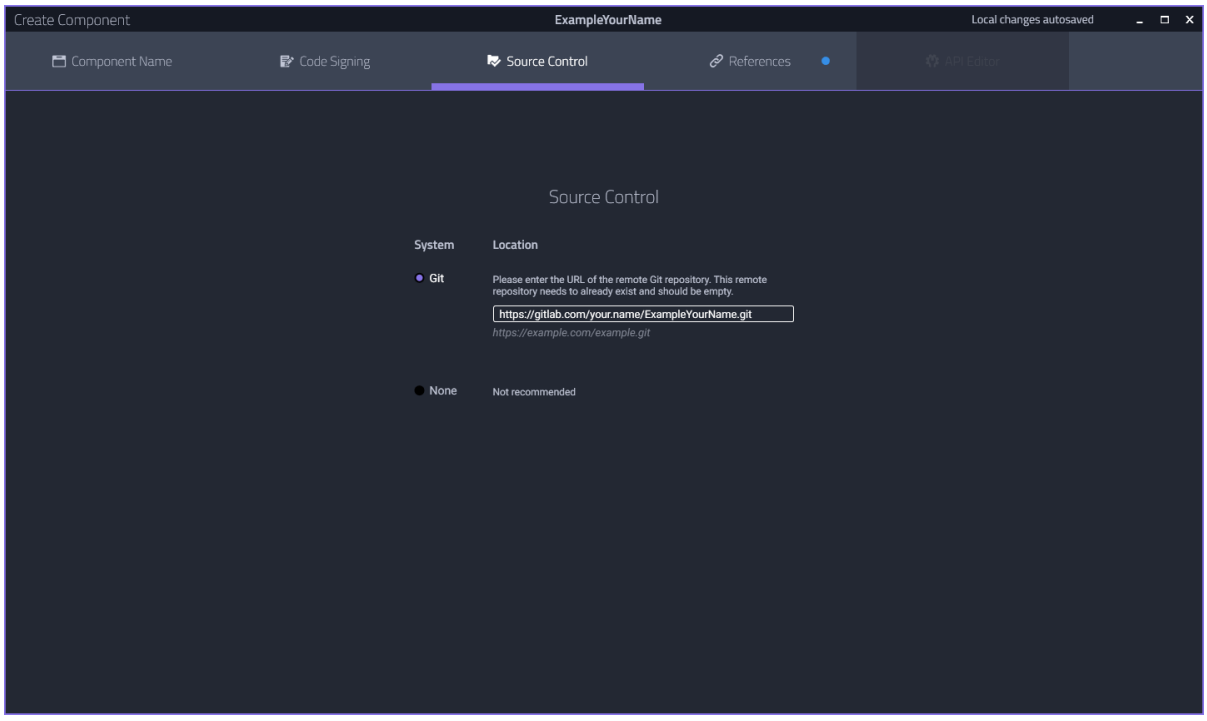

   For the purpose of this example, use your name to ensure that the component does not conflict with a teammate; **Example*YourName***.

2. Select the **Code Signing** tab. All options on this tab are disabled in community mode, because code signing is not supported.




3. Select the **Source Control** tab to specify your component source control.
    1. Select the **System: Git**
    2. Input the **Location** using the URL of the remote Git repository.

    **Note:** The new remote repository should already exist and be empty. For more information, see "Working with Source Control" (page 55).

4. Select the **References** tab to manage the component references.

    References enable you to manage component references. It displays the following:

- **Implemented APIs**

  Implemented APIs are APIs that are exposed for other components to use. By Implementing an API you are stating and advertising that this component implements a specific feature set that is defined by the API functions. When you choose to Implement an API all of the functions are initially empty and ready to populate with the logic to make them work.

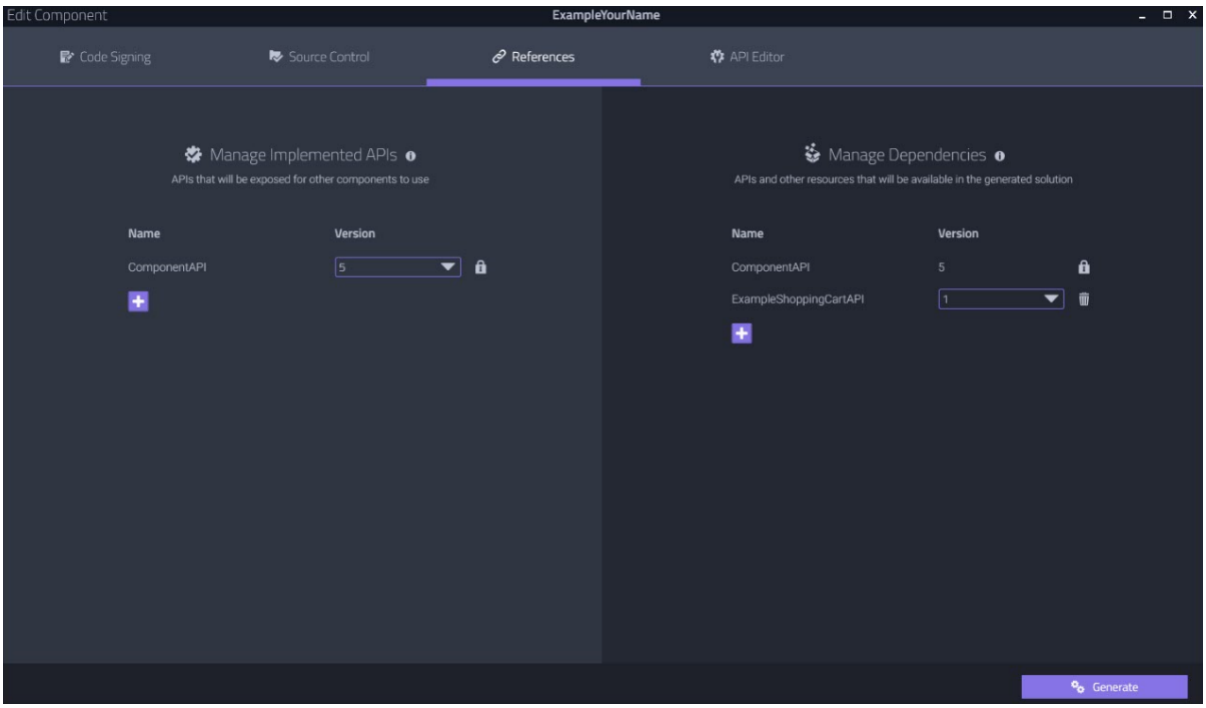
  These API functions allow other components to interact with your component to request data, perform operations, configure options, and more.

- **Dependencies**

  Dependencies are APIs or other resources that are available in the generated solution. Implemented APIs are automatically added to this list and cannot be removed.

  Dependencies are any resources that the component wants to interact with. Examples include APIs other components create, 3rd party libraries, and command line tools.

  Adding a dependency does not change your code at all. It simply makes the resource available within the code solution.




5. For the purpose of this example, no additional Implemented APIs are required.

   **Optional:** In the Manage Implemented APIs list, click the **Add Icon** . Expand the **Select** drop-down and select the **WorldListenerAPI**. This API provides notifications about world events such as object creation and deletion. This helps to demonstrate what Implementing an API does for your component, but is not actually be used in this example.

6. In the Manage Dependencies list, click the **Add Icon** .

   Expand the **Select** drop-down and select the **ExampleShoppingCartAPI** to enable interaction with the ExampleShoppingCart component.

### 4.2.3. Create an Example API

For the purpose of this tutorial, no other components interact with your API, but you can optionally create one to get a sense of how it works.

**Follow these steps:**

1. Select the "API Editor" (page 42) to define new feature sets for components to interact with.

   The API Editor enables you to create new APIs, defining new feature sets for components to Implement.

2. In the Created APIs panel, under **Create API:** click the **Add Icon** .

3. Input a unique name that matches the criteria, and click **Create**.

   **Note:** Gears Studio automatically adds API to the end of the name.

   Gears Studio creates a blank API template at Version 1 and opens the "API Code Tab" (page 43) and "API Details" (page 44).

4. In the "API Code Tab" (page 43), click the **Add Icon** to expand the API Menu, and click the **Function Button**.

   Gears Studio adds function template code to the API.

5. Populate the highlighted text with appropriate inputs for the return type, name, parameters, and comments.

For more information, see "Editing API Code" (page 59).

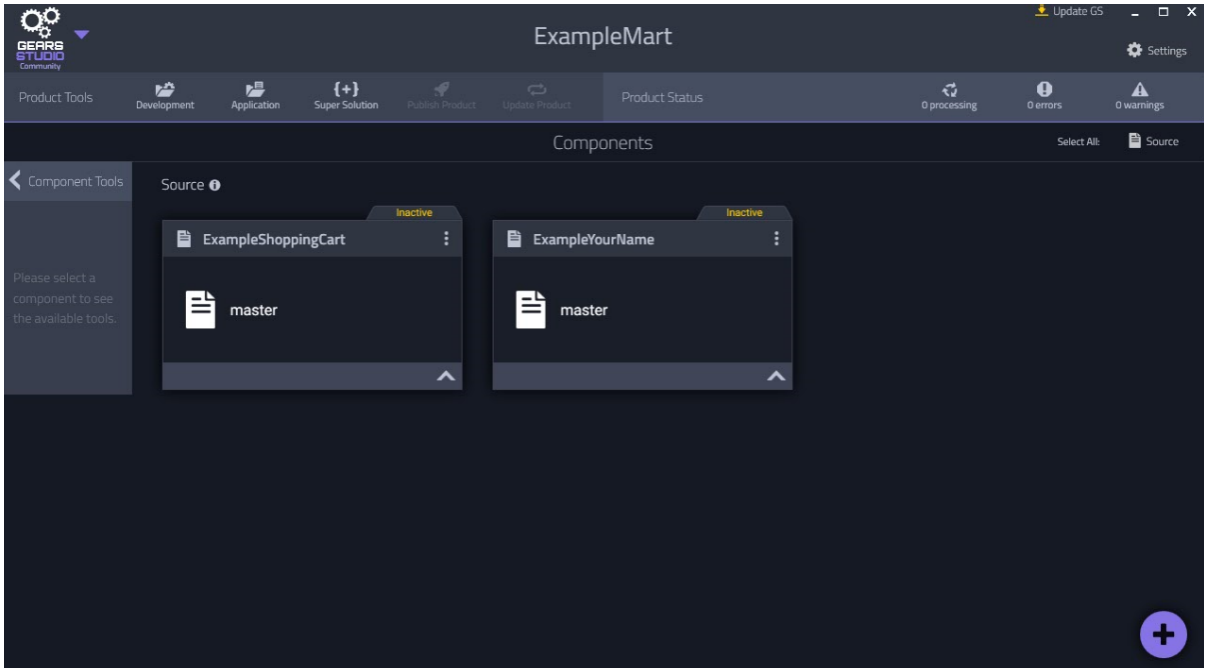
### 4.2.4. Generate the Example Component

A correctly configured component displays a Generate Button in every panel in the "Component Window" (page 41).


Click the **Generate Button**.

Gears Studio generates your component, creating all the necessary files and folders for the Code Solution. The component generates in an **Inactive** state, meaning that no binaries have been built yet. For more information, see "Component States" (page 10).

The Component Window closes and Gears Studio displays the "Product Page" (page 35) showing your new component.




Select the new component and click **Explorer** in "Component Tools" (page 39) to open a File Explorer window in the component directory.

For more information about the files and folders Gears Studio creates, see "Development Files and Folders" (page 11).

## 4.3. Using the Example Code Solution

The generated component creates a code solution for your preferred Code Editor.

Access the example component code from the "Product Page" (page 35).

**Follow these steps:**

1. Select the **Example*YourName*** component card.

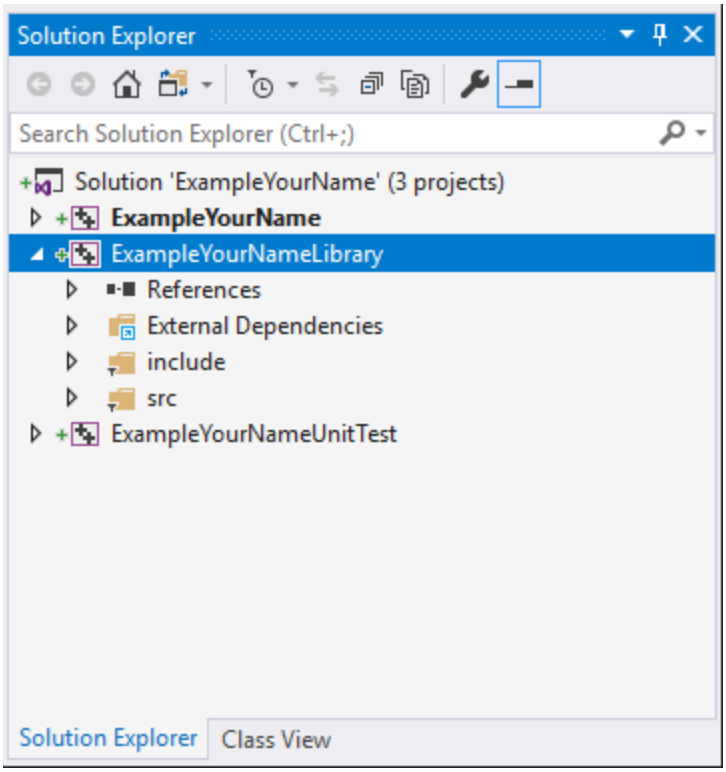
   For more information, see "Component Card" (page 38).

2. In the Component Tools panel, select **Code Editor**.

   For more information, see "Component Tools" (page 39).

   Gears Studio opens the component in your preferred Code Editor, selected during "Gears Studio Setup" (page 14) or in the "Developer Tools Settings" (page 65).

   The Solution Explorer displays the following:




3. Use your Code Editor to build the solution.

   Your Code Editor deploys the built component to the Application Directory. This enables the component to load when the product runs.

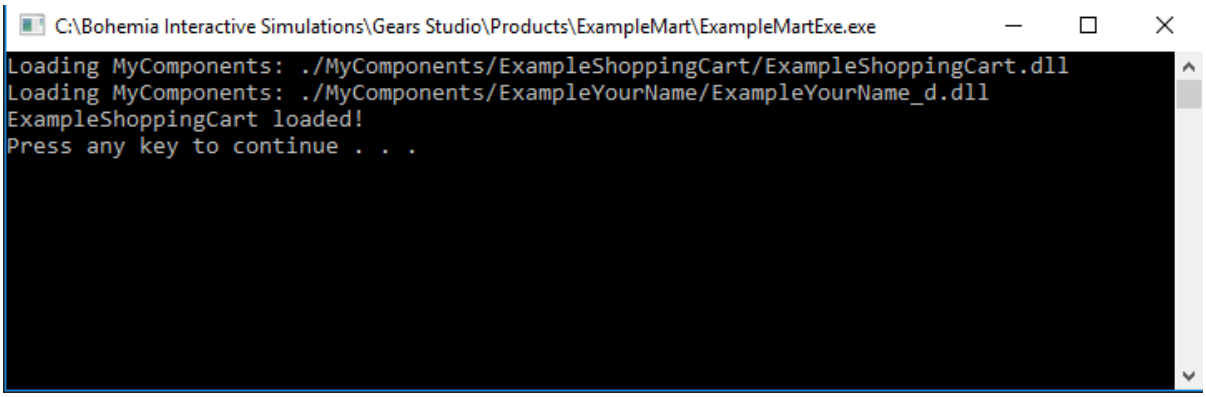
   In the "Product Page" (page 35), the component card removes the **Inactive** indicator.

4. Use your Code Editor to set the Debugging properties for the product:

   - Set the **Command** to the `ExampleMartExe.exe` in the Application Directory.
   - The **Working Directory** defaults to the Application Directory for ExampleMart.

5. Run the product.

   The console window opens, showing that your new component is loaded.

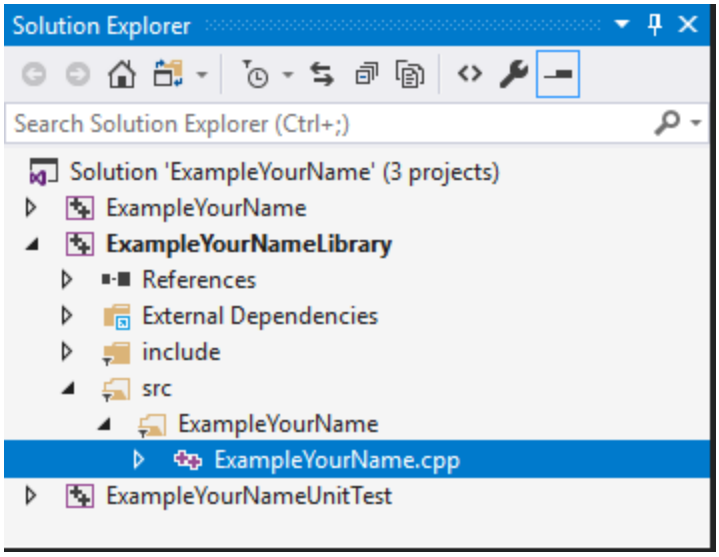



   The component does not do anything because there is no code written yet.

6. Use your Code Editor to add breakpoints and verify that the component is loading and explore the code stubs.
7. Update your component to interact with the ExampleShoppingCart component:

1. Locate and open `Example`*`YourName`*`.cpp` in your Code Editor.




2. Add the includes and create a global variable to store the API.

```
#include <iostream>
#include "ExampleShoppingCartAPI.h"
ExampleShoppingCartAPI_v1* g_shopping_cart_api = nullptr;
```

**Note:** <iostream> is required for logging purposes.

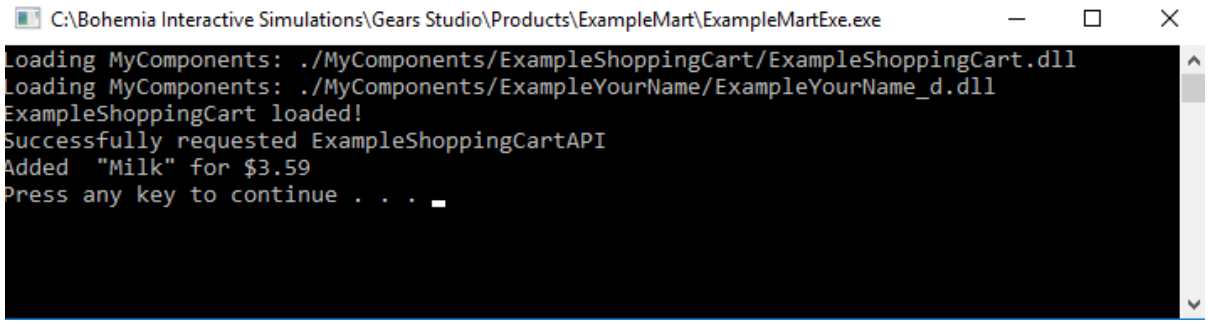
3. Locate the `Component_OnStart` function and replace it with the following code:

```
APIResult GEARS_API Component_OnStart(_In_ APIManager_v5* api_manager, _In_ NativeModuleHandle
proxy_handle)
{
    //GEARS NOTE: Use this function to connect to other components.
    //Using the specified api_manager variable, this component can request access
    //to APIs that have been registered with Gears.
    auto result = api_manager->RequestAPI(ExampleShoppingCartAPI_Handle, 1, (void**)&g_shopping_
cart_api);
    if (APIRESULT_FAIL(result))
    {
        std::cout << "Failed to request ExampleShoppingCartAPI\n";
        return kAPIResult_GeneralError;
    }
    std::cout << "Successfully requested ExampleShoppingCartAPI\n";
    // Add an item to the shopping cart using the ExampleShoppingCart API
    g_shopping_cart_api->AddItem("Milk", 3.59f);
    return kAPIResult_GeneralSuccess;
}
```

Note the insertion of an `AddItem` function to add an item to the shopping cart after a successful request to the ExampleShoppingCart API.

4. Build the Code Solution.

8. Run the product again to see the results.




The product adds the item specified and the shopping cart is no longer empty.

**Optional:** Experiment with the Code Solution to add multiple items to the cart.

Continue to "Editing the Example Component" (page 27) to add additional functionality.

## 4.4. Editing the Example Component

"Set Up the Example Component" (page 19) added basic component interaction for adding items to the shopping cart. To expand the existing functionality to support totaling and clearing the shopping cart, edit the **ExampleShoppingCart** component.

**Follow this process:**

1. "Edit the Example API" (page 27)
2. "Update the Example API Dependencies" (page 29)

Complete this process and then continue to "Using the Example Super Solution" (page 30) to work with the product code.

### 4.4.1. Edit the Example API

Modify the example component API and create new functions for totaling and clearing the shopping cart.

**Follow these steps:**

1. From the "Product Page" (page 35), select the **ExampleShoppingCart** component card.

   For more information, see "Component Card" (page 38).

2. In the Component Tools panel, select **Edit**.

   For more information, see "Component Tools" (page 39).

   Gears Studio opens the "Component Window" (page 41).

3. In the CreatedAPIs list, select **ExampleShoppingCartAPI**.

   Gears Studio displays the "API Code Tab" (page 43) and "API Details" (page 44).

   The API displays Version 1.

   **Note:** Editing an existing version can create issues for other users who are already using this version.

   For more information, see "API Versioning" (page 11).

4. Create a new major version of the API for editing.

   In the API Details panel, select the **Version** tab, and click **Major Version**.

   Gears Studio creates a Version 2 of the API.

5. Add a function to calculate the total for the shopping cart:

1. In the API Editor Panel, click the **Add Icon** to view the API Editor options.

2. Click the **Add Function Icon**.


Gears Studio adds a function template to the API code.

```
/* function name
 * @brief function description
 */
return type function name(parameter);
Error: expected identifier or '('
```

3. Add the following to the highlighted code fragments:

| Code Fragment | Input |
|---|---|
| `function description` | Calculate the shopping cart total |
| `return type` | APIResult |
| `function name` | GetTotalPrice |
| `parameter` | _Out_ float32_t* total_price |
| `parameter description` | Out parameter which gets set to the total price of the cart. |
| `result description` | The result of the function. |

```
/* GetTotalPrice
 * @brief Calculate the shopping cart total.
 * @param _Out_ float32_t* total_price Out parameter which gets set to the total price of the cart.
 * @return APIResult The result of the function.
 */
APIResult GetTotalPrice(_Out_ float32_t* total_price);
```

6. Add a function to clear the shopping cart:

1. In the API Editor Panel, click the **Add Icon** to view the API Editor options.

2. Click the **Add Function Icon**.


Gears Studio adds a function template to the API code.

3. Input the highlighted code fragments:

| Code Fragment | Input |
|---|---|
| `function description` | Clear the shopping cart |
| `return type` | APIResult |
| `function name` | ClearItems |
| `parameter` | Leave blank |
| `result description` | The result of the function. |

```
/* ClearItems
 * @brief Clear the shopping cart.
 * @return APIResult The result of the function.
 */
APIResult ClearItems();
```

7. Click **Generate**.


Gears Studio generates the code stubs and saves the new version in your Development Directory. You can close the Component Window.

### 4.4.2. Update the Example API Dependencies

Update your example component to use the new version of the ExampleShoppingCart API.

**Follow these steps:**

1. From the "Product Page" (page 35), select the **Example*YourName*** component card.

   For more information, see "Component Card" (page 38).

2. In the Component Tools panel, select **Edit**.

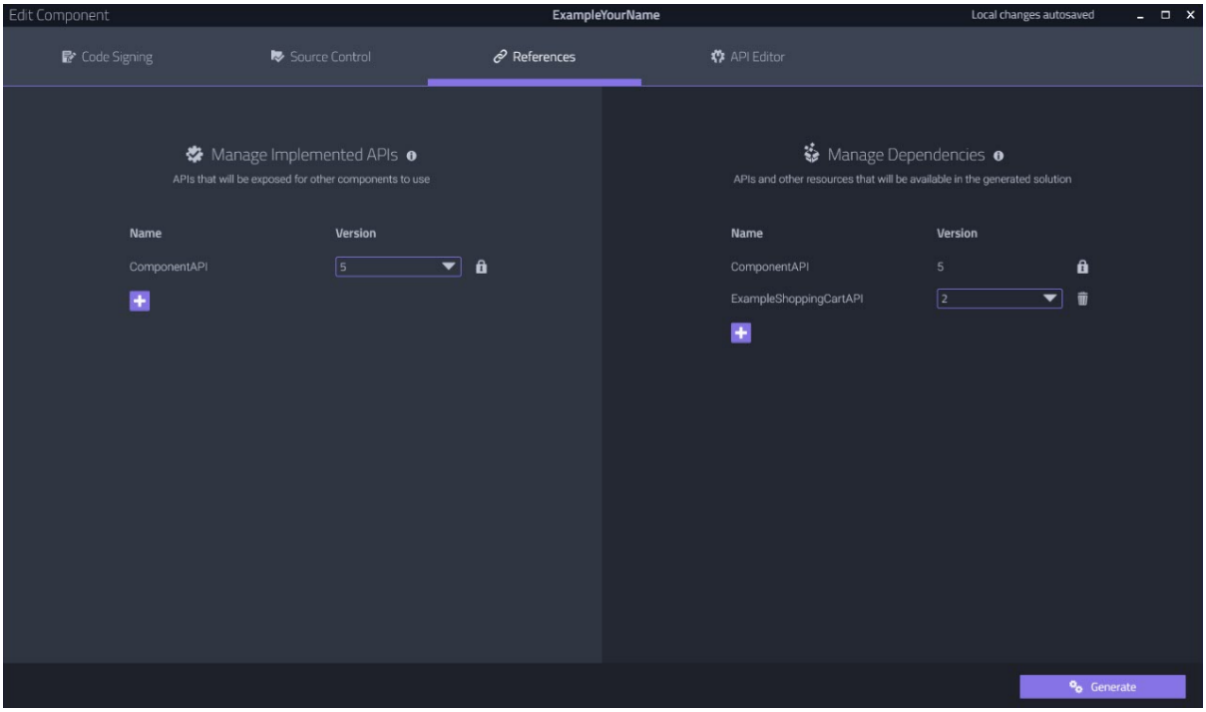
   For more information, see "Component Tools" (page 39).

   The component opens in the "Component Window" (page 41).

3. Select **References**.

4. Under **Manage Dependencies**, expand the **Version** drop-down for ExampleShoppingCartAPI, and select version **2**.

5. Click **Generate**.




Gears Studio regenerates the Code Solution with your changes and updates the "Product Page" (page 35).

## 4.5. Using the Example Super Solution

After making changes in Gears Studio to two different components, our new component and the ExampleShoppingCart component, update the code for both of them to implement the new functions for clearing and totaling the shopping cart.

Gears Studio makes working with multiple components easier by combining all the individual component Code Solutions into a single **Super Solution**.

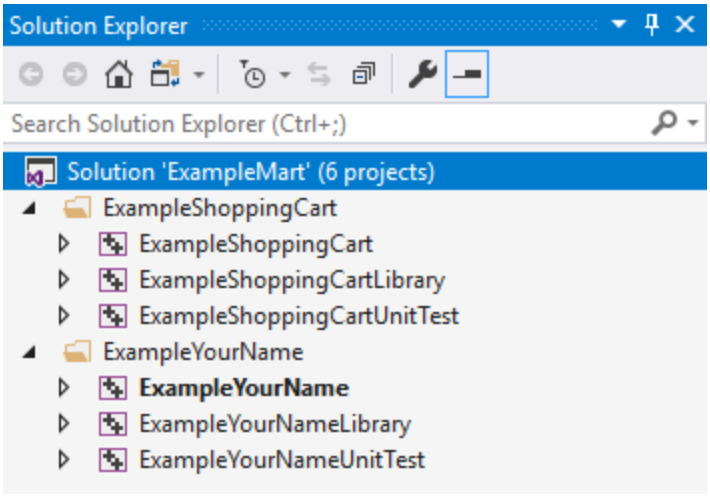
**Follow these steps:**

1. From the "Product Page" (page 35), select **Super Solution** from the Product Tools.

   For more information, see "Product Tools" (page 36).

   Gears Studio opens the product in your preferred Code Editor, selected during "Gears Studio Setup" (page 14) or in the "Developer Tools Settings" (page 65).

   The Solution Explorer displays the following:




2. Edit the **ExampleShoppingCartLibrary** project to implement the new functions:
    1. Open `ExampleShoppingCart.cpp` in your Code Editor.
    2. Locate the template clearing and total functions and replace them with the following code:

```
APIResult GEARS_API ExampleShoppingCart_GetTotalPrice(_Out_ float32_t* total_price)
{
    // Get the total price of all items in the shopping cart
    *total_price = g_shopping_cart.GetTotalPrice();
    return kAPIResult_GeneralSuccess;
}
APIResult GEARS_API ExampleShoppingCart_ClearItems()
{
    // Clear all the items from the shopping cart
    g_shopping_cart.ClearItems();
    return kAPIResult_GeneralSuccess;
}
```

    3. Open `ShoppingCart.h` in your Code Editor.
    4. Declare the new functions with the following code:

```
// TODO :: Declare 'Clear' and 'GetTotalPrice' functions here
void ClearItems();
float GetTotalPrice() const;
```

    5. Open `ShoppingCart.cpp` in your Code Editor.
    6. Define the new functions with the following code:

```
// TODO :: Define 'Clear' and 'GetTotalPrice' functions here
void ShoppingCart::ClearItems()
{
    _items.clear();
    std::cout << "Items cleared\n";
}
float ShoppingCart::GetTotalPrice() const
{
    float total_price = 0.0f;
    for (auto iter = _items.begin(); iter != _items.end(); ++iter)
    {
        total_price += iter->price;
    }
    return total_price;
}
```

3. Edit the **Example*YourName*Library** project to use the new functions:

1. Open `Example`*`YourName`*`.cpp` in your Code Editor.
2. Update to use version 2 of the ExampleShoppingCartAPI:

   ```
   ExampleShoppingCartAPI_v2* g_shopping_cart_api = nullptr;
   ```

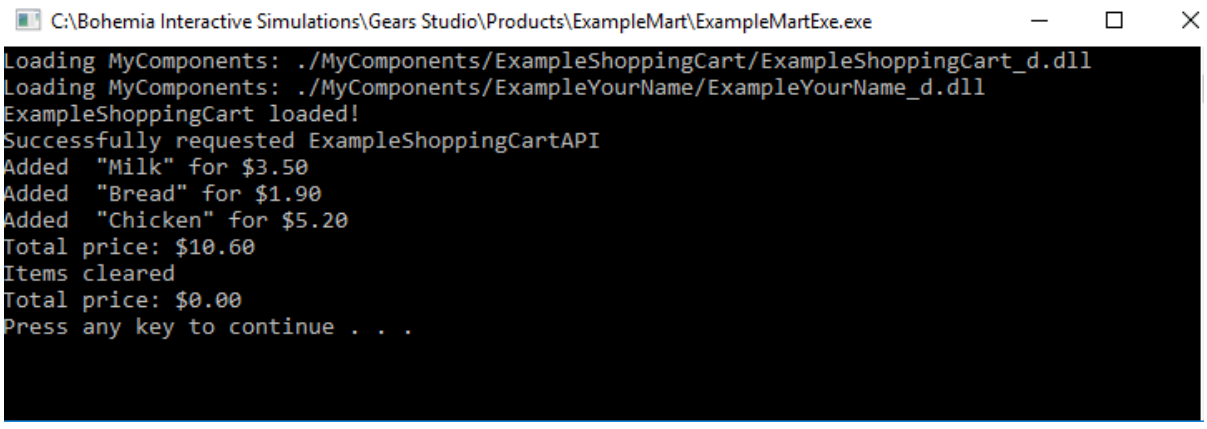
   and:

   ```
   auto result = api_manager->RequestAPI(ExampleShoppingCartAPI_Handle, 2, (void**)&g_shopping_cart_
   api);
   ```

3. Update the `Component_OnStart` function to demonstrate the new API functions with the following code:

   ```
   std::cout << "Successfully requested ExampleShoppingCartAPI\n";

   // Add items to the shopping cart using the ExampleShoppingCart API
   g_shopping_cart_api->AddItem("Milk", 3.50f);
   g_shopping_cart_api->AddItem("Bread", 1.90f);
   g_shopping_cart_api->AddItem("Chicken", 5.20f);

   // Get the total price
   float32_t total_price = 0.0f;
   g_shopping_cart_api->GetTotalPrice(&total_price);
   std::cout << "Total price: $" << total_price << '\n';

   // Clear all the items
   g_shopping_cart_api->ClearItems();

   // Get the total price again
   g_shopping_cart_api->GetTotalPrice(&total_price);
   std::cout << "Total price: $" << total_price << '\n';

   return kAPIResult_GeneralSuccess;
   ```

4. Save the modified files.
5. Use your Code Editor to set the Debugging properties for the product:
   - Set the **Command** to the `ExampleMartExe.exe` in the Application Directory.
   - The **Working Directory** defaults to the Application Directory for ExampleMart.
6. Build the Solution in your Code Editor.
7. Run the Solution to demonstrate the modified functionality.




The product adds the item specified, calculates the total, and then clears the shopping cart.

**Optional:** Experiment with the Super Solution to make further use of the new functions.

This concludes the Gears Studio example.

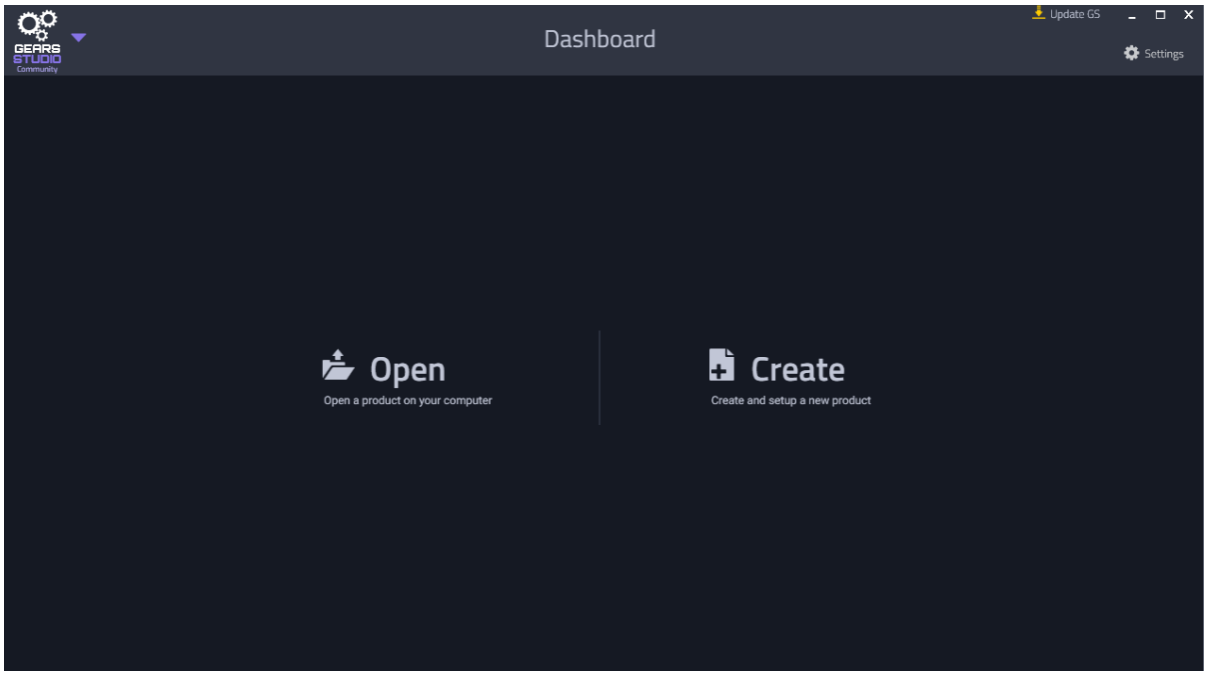
# 5. Gears Studio UI Overview

The Gears Studio User Interface is designed to provide quick access to everything you need to develop your Gears Products, Components, and APIs.

The UI consists of the following main elements:

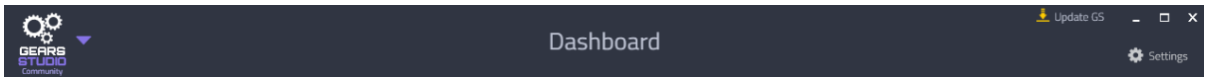
- "Gears Studio Header" (page 33)
- "Dashboard" (page 35)
- "Product Page" (page 35)
- "Component Window" (page 41)
- "API Editor" (page 42)

After the initial "Gears Studio Setup" (page 14), opening Gears Studio displays the "Dashboard" (page 35).




## 5.1. Gears Studio Header

The Header provides access to Gears Studio information, updates, and settings.

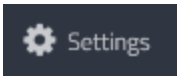



The Header displays in the "Dashboard" (page 35) and the "Product Page" (page 35) and the title displays Dashboard or the name of the current Product.

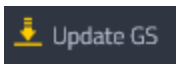
The Header contains the following elements:

- "Gears Studio Menu" (page 34)

- "Gears Studio Settings" (page 34)



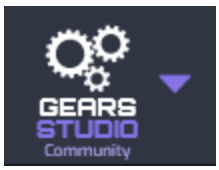

If an update to Gears Studio is available, the **Update Indicator** displays.



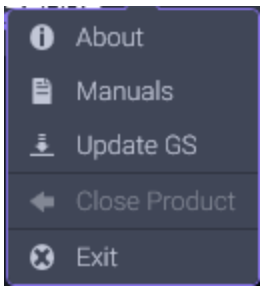

Click the Indicator to open the Update Version panel and "Update Gears Studio" (page 68).

### 5.1.1. Gears Studio Menu

The Menu provides access to Gears Studio information, updates, and some navigation.




Click the **Menu** icon to view the Menu.

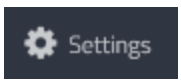



The Menu provides the following options:

- Click **About** to view Gears Studio version information.
- Click **Manuals** to access the Gears Studio documentation.
- Click **Update GS** to "Update Gears Studio" (page 68).
- Click **Close Product** to exit the Product View and return to the Dashboard.
- Click **Exit** to close Gears Studio.

### 5.1.2. Gears Studio Settings

Click the **Settings** icon to open the Settings Panel.




The Settings Panel provides access to the following settings:

- **General**

  Overall settings for Gears Studio.

- **Developer Tools**

  Specify your code editor and source control tools.

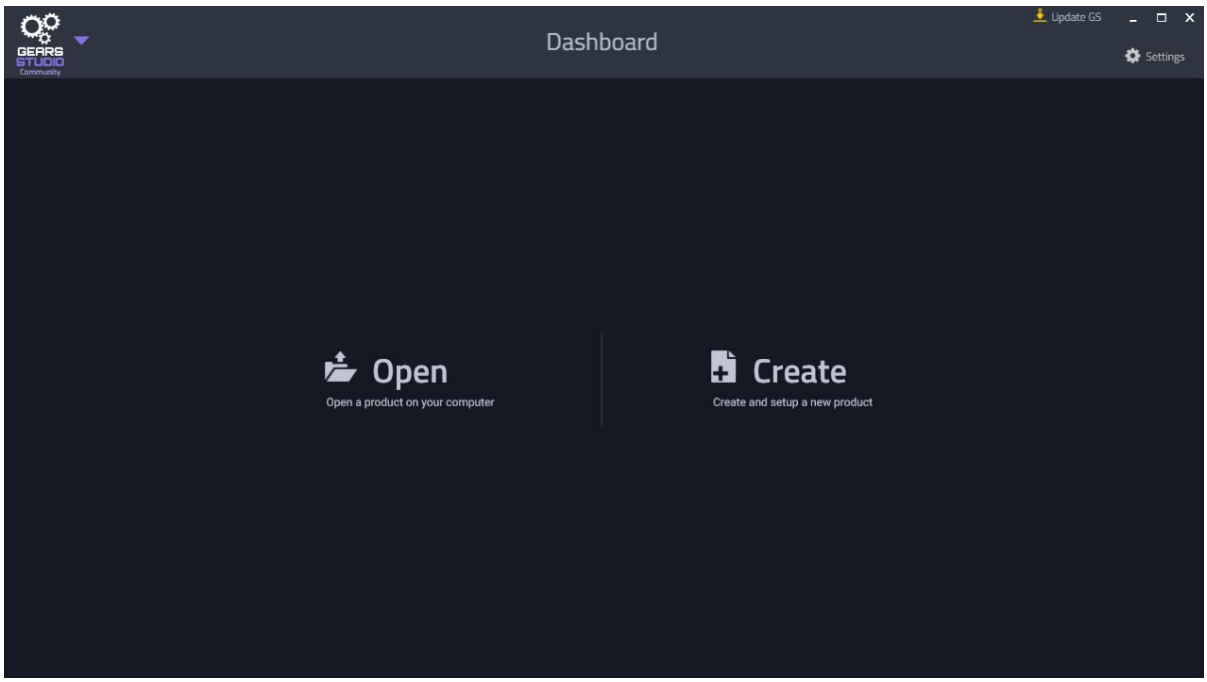
- **API Warnings**

  Specify which types of warnings to display in the API Editor.

For more information, see "Configure Gears Studio Settings" (page 64).

## 5.2. Dashboard

The Dashboard is the initial page that opens when you start Gears Studio, and provides access to your Products.




Use the Dashboard to perform the following actions:

- Use the "Gears Studio Header" (page 33) to access Gears Studio information, updates, and settings.
- Click **Open** to access or delete existing products.

  For more information, see "Opening Products" (page 47) and "Deleting Products" (page 47).
- Click **Create** to set up a new product.

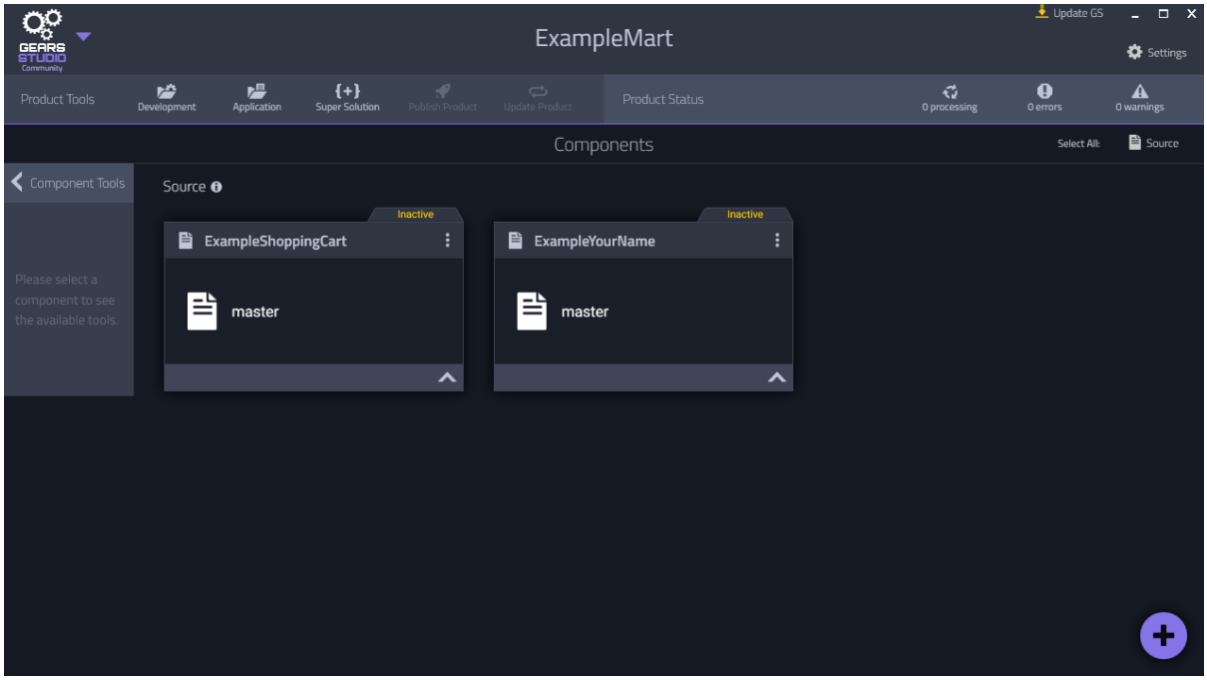
  For more information, see "Creating Products" (page 46).

All of these processes open the "Product Page" (page 35).

To return to the Dashboard from the Product Page, open the "Gears Studio Menu" (page 34) and select **Close Product**.

## 5.3. Product Page

Product Pages provide access to all the information and tools you need to work with your Gears Product.




Access the Product Page from the "Dashboard" (page 35), using any of the following processes:

- "Opening Products" (page 47)
- "Creating Products" (page 46)

To return to the Dashboard, open the "Gears Studio Menu" (page 34) and select **Close Product**.

Use Product Pages to perform the following actions:

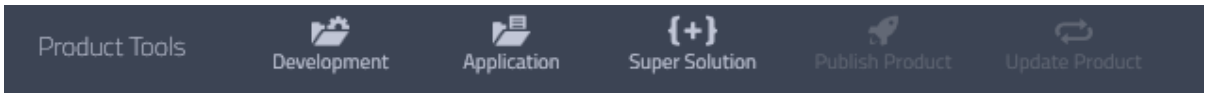
- Use the "Gears Studio Header" (page 33) to access Gears Studio information, updates, and settings.
- Use "Product Tools" (page 36) to perform specific actions on the product.
- Use "Product Status" (page 37) to review the overall status of the product.
- Use the "Components" (page 37) area to manage the components in the product.

For more information about working with Products, see "Developing Products" (page 46).

A set of "Keyboard Shortcuts" (page 40) are available for use in the Product Page.

### 5.3.1. Product Tools

Product Tools provides specific actions for the current product.




Click a Product Tools icon to perform the following actions:

- **Development**

  Open a File Explorer window for Product Development Directory location.

- **Application**

  Open a File Explorer window for Product Application Directory location.

- **Super Solution**

  Opens the solution in your Code Editor containing all the components in the current product.

- **Update Product**

  Opens the Update Product panel to update your product to a different version.

  **Note:** Update Product is not available in Gears Studio Community Edition.

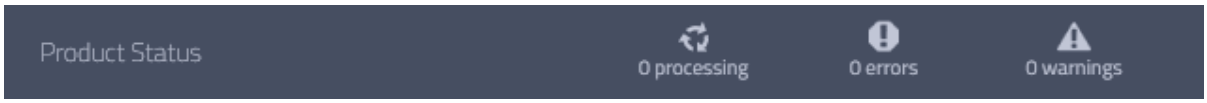
- **Publish Product**

  Opens the Publish Product panel to publish your changes to JFrog Artifactory.

  **Note:** Publish Product is not available in Gears Studio Community Edition.

### 5.3.2. Product Status

Product Status displays the current state of the product.




Product Status provides the following information:

- **Processing**

  Displays the current number of tasks being processed for the current product. Click the **Processing Icon** to view a list of the items currently processing in more detail.

- **Errors**

  Displays the current number of errors in the product. Click the **Errors Icon** to cycle through the errors and highlight the component that contains the error.

  For more information about the specific error, see the "Component Card" (page 38).

- **Warnings**

  Displays the current number of warnings in the product. Click the **Warnings Icon** to cycle through the warnings and highlight the component that contains the warning.

  For more information about the specific warning, see the "Component Card" (page 38).

### 5.3.3. Components

The Components area of the Product Page provides access to information and tools for the components in the product.

Use the Components area to manage the components in your product:

- Use a "Component Card" (page 38) to view information about the component and perform actions on it.
- Select Component Cards to enable component actions on them:
  - Click a card to select a single card.
  - Hold **Ctrl** and click additional cards to add them to the selection.
  - Hold **Shift** and click a card to add all cards between the initial selection and that card.
  - Click the **Source Icon** in the **Select All:** part of the Components area header to select all Source components.
  - To deselect cards, click the background.
- Use "Component Tools" (page 39) to perform actions on selected components.

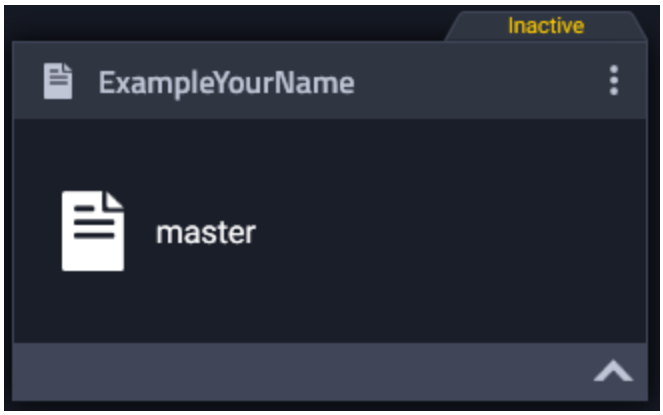
- Click the **Add Icon** , to create or import a component as part of your product.

  For more information, see "Creating Components" (page 49) and "Importing Components" (page 52).

For more information about working with Components, see "Developing Components" (page 49).

### Component Card

Component Cards provide visual representations of each component in the product.




Component Cards display the following information:

- The Card Tab indicates if the component is **Inactive**.

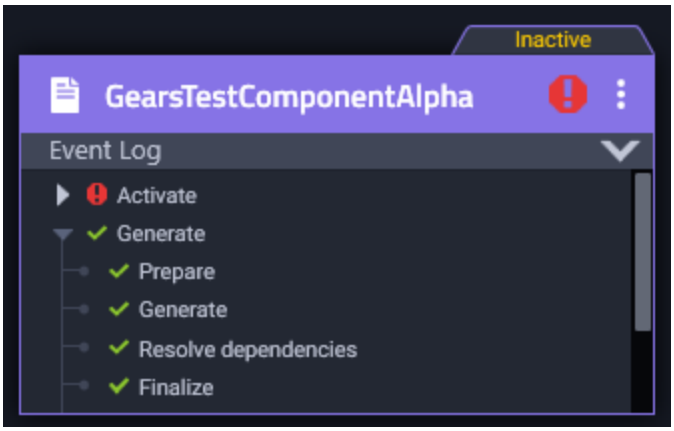
  For more information, see "Component States" (page 10).

- The Card Header displays the component name, error and warning icons, and the Component Menu.

  Selected components highlight the **Card Header**.

  Click the **Menu Icon** to open a Component Actions menu for that component. See "Component Tools" (page 39) for the menu options.

- The Card Body displays information about the component:
  - The source version.
- Click the Card Footer arrow icon or the Error or Warning icon to expand the Event Log and processes.

  The Event Log shows errors and warnings related to the component along with a log of its processes.

  Some items in the can be expanded to show more detail by clicking the arrow next to an item. If an item is fully expanded, it show as a grey dot. The status of each process is shown to the left as follows:

-  - The process is currently being executed.
-  - The process failed to complete.
-  - The process was skipped because it was not needed.
-  - The process completed successfully.

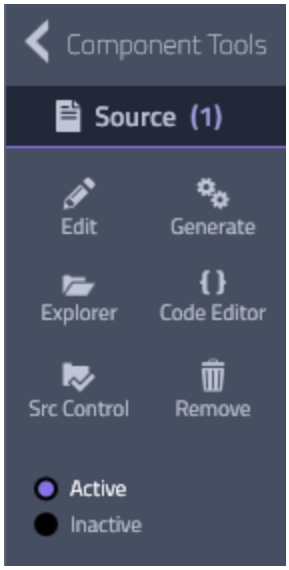
Some items in the status window can be hovered over for additional details.

- Double-click a Component Card to open its "Component Window" (page 41).
- Right-click a Component Card to view its menu. See "Component Tools" (page 39) for the menu options.

## Component Tools

Component Tools displays all the actions available for the currently selected components.

**Note:** To access the same actions for a single component, right-click a "Component Card" (page 38) or click its **Menu Icon**  .




Collapse and expand Component Tools with the **Arrow Icon**.

Use the Component Tools to perform the following actions:

- **Edit**

  Opens a "Component Window" (page 41) for each selected component, enabling you to add new APIs, edit existing APIs, update references, and perform other component actions.

  For more information, see "Editing Components" (page 52).

  After editing a component, regenerate the component in order to apply the changes to the component code.

- **Generate**

  Generates each selected component.

  For more information, see "Generating Components" (page 53).

- **Explorer**

Opens a File Explorer window for each selected component for the component folder in the Product Development Directory.

- **Code Editor**

  Opens each selected component in your Code Editor, selected during "Gears Studio Setup" (page 14) or in the "Developer Tools Settings" (page 65).

  For more information, see "Using Code Editors" (page 54).

  **Note:** To open a combined product solution in your Code Editor, use **Super Solution** in "Product Tools" (page 36).

- **Src Control**

  Opens each selected component in your Source Control Client, selected during "Gears Studio Setup" (page 14) or in the "Developer Tools Settings" (page 65).

  For more information, see "Working with Source Control" (page 55).

- **Remove**

  Removes each selected component from the product.

  For more information, see "Removing Components" (page 55).

  Removed components are only removed from the product and not deleted from the Product Development Directory. To restore a removed component to a product, see "Importing Components" (page 52).

- **Active / Inactive**

  Indicates whether the selected components are Active or Inactive. If multiple components are selected in different states, neither state is indicated.

  Click a state to set all selected components to that state.

  For more information, see "Component States" (page 10).

### 5.3.4. Keyboard Shortcuts

Use the following keyboard shortcuts to perform actions in the Product page.

| Shortcut | Action | Description |
|---|---|---|
| **Ctrl+A** | Select All: Source | Selects all Source mode components. |
| **Ctrl+W** | Edit | Opens an Edit Component window for all selected components. |
| **Ctrl+G** | Generate | Generates project files and source code for all selected components. |
| **Ctrl+E** | Explorer | Opens a File Explorer window for all selected components. |
| **Ctrl+R** | Code Editor | Opens a Code Editor window for all selected components.<br>The Code Editor is configured in "Developer Tools Settings" (page 65). |
| **Ctrl+T** | Source Control | Opens a Source Control window for all selected components.<br>The Source Control is configured in "Developer Tools Settings" (page 65). |
| **Delete** | Remove | Removes the selected components after user confirmation. |
| **Ctrl++** | Active | Sets all selected components to Active (if possible). |
| **Ctrl+-** | Inactive | Sets all selected components to Inactive. |

## 5.4. Component Window

Component Windows provide all the information and tools you require to develop your Gears Components.

Access Component Windows from the "Product Page" (page 35):

1. Select Components in the Product Page.
2. In the "Component Tools" (page 39) or "Component Card" (page 38) Menu, click **Edit**.

A separate Edit Components window opens for each selected component.

The Component Window contains the following tabs to enable the development of components:

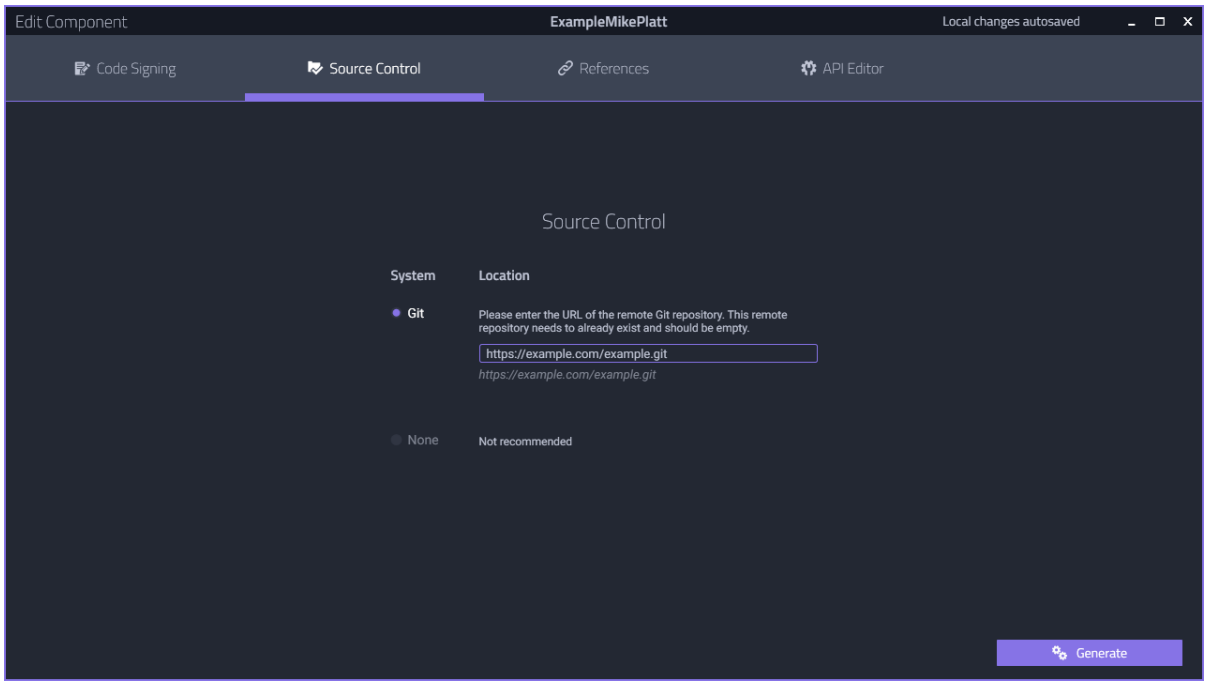
- "Source Control" (page 41)
- "References" (page 41)
- "API Editor" (page 42)

Click the **Generate Button** to start the component generation process and create component files. For more information, see "Generating Components" (page 53).


For more information about working with Components, see "Developing Components" (page 49).

### 5.4.1. Source Control

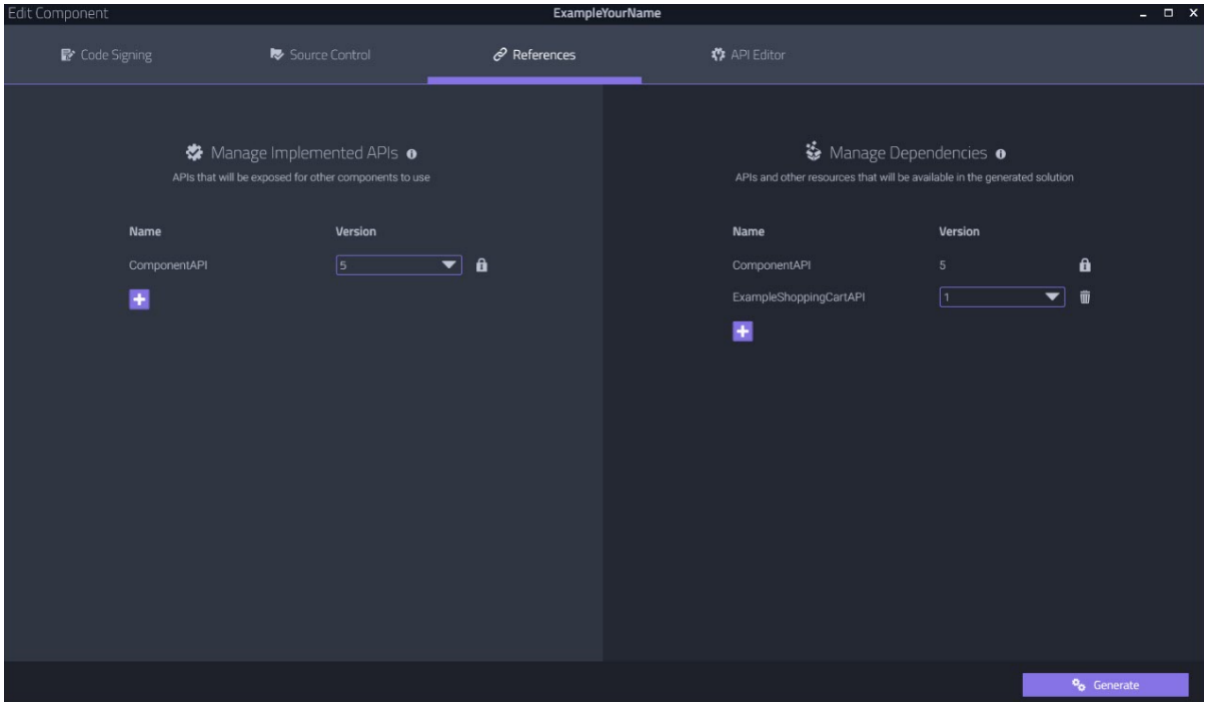
Use Source Control to set the remote Git repository for your component. Gears Studio uses this to import the source code from the remote repository to the Development Directory to enable component development.




For more information about source control, see "Working with Source Control" (page 55).

### 5.4.2. References

Use References to manage the API Implementation and Dependencies for the component.




The References tab divides the API references into the following parts:

- **Manage Implemented APIs**

  This area enables you to control the APIs that the component exposes for other APIs to use.

- **Manage Dependencies**

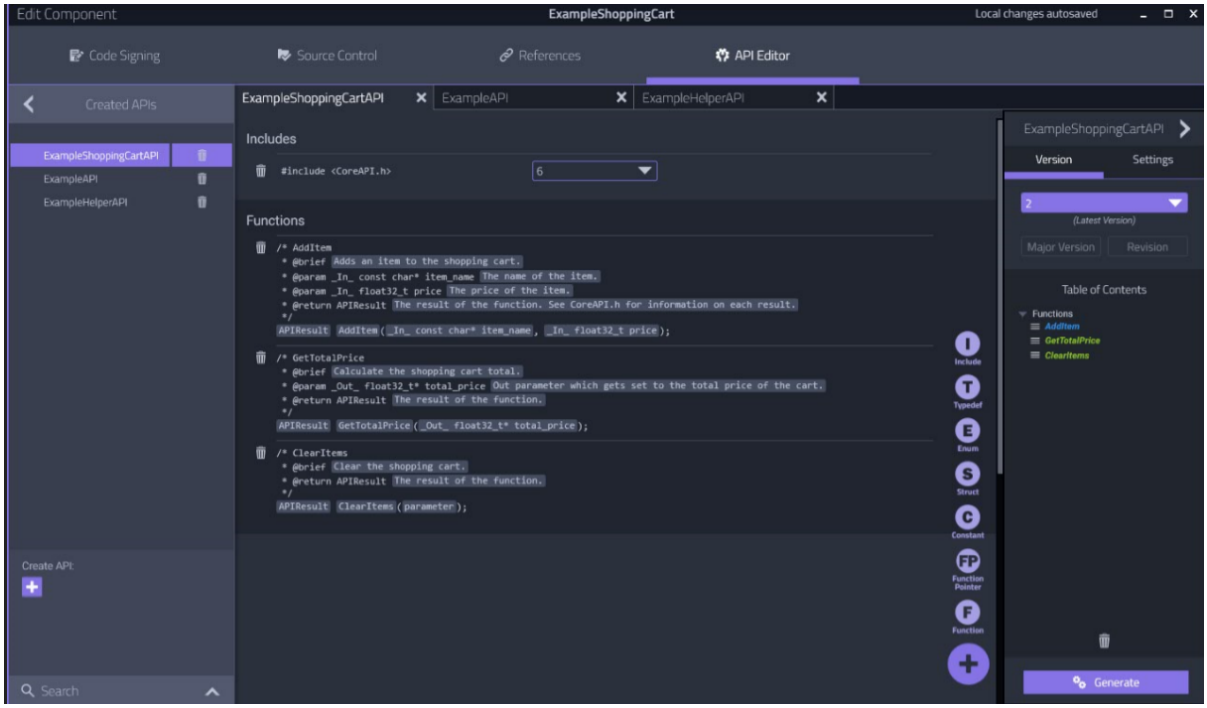
  This area displays the APIs and other resources that will be available in the generated solution.

For more information about managing references, see References in "Creating Components" (page 49).

## 5.5. API Editor

Use the API Editor to edit and create component APIs, defining new feature sets for components to implement.

The API Editor is a tab in the "Component Window" (page 41), accessed by "Creating Components" (page 49) or "Editing Components" (page 52).




The API Editor contains the following elements:

- Use the "Created APIs" (page 43) to manage the APIs created for the component.
- Use each "API Code Tab" (page 43) to edit APIs.
- Use each "API Details" (page 44) to control API versions and settings.

### 5.5.1. Created APIs

Created APIs displays the set of APIs created for the component. The panel also contains API create and search options.

Use the Created APIs Panel to perform the following actions:

- Click an API name in the panel to open its "API Code Tab" (page 43) and "API Details" (page 44).

  For more information, see "Opening APIs" (page 59).
- Click the **Add Icon** , to create a new API for the component.

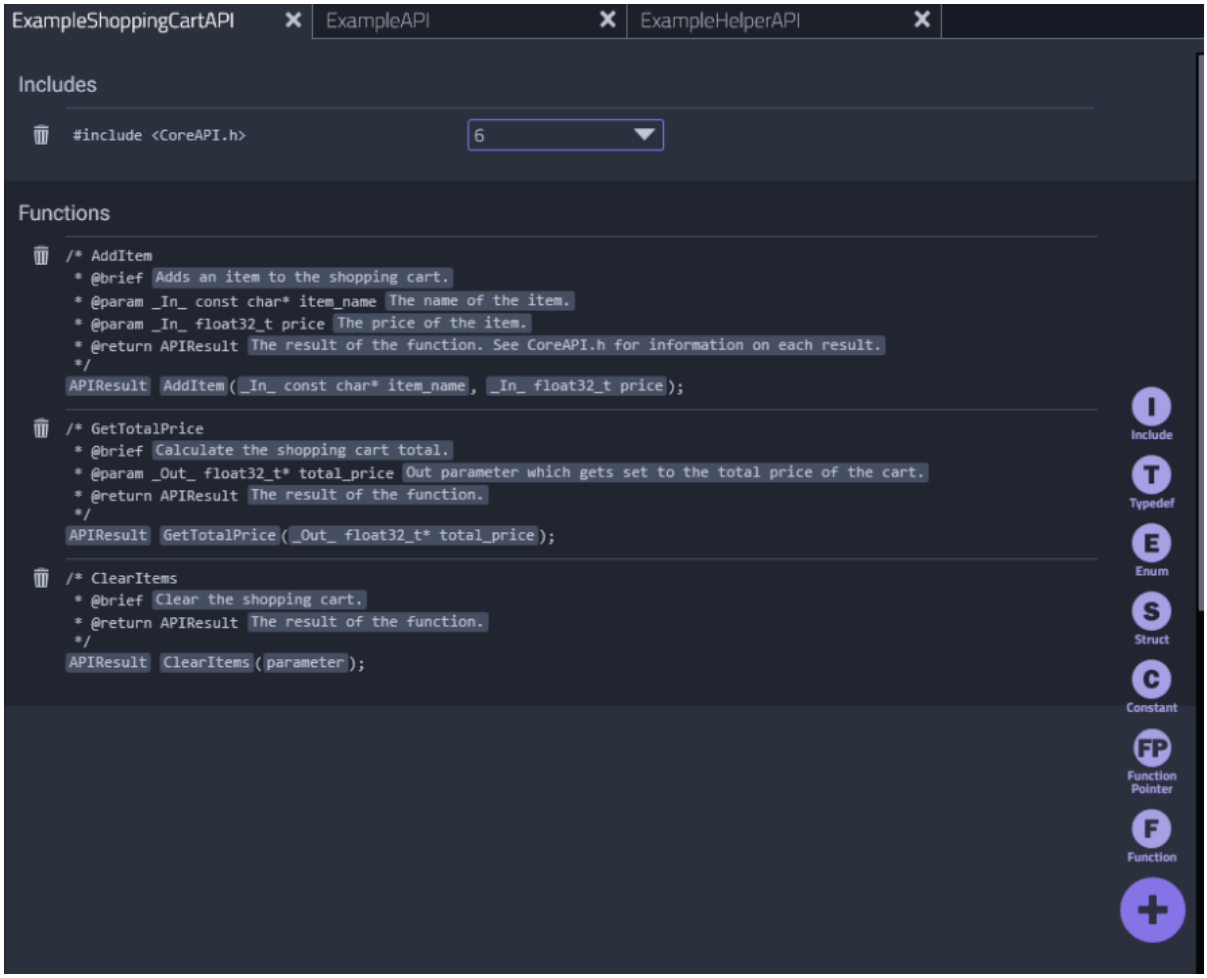
  For more information, see "Creating APIs" (page 58).
- Expand **Search** to perform searches within the component APIs.

  For more information, see "Searching APIs" (page 62).
- Click the **Trash Icon** in the Created APIs list, to remove the API from the component.

  For more information, see "Removing APIs" (page 63).

### 5.5.2. API Code Tab

"Creating APIs" (page 58) or selecting an API from the "Created APIs" (page 43) list displays the API Code in an editable tab.




An editable API displays an **Add Icon** .

Use the API Code Editor to do any of the following:

- Click the **Add Icon** to expand the API Menu and add API elements.
- Click the **Trash Icon** next to an API element to remove it from the API.
- Input the highlighted text fields to edit APIs.

For more information, see "Editing API Code" (page 59).

### 5.5.3. API Details

Use the API Details Panel to control API versions and settings, and to navigate the API content.

- Use the **Version** tab to view API versions and create new **Major Versions** and **Revisions**.

  Initially, the Version displays the current version for the selected API. It is marked *(Latest Version)*, if it is the latest version.

  Use the drop-down to change the API version to display. This does not make any changes to the API or component; it simply changes which version of the API to display. This is useful for viewing the history of an API.

  For more information, see "Versioning APIs" (page 61).

- The **Table of Contents** in the **Version** tab lists all the API Elements in a compact view.

The API Elements are grouped into their respective types (structs, constants, functions, and so on). If no elements exist for a type, that type does not appear in the list.

Use the arrows to expand and contract API types.

The API Elements are color-coded to indicate their state:

- **Unchanged API Element**
- **New API Element**
- **Updated API Element**
- ~~**Removed API Element**~~

Use the Table of Contents to rearrange and navigate to specific API elements in the "API Code Tab" (page 43).

You can also delete API Elements using the Table of Contents. Drag an API Element from the list to the **Trash Icon**  at the bottom of the Version tab and confirm the removal.

- Use the **Settings** tab to control API Instances.

  For more information, see "API Settings" (page 61).

# 6. Developing Products

Products are the way that Gears Studio organizes, groups, and manages components that are related.

Before starting to develop products, review the following concepts:

- "Product Directories" (page 9)

Gears Studio provides the following options to start working with a product from the "Dashboard" (page 35):

- "Creating Products" (page 46)
- "Opening Products" (page 47)

All these options open the product in the "Product Page" (page 35) to enable your product development:

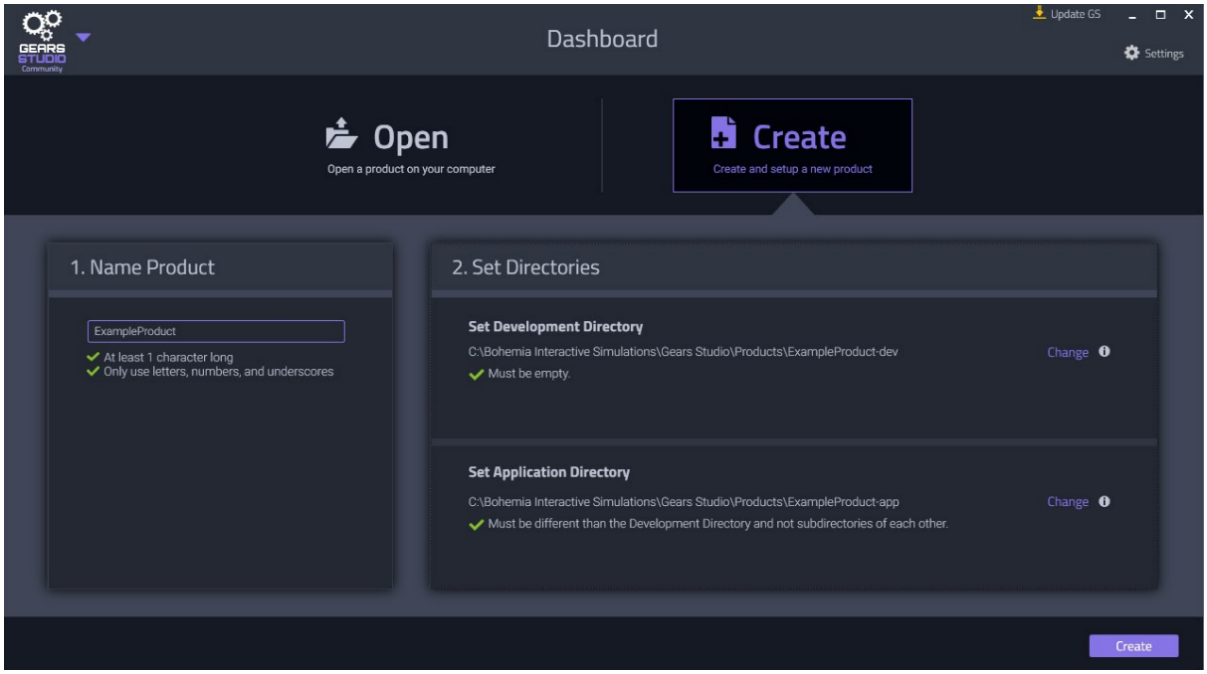
- To work with components in your product, see "Developing Components" (page 49).
- To work with a combined product solution in your Code Editor, use the Super Solution option in "Using Code Editors" (page 54).

Gears Studio also provides the following additional product operations:

- "Deleting Products" (page 47)

## 6.1. Creating Products

Create new products from the "Dashboard" (page 35).




**Follow these steps:**

1. Open the Gears Studio Dashboard, and click **Create**.

   The Name Product panel opens.

2. Input a name for your product, with the following conditions:

   - Must be at least 1 character long.
   - Must only use letter, numbers, and underscores _.

The Set Properties panel opens.

3. In the **Set Properties** panel, set the Development Directory.

   Click **Change**, browse for a folder, and click **OK**.

   **Note:** The Development Directory must be empty.

   Gears Studio uses this folder to store your code for the product and it must be empty.

4. In the **Set Properties** panel, set the Application Directory.

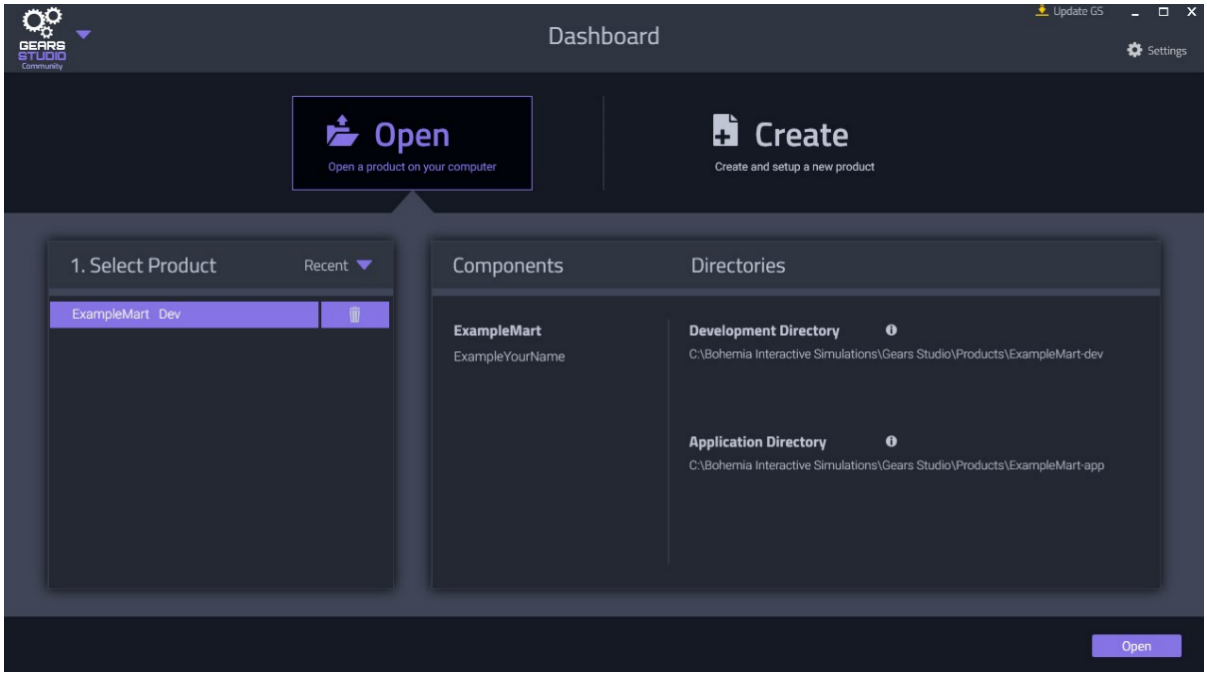
   Select **Custom**, click **Change**, browse for a folder, and click **OK**.

   Gears Studio uses this folder to store the built product application.

Gears Studio creates the product, and displays the "Product Page" (page 35).

## 6.2. Opening Products

Access existing products from the "Dashboard" (page 35).




**Follow these steps:**

1. Open the Gears Studio Dashboard, and click **Open**.

   The Dashboard displays a Select Product panel with a list of previously created products that exist on your computer,

2. **Optional:** Use the drop-down to reorder the products in **Recent** or **Alphabetical** order.
3. Click a product name to select it.

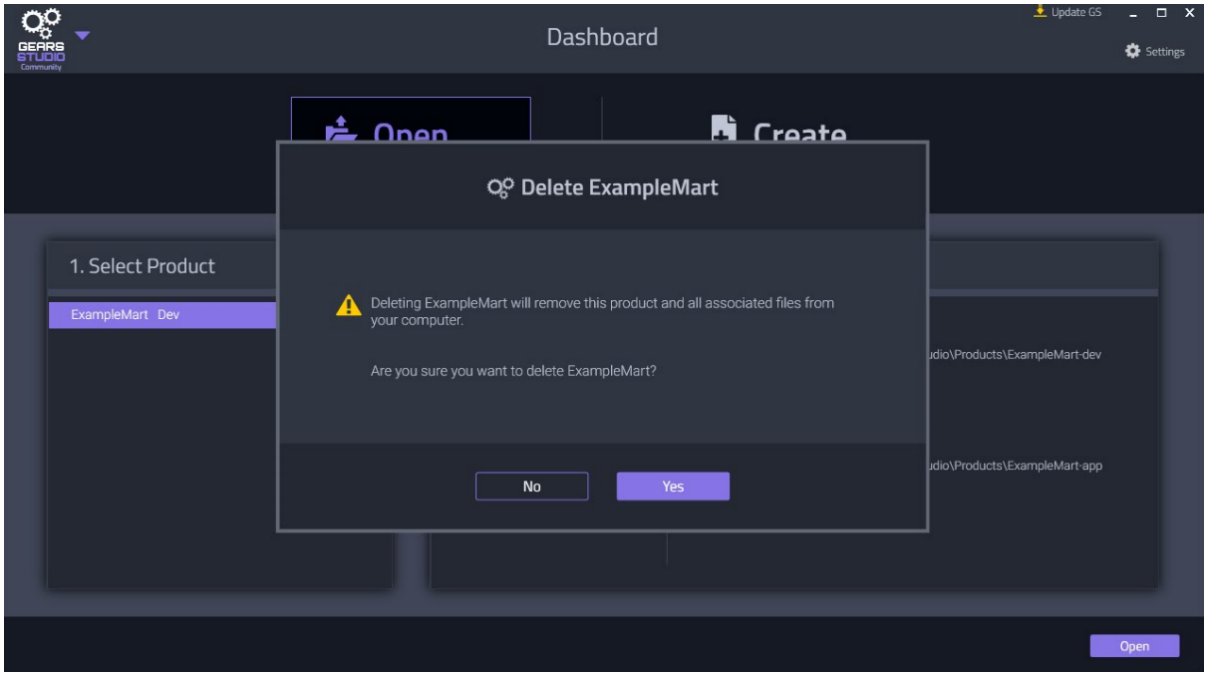
   The Dashboard displays the Product Information panel.

4. Click **Open**.

Gears Studio displays the "Product Page" (page 35).

## 6.3. Deleting Products

Delete products from the "Dashboard" (page 35).




**Follow these steps:**

1. Open the Gears Studio Dashboard, and click **Open**.

   The Dashboard displays a Select Product panel with a list of previously created products that exist on your computer,

2. **Optional:** Use the drop-down to reorder the products in **Recent** or **Alphabetical** order.
3. **Optional:** Click a product name to select it.

   The Dashboard displays the Product Information panel.

4. Click the **Trash Icon** next to a product name.

   Gears Studio displays a Delete Confirmation panel.

5. Click **Yes** to confirm deletion.

   **Warning**: Deleting a product will remove it and all associated files from your computer. This action cannot be undone.

Gears Studio deletes the product and all its associated files from your computer.

# 7. Developing Components

A Gears Component consists of a Dynamic-Link Library (DLL) and resource files. The DLL provides the Component services. A service within a Gears Component is accessible using an Application Programming Interface (API).

Gears Components are standardized to enable powerful functionality such as:

- Dependency management
- Versioning
- Distribution of versioned binaries (Gears Studio Professional)
- One-click download of binaries or source code (Gears Studio Professional)
- Code signing (Gears Studio Professional)

In Gears Studio you work with components within the context of a product.

Before working with components, review the following concepts:

- "Component Concepts" (page 10)
- "Component States" (page 10)
- "Development Files and Folders" (page 11)

Access and start working with components from the "Product Page" (page 35):

- "Creating Components" (page 49)
- "Importing Components" (page 52)

Select Components in the Product Page to enable "Component Tools" (page 39) and perform the following actions:

- "Editing Components" (page 52)
- "Generating Components" (page 53)
- "Using Code Editors" (page 54)
- "Removing Components" (page 55)

To work with APIs in your components, see "Developing APIs" (page 58).

During Component development you may also want to perform the following processes:

- "Working with Source Control" (page 55)
- "Example Build Server Setup" (page 56)
- "Upgrading Code Editors" (page 57)

## 7.1. Creating Components

Create new components as part of your product from the "Product Page" (page 35).

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. Click the **Add** icon and select **Create**.

   The "Component Window" (page 41) opens in Create Component mode.

3. In the **Component Name** tab, input a unique name.

   The component name cannot be changed after the initial component generation. Make sure that you use an appropriate name that accurately describes what the component does.

4. Select the **Source Control** tab to specify your component source control.

   1. Select the **System**: **Git** or **None**.

      **Note: None** is not recommended as it limits several features of Gears Studio.

   2. Input the **Location** using the URL of the remote Git repository.

      **Note:** The new remote repository should already exist and be empty. For more information, see "Working with Source Control" (page 55).

5. Select the **References** tab to manage the component references.

   References enable you to manage component references. It displays the following:

   - **Implemented APIs**

     Implemented APIs are APIs that are exposed for other components to use. By Implementing an API you are stating and advertising that this component implements a specific feature set that is defined by the API functions. When you choose to Implement an API all of the functions are initially empty and ready to populate with the logic to make them work.

     These API functions allow other components to interact with your component to request data, perform operations, configure options, and more.

   - **Dependencies**

     Dependencies are APIs or other resources that are available in the generated solution. Implemented APIs are automatically added to this list and cannot be removed.

     Dependencies are any resource that the component wants to interact with. Examples include APIs other components create, 3rd-party libraries, and command line tools

     Adding a dependency does not change your code at all. It simply makes the resource available within the code solution.

   - **Locked References**

     The Lock Icon  indicates that the implemented API or dependency can not be changed or removed because the component requires it to function properly. All components must implement ComponentAPI which enables the component to be loaded by Gears and request other APIs.

6. Use **Manage Implemented APIs** to indicate which APIs the component exposes for other components to use:

   - **To add an API:**

     1. Click the **Add Icon**  .

     2. Use the **Select** drop-down to select the API to implement from the list.

        The list displays any APIs available locally.

     3. Use the **Version** drop-down to select the version of the API to implement.

        A component may only implement a single major version of an API, but may implement as many major versions as required.

For example, you cannot implement versions 1 and 1-rev2 of an API because they both have the same major version.

- **To remove an API:**

  Click the **Trash Icon** next to the API name.

  **Note:** APIs that are manually removed from the list, are not re-added or updated when new versions of the API are created.

"Creating APIs" (page 58) automatically adds them to the Implemented APIs list. In most cases the component that creates the API implements it. However, if the component does not implement the API, you can remove the API from the implemented APIs list.

7. Use **Manage Dependencies** to make APIs and other resources to available in the generated solution:
   - **To add an API:**
     1. Click the **Add Icon**.
     2. Use the **Select** drop-down to select the API dependency from the list.

        The list displays any APIs available locally.
     3. Use the **Version** drop-down to select the version of the API.
   - **To remove an API:**

     Click the **Trash Icon** next to the API name.

   Implemented APIs are automatically added to the Dependencies list.
8. Select the **API Editor** tab to define new feature sets for components to interact with.

   It is not necessary to create APIs at this point. You can continue and create them later, if required.

   For more information, see "Developing APIs" (page 58).
9. Click **Generate**.

Gears Studio creates the new component, performing the following actions:

- Creates a new component folder in the Development Directory of your product.
- Automatically generates standard files and folders in the component folder:
  - README
  - Code Projects and Solution
  - Properties files
  - Header and Source files
  - APIs created during component creation.
  - Config files that describe the component, its APIs, and dependencies.

  For more information, see "Development Files and Folders" (page 11).
- Closes the Create Component window.

- Updates the "Product Page" (page 35) and "Component Card" (page 38) with the new component in Inactive State.

  For more information, see "Component States" (page 10).

Select the Component Card and use the "Component Tools" (page 39) to perform component actions.

## 7.2. Importing Components

Import existing components to a product from the "Product Page" (page 35).

Components are available to import from the local Development Directory.

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).
2. Click the **Add Icon** and select **Import**.

   The Import Component window opens.
3. Click the **Add Icon** to open a set of component selection options.
4. Expand the **Select** drop-down and select a component to import.

   Start typing a component name to find it in the list.

   The list displays any components available locally in the Development Directory.
5. **Optional:** Repeat from Step 2 to import additional components.

   Click the **Trash Icon** for a component to remove it from the import list.
6. Click **Done** to import the selected components.

Gears Studio imports the components, adding their "Component Card" (page 38) to the "Product Page" (page 35).

Gears Studio adds the components to the Development Directory of your product and marks them as **Inactive**.

Select Component Cards and use the "Component Tools" (page 39) to perform component actions.

## 7.3. Editing Components

Edit components as part of your product from the "Product Page" (page 35).

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).
2. To open the "Component Window" (page 41) for editing, do one of the following:
   - To edit multiple components, select components, and click **Edit** in "Component Tools" (page 39).
   - To edit a single component, double-click its "Component Card" (page 38) or expand its menu, and select **Edit**.

   A Component Window opens for each selected component.
3. Select the **Source Control** tab to specify your component source control.

1. Select the **System**: **Git** or **None**.

   **Note: None** is not recommended as it limits several features of Gears Studio.

2. Input the **Location** using the URL of the remote Git repository.

   **Note:** The new remote repository should already exist and be empty. For more information, see "Working with Source Control" (page 55).

4. Select the **References** tab to modify the component references.

   For more information, see References in "Creating Components" (page 49).

5. Select the **API Editor** tab to modify the component APIs.

   For more information, see "Developing APIs" (page 58).

6. Click **Generate**.

Gears Studio updates the component, closes the Component Window, and refreshes the "Product Page" (page 35).

For more information about the generated Code Solution, see "Development Files and Folders" (page 11).

## 7.4. Generating Components

Generating Components in Gears Studio creates code solutions ready to use in your preferred Code Editor.

**Important!** The specific files generated by Gears Studio are determined by the Code Editor selected during "Gears Studio Setup" (page 14) or in the "Developer Tools Settings" (page 65).

Gears Studio provides the following methods to generate components:

- "Generating from the Product Page" (page 53)
- "Generating from the Component Window" (page 54)

Code Solutions are also generated as the last step when "Creating Components" (page 49).

- "Component Folders" (page 11) describes the generated files and folder structure.
- "Project Files and Property Sheets" (page 13) describes additional files auto-generated by Gears Studio.

### 7.4.1. Generating from the Product Page

You can generate a single or multiple components from the "Product Page" (page 35).

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. To generate components, do one of the following:

   - To generate multiple components, select components, and click **Generate** in "Component Tools" (page 39).
   - To generate a single component, expand its "Component Card" (page 38) menu, and select **Generate**.

Gears Studio generates the code solutions for the selected components.

For more information, see "Component Folders" (page 11) and "Project Files and Property Sheets" (page 13).

### 7.4.2. Generating from the Component Window

You can generate a single component while "Editing Components" (page 52).

**Follow these steps:**

1. Open a component in its "Component Window" (page 41).
2. Click the **Generate Button** in any of the Component Window tabs.


Gears Studio generates the code solution for the component.

For more information, see "Component Folders" (page 11) and "Project Files and Property Sheets" (page 13).

## 7.5. Using Code Editors

Use a Code Editor to edit the component code, build and deploy components, and to debug the product.

Set your preferred Code Editor during "Gears Studio Setup" (page 14) or change your "Developer Tools Settings" (page 65).

Gears Studio supports the following Code Editors:

- Microsoft Visual Studio 2010
- Microsoft Visual Studio 2012
- Microsoft Visual Studio 2013
- Microsoft Visual Studio 2015
- Microsoft Visual Studio 2017

Create a Code Solution by "Generating Components" (page 53).

To access and use your Code Editor, see the following sections:

- "Opening Code Editors" (page 54)
- "Debug Settings" (page 55)

**Note:** If you change Code Editors and need to update already built Code Solutions, see "Upgrading Code Editors" (page 57).

### 7.5.1. Opening Code Editors

Open your Code Editor from the "Product Page" (page 35).

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).
2. To open code solutions for components do one of the following:

- To open all components as a single combined solution, click **Super Solution** in "Product Tools" (page 36).
- To open multiple code solutions, select components, and click **Code Editor** in "Component Tools" (page 39).
- To open a single code solution, expand its "Component Card" (page 38) menu, and select **Code Editor**.

Gears Studio opens a Code Editor window for each selected component or for the combined super solution.

For more information about the Code Solution, see "Development Files and Folders" (page 11).

### 7.5.2. Debug Settings

Before you can run your product set the Debugging Properties for the project.

- Set the **Command** to the product executable in your Application Directory.
- The **Working Directory** defaults to the Application Directory for your product.

## 7.6. Removing Components

Remove components from products in the "Product Page" (page 35).

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. To remove components, do one of the following:
   - To remove multiple components, select components, and click **Remove** in "Component Tools" (page 39).
   - To remove a single component, expand its "Component Card" (page 38) menu, and select **Remove**.

Gears Studio removes the selected components from the product. Removed products are not deleted from the Product Development Directory.

To restore a removed component, see "Importing Components" (page 52).

## 7.7. Working with Source Control

Gears Studio fully integrates with Git for distributed version control.

To use the Source Control features of Gears Studio, create a new remote repository using your preferred Git Host, for example GitHub or GitLab.

When connecting new components to your repository, we recommend creating empty remote repository locations for the component. Gears Studio creates placeholder files including `ReadMe` and `.gitignore` files.

- On GitHub, do not select the **Initialize with readme** option.
- On GitLab, projects are created empty by default.
- For other hosts, refer to their documentation.

When creating remote repositories, we recommend giving the repository the same name as your component to avoid confusion.

Once you have an empty remote repository for a component, use the URL in the Source Control step when "Creating Components" (page 49).

Gears Studio also integrates with Source Control Clients. Select your preferred client during "Gears Studio Setup" (page 14) or in "Developer Tools Settings" (page 65).

**Note:** Gears Studio does not provide SSH support. Gears Studio prompts you for repository credentials when they are required.

Use a Source Control tool to connect your components to a source control repository.

**To use Source Control:**

- Set up a remote repository using your preferred Git host.
- Specify your preferred Source Control Client during "Gears Studio Setup" (page 14) or in "Developer Tools Settings" (page 65).
- Specify remote repository locations while "Creating Components" (page 49).
- Modify Source Control Settings while "Editing Components" (page 52).

While working with components at the product level, you can open your Source Control client to perform repository actions.

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. To open your Source Control Client, do one of the following:
   - To manage multiple components, select components, and click **Src Control** in "Component Tools" (page 39).
   - To manage a single component, expand its "Component Card" (page 38) menu, and select **Src Control**.

Gears Studio opens a Source Control Client window for each selected component to enable repository actions.

## 7.8. Example Build Server Setup

Once you have working components, being able to share them is a good next step. Setting up the infrastructure to support the sharing of new component packages and APIs is a critical step in taking full advantage of Gears Studio. It is not strictly required, but strongly encouraged as a best practice.

Everything needed to set up automatic building is included within the component repository and consists of command-line calls using the Giri command-line tool. For more information, see "Giri Command-Line Tool" (page 69).

This topic creates a simple build server setup that performs a pre-merge step. However, you can expand and customize this as required. For example, by running unit tests on different system setups.

**Note:** All commands in the following example are run from the `/build/giri/` folder in the Development Directory component folder, where `giri.exe` is located. If you run Giri from a different location, adjust the command and parameters.

**Follow these steps:**

1. Perform a pre-merge step when a new commit is pushed that is not on the master branch. We recommend the following build and test commands:
   - To build, run the following command:

     ```
     giri.exe Build --component_folder "../../" --dependency_file "Dependencies.yml"
     ```

   - To run the unit tests, run:

     ```
     giri.exe RunTests --component_folder "../../"
     ```

2. Modify the VersionConfig file:

   Component packages are versioned using Semantic Versioning (see https://semver.org/), expressed in the form `Major.Minor.Patch`.

   Packages are versioned using a manually specified version. This ensures that you have control over versioning, that versions are not accidentally overwritten, and that versioning is not too difficult to deal with.

   The manual version is controlled by the VersionConfig file located at:
   `/ComponentName/build/VersionConfig.yml`

   Inside the file are properties for Major and Minor version, and Patch number. The properties start at 0 for newly created components and can be manually changed as required.

3. Test the build server:

   1. Open your Source Control client. If the remote repository is empty, push a single file to master before making a branch.

      The `readme` is a good choice, but any file is ok.

   2. After pushing a file to the remote repository on master, make a branch off that and push the remaining files.

      The pre-merge step should run to build your code and run the unit tests. Check the output to verify that everything worked as expected.

## 7.9. Upgrading Code Editors

Gears Studio generates Code Solutions specifically for your preferred Code Editor, specified during "Gears Studio Setup" (page 14) or in "Developer Tools Settings" (page 65).

If for any reason, you need to change already built code solutions to a different Code Editor you can edit code solutions directly in the Product Development Directory.

For example, to upgrade a code solution from Microsoft Visual Studio 2015 to 2017.

**Follow these steps:**

1. Make sure the project is not open in Microsoft Visual Studio.
2. Rename the Development Directory folder on disk:

   `/ComponentName/build/vs2015` to `/ComponentName/build/vs2017`
3. Open `/ComponentName/build/ComponentConfig.yml` in a text editor.
4. Change the `Compilers:` entry from `vs2015` to `vs2017`.
5. Open the solution in Microsoft Visual Studio 2017 and proceed to upgrade it.
6. Generate the component inside Gears Studio and ensure the Code Editor button works.
7. Change the **OutputDirectory** in Microsoft Visual Studio for all 3 projects to reflect the new version: from `vs2015` to `vs2017`.

# 8. Developing APIs

Components in the Gears framework communicate using APIs. These APIs must conform to a very specific set of rules. For instance, the API must adhere to certain C Programming language standards. Using Gears Studio to create APIs ensures that these rules are followed.

In Gears Studio you always work with APIs in the context of a component.

Before working with APIs, review the following API concepts:

- "API Versioning" (page 11)

Access and start working with APIs from the "API Editor" (page 42):

- "Creating APIs" (page 58)
- "Opening APIs" (page 59)
- "Versioning APIs" (page 61)
- "Editing API Code" (page 59)
- "API Settings" (page 61)
- "Searching APIs" (page 62)
- "Removing APIs" (page 63)

Gears Studio automatically saves changes made in the API Editor, but does not apply them to the Code Solution until the component is generated. To apply your API changes, see "Generating Components" (page 53).

## 8.1. Creating APIs

Create APIs for a component from the "API Editor" (page 42).

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. Select the component, to add the API to, and click **Edit** in "Component Tools" (page 39) or its "Component Card" (page 38) menu.

   The Component opens in the "Component Window" (page 41).

3. Select the "API Editor" (page 42) tab.

4. In the "Created APIs" (page 43) panel, under **Create API:** click the **Add Icon**  .

   Gears Studio displays the Name API input.

5. Input a unique name, and click **Create**.

   **Note:** Gears Studio automatically adds *API* to the end of the name.

Gears Studio creates a blank API template at Version 1 and opens the "API Code Tab" (page 43) and "API Details" (page 44).

Continue to any of the following topics to continue developing the API:

- "Editing API Code" (page 59)
- "API Settings" (page 61)

- "Versioning APIs" (page 61)

## 8.2. Opening APIs

Review APIs for a component from the "API Editor" (page 42).

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. Select the component to review, and click **Edit** in "Component Tools" (page 39) or its "Component Card" (page 38) menu.

   The Component opens in the "Component Window" (page 41).

3. Select the "API Editor" (page 42) tab.
4. Click an API name in the Created APIs list to open its "API Code Tab" (page 43) and "API Details" (page 44).

Continue to any of the following topics to modify the API:

- "Editing API Code" (page 59)
- "API Settings" (page 61)
- "Versioning APIs" (page 61)

## 8.3. Editing API Code

Edit API code in the "API Editor" (page 42) while "Creating Components" (page 49) or "Editing Components" (page 52).

Gears Studio implements the following edit restrictions to protect existing APIs:

- Existing versions are not editable. See "Versioning APIs" (page 61) to create a new revision or major version to edit.
- For new major versions, you cannot edit any existing typedefs, enums, or constants created in previous versions.
- For new revisions, you can edit any API element created in the same major version.

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. Select the component to review, and click **Edit** in "Component Tools" (page 39) or its "Component Card" (page 38) menu.

   The Component opens in the "Component Window" (page 41) displaying the "API Editor" (page 42).

3. Click an API name in the Created APIs list to open its "API Code Tab" (page 43) and "API Details" (page 44).

   An editable API displays the **Add Icon**  in the API Code Editor.

4. Do any of the following:


- **To Add API Elements:**
  1. Click the **Add Icon** to expand the API Menu.
  2. Click an **API Element Icon** to add an API Element.

     Gears Studio adds template code for the API Element and adds an entry to the Table of Contents in the "API Details" (page 44) Version tab.
  3. Input the appropriate content in the highlighted inputs.
- **To Edit API Elements:**
  1. Locate the API element to edit:
     - Use the API Code Editor scrollbar.
     - Use the Table of Contents in the "API Details" (page 44) Version tab.
     - By "Searching APIs" (page 62) in the "Created APIs" (page 43) panel.
  2. Modify the highlighted API elements to edit the API.
- **To Remove API Elements using the API Code Editor:**
  1. Locate the API element to remove:
     - Use the API Code Editor scrollbar.
     - Use the Table of Contents in the "API Details" (page 44) Version tab.
     - By "Searching APIs" (page 62) in the "Created APIs" (page 43) panel.
  2. Click the **Trash Icon** next to the API element to remove.

     A Delete Confirmation panel opens.
  3. Click **Yes** to confirm the removal.
- **To Remove API Elements using API Details:**
  1. Select the Version tab in "API Details" (page 44).
  2. Drag an API Element from the list to the **Trash Icon** at the bottom of the Version tab.

     A Delete Confirmation panel opens.
  3. Click **Yes** to confirm the removal.
- **To Rearrange API Elements:**
  1. Select the Version tab in "API Details" (page 44).
  2. Drag and drop an API Element in the Table of Contents.

  **Note:** API Elements can be rearranged only within their API type, the order of API Element types is fixed.

5. Gears Studio displays warnings and coding standards messages when content is missing or incorrect.

   To configure the API Warnings to display, see "API Warnings Settings" (page 66).
6. For **Includes**, use the drop-down to select the version of the API to use.
7. When you have finished making changes, click **Generate** to update your code.

Your updated code is ready to use.

## 8.4. API Settings

Specify API instancing in the "API Editor" (page 42) while "Creating Components" (page 49) or "Editing Components" (page 52).

**Note:** Instanced APIs are not accessible through the Request API. Instead, instances are created on demand using the Allocate API and freed using the Free API. This enables you to associate specific data with an API instance (mimicking C++ class functionality). The first parameter of all functions in this API are automatically made APIType* api.

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. Select the component to view, and click **Edit** in "Component Tools" (page 39) or its "Component Card" (page 38) menu.

   The Component opens in the "Component Window" (page 41).

3. Select the "API Editor" (page 42) tab.
4. Click an API name in the Created APIs list to open its "API Code Tab" (page 43) and "API Details" (page 44).
5. Select **Settings** in the API Details panel.
6. Select **Instanced** to create an instanced API.

## 8.5. Versioning APIs

To help ensure that already implemented APIs are protected, Gears Studio provides API versioning.

To make changes to an API, create a new version or revision.

**Important!** Review "API Versioning" (page 11) before using this process.

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. Select the component to view, and click **Edit** in "Component Tools" (page 39) or its "Component Card" (page 38) menu.

   The Component opens in the "Component Window" (page 41).

3. In the "Created APIs" (page 43) list, click the name of the API to version.

   Gears Studio displays the "API Code Tab" (page 43) and "API Details" (page 44) for the API.

4. Select **Version** in the API Details panel.
5. Do one of the following:
   - To create a new version, click **Major Version**.
   - To create a new revision, click **Revision**.

**Note:** These buttons are disabled if you are already working with a new major version or revision or you have already made changes to the current version. Generate the component first to continue.

Gears Studio creates the new major version or revision of the API using automatic numbering.

Continue to any of the following topics to make your API changes:

- "Editing API Code" (page 59)
- "API Settings" (page 61)

## 8.6. Searching APIs

Gears Studio includes a search function in the "API Editor" (page 42) to help find items in component APIs.

**Follow these steps:**

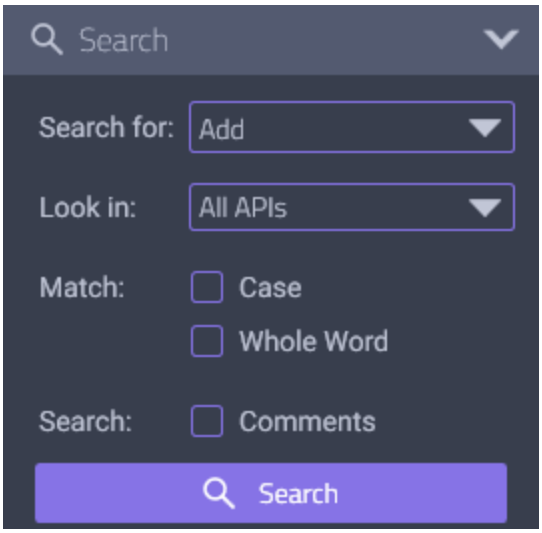
1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. Select the component to search, and click **Edit** in "Component Tools" (page 39) or its "Component Card" (page 38) menu.

   The Component opens in the "Component Window" (page 41).

3. In the Created APIs panel, click the **Search Arrow** to expand the Search Panel.



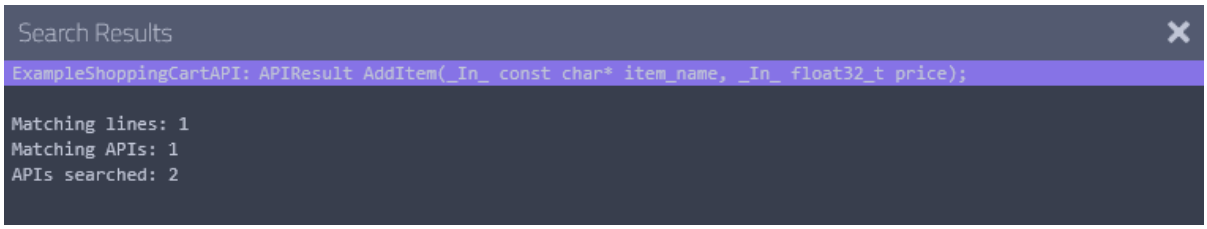

4. Input your search term in the **Search for:** input.

   **Note:** The drop-down displays recent search terms.

5. Use the **Look in:** drop-down to select the APIs to search:
   - **All APIs** - all APIs in the component.
   - **Current API** - the API in the active API Code Editor.
   - **All Open APIs** - all APIs with an open API Code Editor.
6. Select **Case** to perform the search matching the case of the search term.
7. Select **Whole Word** to exclude results where the search term is part of a word.

8. Select **Comments** to include search result in comments.
9. Click **Search**.

Gears Studio performs the specified search and opens a Search Results panel.




Click a result to navigate to that line in the "API Code Tab" (page 43).

### 8.6.1. Search Keyboard Shortcuts

Use the following keyboard shortcuts to perform API search actions in Component Windows.

| Shortcut | Action | Description |
|---|---|---|
| **Ctrl+F** | Search | Expands the API Search panel. |
| **F8** | Select Next Search Result | Selects the next search result in the Search Results window. |
| **Shift+F8** | Select Previous Search Result | Selects the previous search result in the Search Results window. |

## 8.7. Removing APIs

Remove APIs from components in the "Component Window" (page 41).

**Follow these steps:**

1. Open a product in the "Product Page" (page 35).

   For more information, see "Opening Products" (page 47).

2. Select the component to review, and click **Edit** in "Component Tools" (page 39) or its "Component Card" (page 38) menu.

   The Component opens in the "Component Window" (page 41).

3. In the "Created APIs" (page 43) list, click the **Trash Icon** next to the API to remove.

   A Delete Confirmation panel opens.

4. Click **Yes** to confirm the removal.

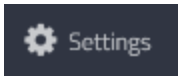
Gears Studio removes the API from the component.

**Note:** The API is not deleted from local repositories.

# 9. Configure Gears Studio Settings

Access and change your settings from the "Gears Studio Header" (page 33) in the Dashboard or Product Page.

Click **Settings** in the header to open the Settings panel.

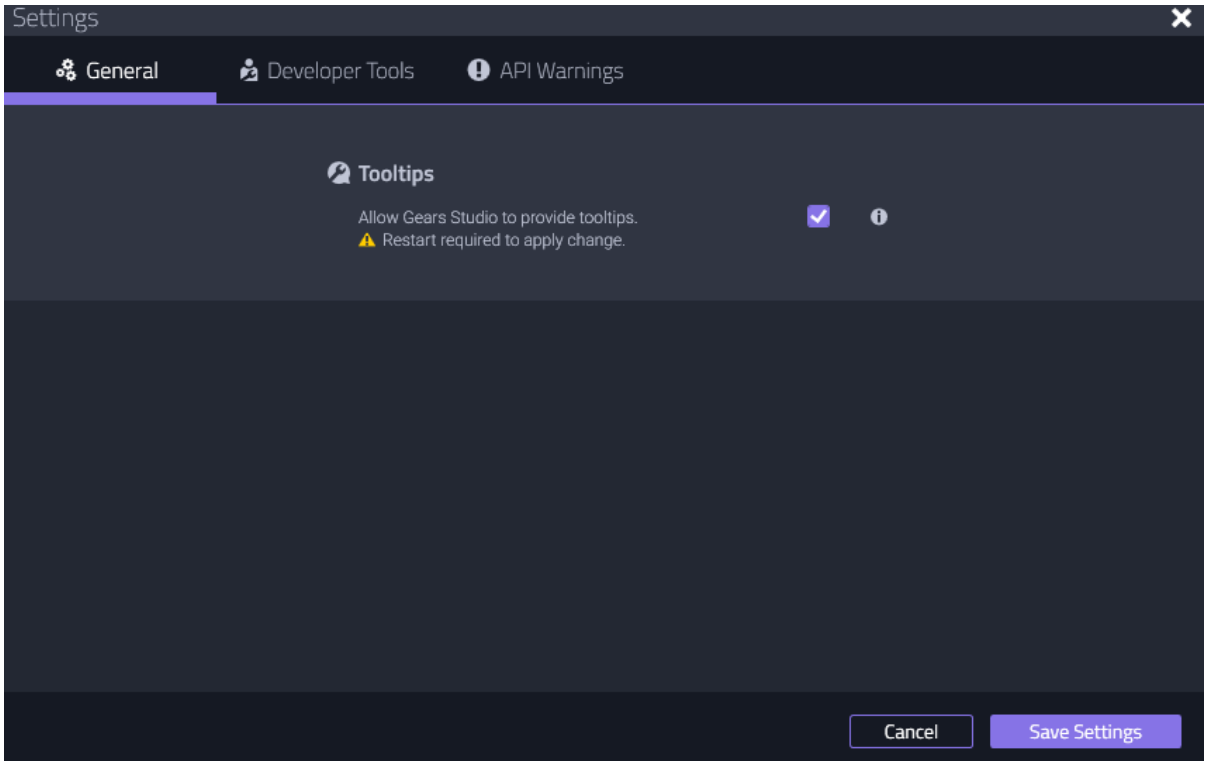



The Settings Panel displays tabs for the following settings:

- "General Settings" (page 64)
- "Developer Tools Settings" (page 65)
- "API Warnings Settings" (page 66)

## 9.1. General Settings

Use the General Settings to control general Gears Studio options.




**Follow these steps:**

1. Click **Settings** in the Dashboard or Product Page header.

   The Settings Panel opens.

2. Select **General** to display the general settings for Gears Studio.
3. Select **Tooltips** to enable additional help to display when you hover the mouse over specific UI elements.

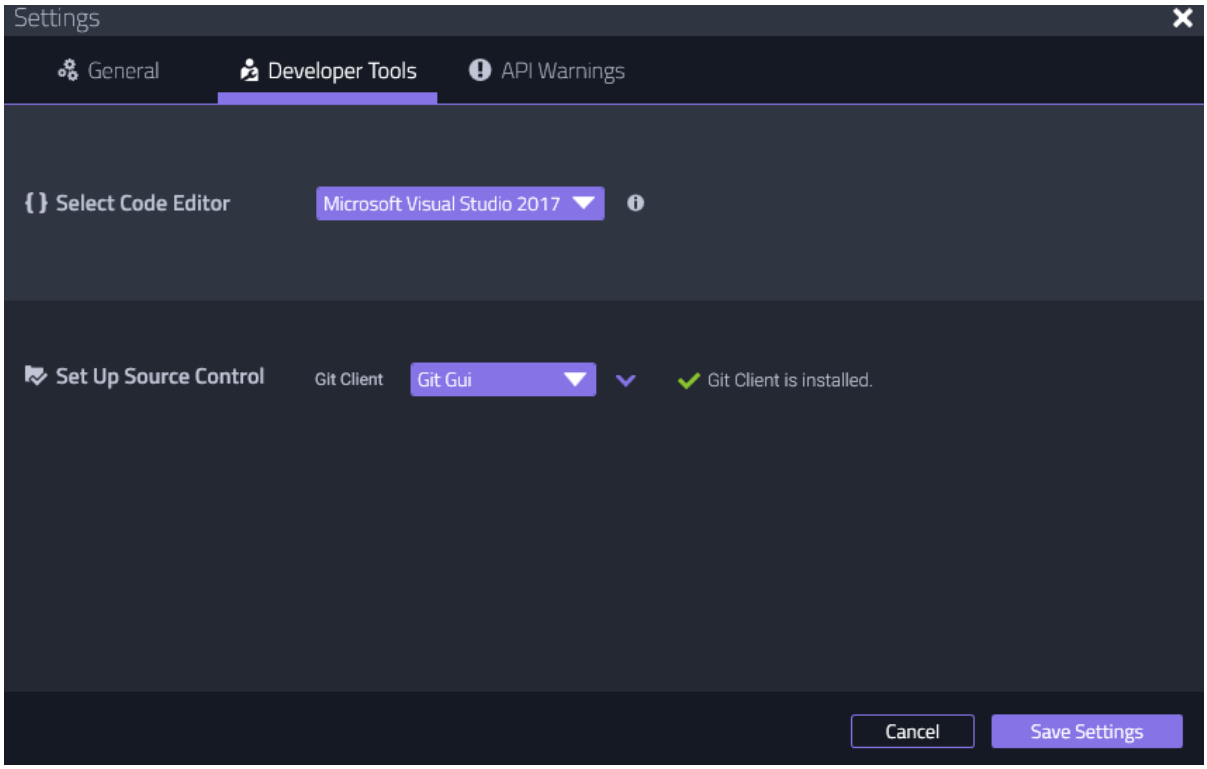
   **Note:** Restart Gears Studio after saving to apply any change to the **Tooltips** setting.

4. Click **Save Settings**.

Gears Studio updates the General Settings.

## 9.2. Developer Tools Settings

Use the Developer Tools Settings to change the tools you use, initially applied during "Gears Studio Setup" (page 14).




**Follow these steps:**

1. Click **Settings** in the Dashboard or Product Page header.

   The Settings Panel opens.

2. Select **Developer Tools** to display the tools settings for Gears Studio.

3. Use the **Select Code Editor** drop-down to select your preferred developer tool.

   Gears Studio supports the following Code Editors:

   - Microsoft Visual Studio 2010
   - Microsoft Visual Studio 2012
   - Microsoft Visual Studio 2013
   - Microsoft Visual Studio 2015
   - Microsoft Visual Studio 2017

   Gears Studio generates source code projects compatible with the selected code editor application. Gears Studio operations that launch a code editor launch the selected code editor application.

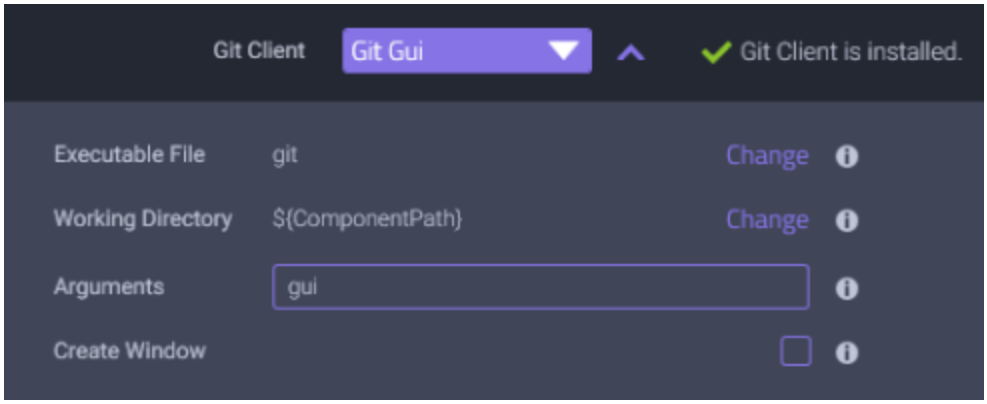
   **Note:** To modify already generated components for the new Code Editor, see "Upgrading Code Editors" (page 57).

4. Use the **Git Client** drop-down to select your preferred source control client.

   To configure advanced settings, **follow these steps**:

1. Click the **Arrow Icon** to expand the advanced settings.




2. To configure the **Executable File**, click **Change**, browse for your source client executable and click **OK**.
3. To configure the **Working Directory** that contains the source control client executable, click **Change**, browse for the directory, and click **OK**.
4. To launch the source control client with specific arguments, modify the **Arguments** input:

   If the argument variable `${ComponentPath}` is specified in the argument list, it is replaced with the path to the Component being opened.

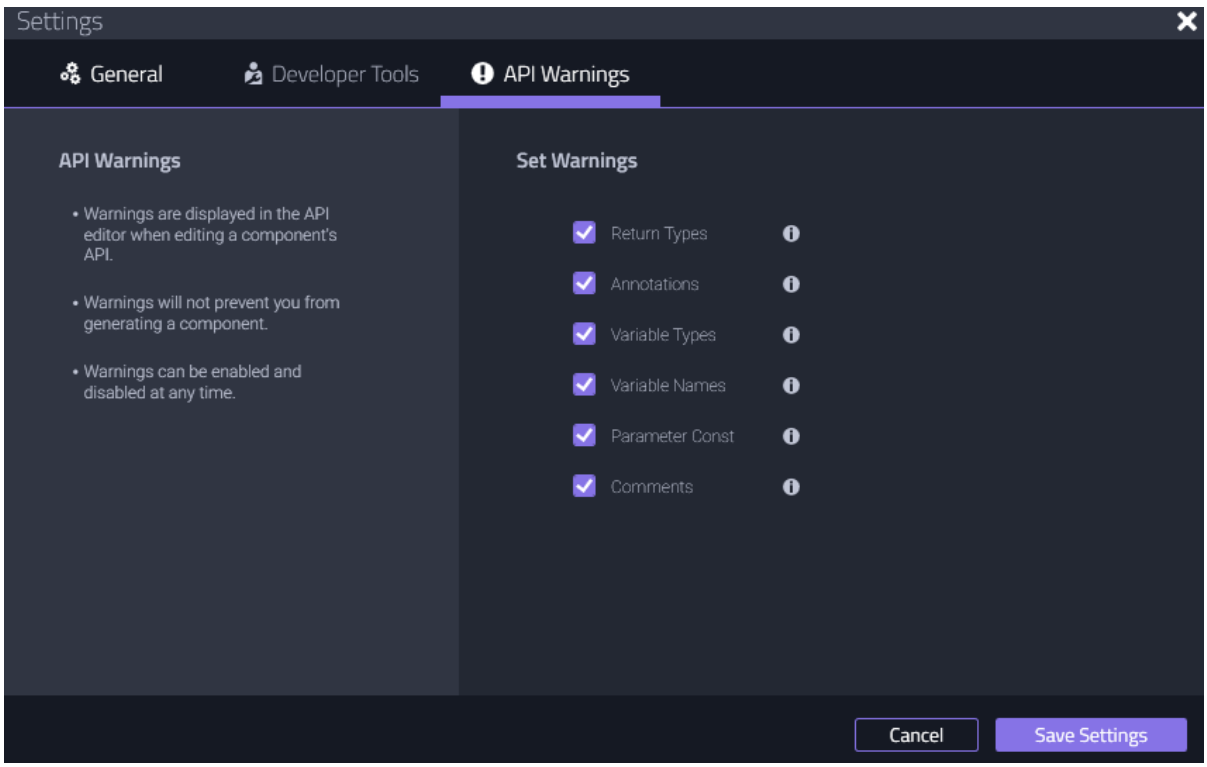
   For example: `--open ${ComponentPath}`
5. Select **Create Window** to specify that the source control client should launch in a new window.

Gears Studio operations that launch a source control client on behalf of the user launch the selected source control application.

5. Click **Save Settings**.

## 9.3. API Warnings Settings

Use the API Warning Settings to configure which warnings display in the "API Code Tab" (page 43).



**Follow these steps:**

1. Click **Settings** in the Dashboard or Product Page header.

   The Settings Panel opens.

2. Select **API Warnings** to display API warnings in the API Editor.

3. Select the Warnings to display:

   - **Return Types**

     Select to display a warning when return types are not `APIResult` or `void`.

   - **Annotations**

     Select to display a warning when parameters do not use annotations, such as:

     `_In_, _Out_, _Inout_opt_`

   - **Variable Types**

     Select to display a warning when variables do not use standardized types, such as:

     `int32_t, float32_t, float64_t`

   - **Variable Names**

     Select to display a warning when variable names do not adhere to the Snake Case naming convention, such as:

     `_my_member, my_parameter, some_variable`

   - **Parameter Const**

     Select to display a warning when a parameter is not marked as `const` and it meets the following criteria:

     - A pointer
     - Annotated with `_In_` or `_Inout_`

   - **Comments**

     Select to display a warning when an API element is missing comments.

4. Click **Save Settings**.

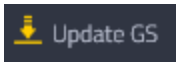
Gears Studio updates the API Settings. Only selected warnings display in the "API Code Tab" (page 43).

# 10. Update Gears Studio

Access updates from the "Gears Studio Header" (page 33) in the Dashboard or Product Page.

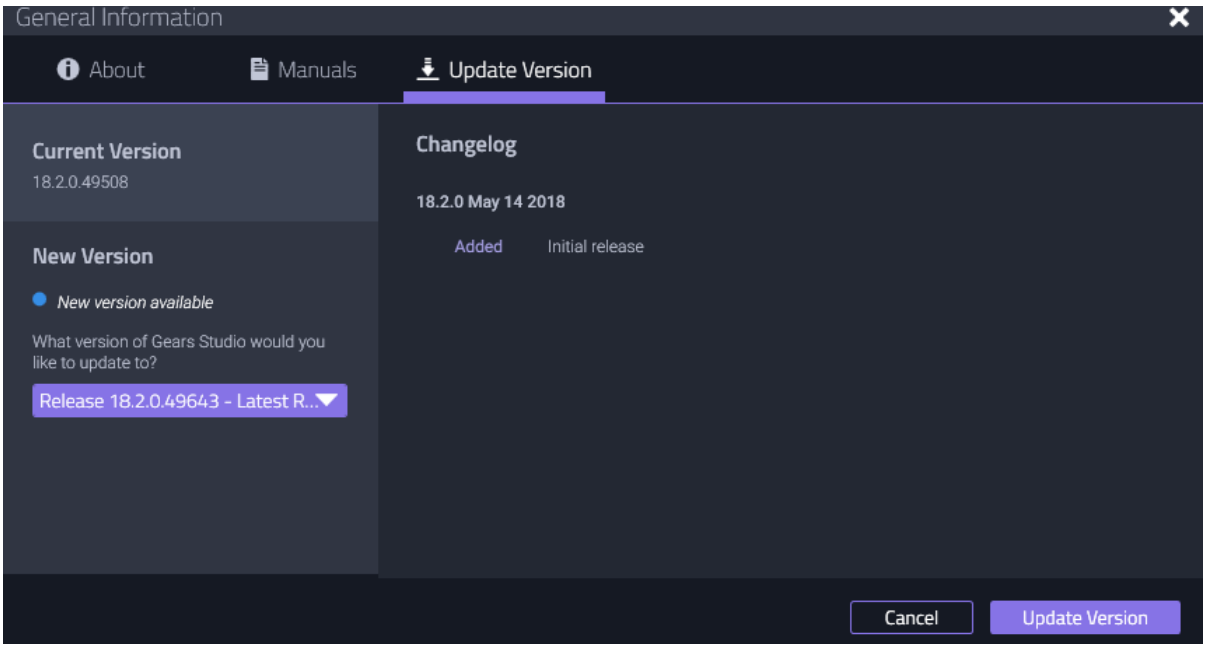
When a Gears Studio update is available the Update Indicator displays in the header.




**Follow these steps:**

1. In the Dashboard or Product Page header, click the **Update Indicator** or expand the Gears Studio Menu and select Update GS.

   The General Information Panel opens in the Update Version tab.




2. Use the **New Version** drop-down to select the version to update to.

   The Changelog refreshes to display the changes up to the selected version.

3. Review the Changelog and click **Update Version**.

   A Switch Version confirmation panel opens.

4. Click **Switch Version** to confirm the Gears Studio update.

Gears Studio updates to the selected version and restarts.

# 11. Giri Command-Line Tool

Giri (Gear-e) is an executable tool built from the same code used in Gears Studio, located in every component folder in the Development Directory:

`/ComponentName/build/giri/giri.exe`.

Gears Studio uses Giri to perform operations automatically. There is typically no need to run Giri manually, unless you want to run it on the Build Server, or to publish 3rd-party dependencies.

Giri throws an exception if any commands fail, indicating that the build pipeline should also fail. This can occur if there are compiler errors, unit test failures, and so on. You can disable error checking using `--no_exceptions`, however, this may cause you to miss failing build pipelines.

**Note:** Gears Studio must be closed before running most Giri commands.

**Syntax:** `giri [command]`

| Command | Arguments | Description |
|---|---|---|
| `PrintDependencyTree` | `component_folder`<br>`dependency_file` | Prints out the dependency tree of the component. |
| `Build` | `component_folder`<br>`dependency_file` | Builds the component. |
| `RunTests` | `component_folder` | Runs the component unit tests. |
| `Deploy` | `component_folder`<br>`output_directory`<br>`compiler`<br>`debug` | Deploys the component binaries to the output directory. |

The commands use the following arguments:

| Command Argument | Description |
|---|---|
| `component_folder` | The root directory of the component. Typically, a folder in the Development Directory. |
| `dependency_file` | The `.yml` file used to resolve dependencies. |
| `publish_file` | The `.yml` file used to publish dependencies. |
| `output_directory` | The output directory to deploy to. Typically, a folder in the Application Directory. |
| `compiler` | The compiler version of binaries to Deploy. |
| `debug` | Deploy debug binaries. The default is false. |

# 12. Known Issues

### Windows 7 and TLS Protocols

Gears Studio running on Windows 7 may not be able to connect to remote Git servers that require TLS 1.1 or 1.2 protocols.

Refer to the following Windows Support article:

- Update to enable TLS 1.1 and TLS 1.2 as a default secure protocols in WinHTTP in Windows

  https://support.microsoft.com/en-us/help/3140245/update-to-enable-tls-1-1-and-tls-1-2-as-a-default-secure-protocols-in

The Microsoft Update Catalog website contains a stand-alone package that can resolve issue KB3140245.

1. Use this link to download the appropriate stand-alone package:

   https://www.catalog.update.microsoft.com/search.aspx?q=kb3140245

   There are separate downloads for:

   - Windows 7 (32bit)
   - Windows 7 x64-based Systems
2. Once installed, refer back to the Windows Support article mentioned above.
3. Follow the steps for updating the Windows Registry or download the **Easy Fix** package:

   https://aka.ms/easyfix51044
4. Complete all the steps and then restart the computer.

Gears Studio should be able to connect to the remote Git servers.