Ocean

Software development framework

Ocean for Techlog 2024

Getting Started with Ocean

Version 2024.4

Contents

1.Welcome to Ocean for Techlog

Ocean for Techlog is an application development framework, part of the Ocean suite of SLB software development frameworks, focused on wellbore data processing and visualization. It allows the application developers to extend the functionality and workflows of the Techlog platform.

The Ocean framework provides a productive development environment that allows the developers to focus on science.

Ocean plug-ins are loaded on-demand by the Techlog end-user as libraries (dll) using the Techlog module manager.

A plug-in integrates in Techlog menus and workflows like native modules. They may appear as:

- activities started for instance through a menu item, which decide by themselves when they are finished. They are displayed as tasks in Techlog, such that you can monitor and possibly stop them manually.
- activities like worksteps which run within a Techlog workflow.

 All the code snippets in this document have been built with Ocean for Techlog 2024.4.

Ocean for Techlog Advantage

Ocean is built in the Qt (cute) environment. Qt is a cross-platform application framework that is widely used for developing application software with a graphical user interface. Qt uses standard C++ but makes extensive use of a special code generator (called the Meta Object Compiler, or moc) together with several macros to enrich the language. For software developers, the use of Qt is seen as a productivity enhancement.

Ocean is designed to promote code reusability for maintenance efficiency and robustness. The Ocean Framework enables independent development of decoupled modules. These modules can then be deployed independently of the main Techlog application. This enhances robustness while preserving the evolution of the Techlog platform.

Ocean also promotes the independence of data display and data access. Traditional applications produce data and provide sophisticated rendering and interactions for this data. This isolates them from other applications. In Ocean, data access and data display are not handled by the same classes. This promotes code reuse and data sharing in the same graphical environment. For instance, the **Logview** window simultaneously shows data processed by Petrophysics, Acoustics, and Geology modules. It becomes an essential tool for asset team collaboration.

Ocean for Techlog Architecture

Ocean for Techlog provides lifecycle management, a runtime environment, and a public API for plug-ins to interoperate with Techlog functionalities. Figure 1 shows how Ocean for Techlog provides the glue between Techlog and the plug-ins.

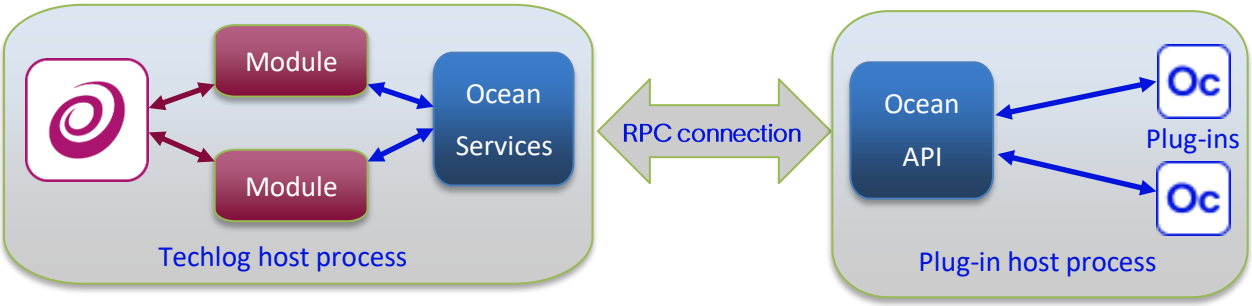


Figure 1: Ocean for Techlog architecture: Plug-in isolation

The Ocean for Techlog architecture is based on native C++ and the Qt framework, with plug-ins running outside of the main Techlog process. Each plug-in running in its own process provides stability and compatibility as it:

- allows plug-ins to run in debug mode with the release version of Techlog
- avoids conflicts between third-party libraries used by the different plug-ins
- allows debugging, fixing, recompiling and rerunning a plug-in without having to restart Techlog
- allows binary compatibility over multiple versions of Techlog and Qt
- allows isolation of Techlog in case of a crash of one plug-in

The Ocean for Techlog public API (**Slb.Ocean.Techlog.dll**) is the host for Ocean applications and is the environment in which the Ocean module needs to run. The public API provides:

- the domain objects and their data source
- the graphical environment in which the applications will display their data
- a common look and feel for all application user interface components

Access to the Techlog data model

The Ocean for Techlog API accesses these data types and properties of the Techlog data model:

- Well
- Dataset
- Variable (Well logs)
- Data properties
- Zonation

You extend the Techlog data model by creating new data objects which are called *plug-in domain objects*.

See the "Plug-in domain object" section in Ocean for Techlog Developer Guide - Plugin Domain Object - Importer&Exporter for more information on how to implement a plug-in domain object with Ocean for Techlog.

Ocean for Techlog UI Infrastructure

The Ocean for Techlog API does not limit itself to accessing the Data domain of Techlog. It also provides a rich environment for integrating the Ocean module with the graphic environment familiar to Techlog users.

The User Interface API provides functionality to customize many elements of the Techlog window system.

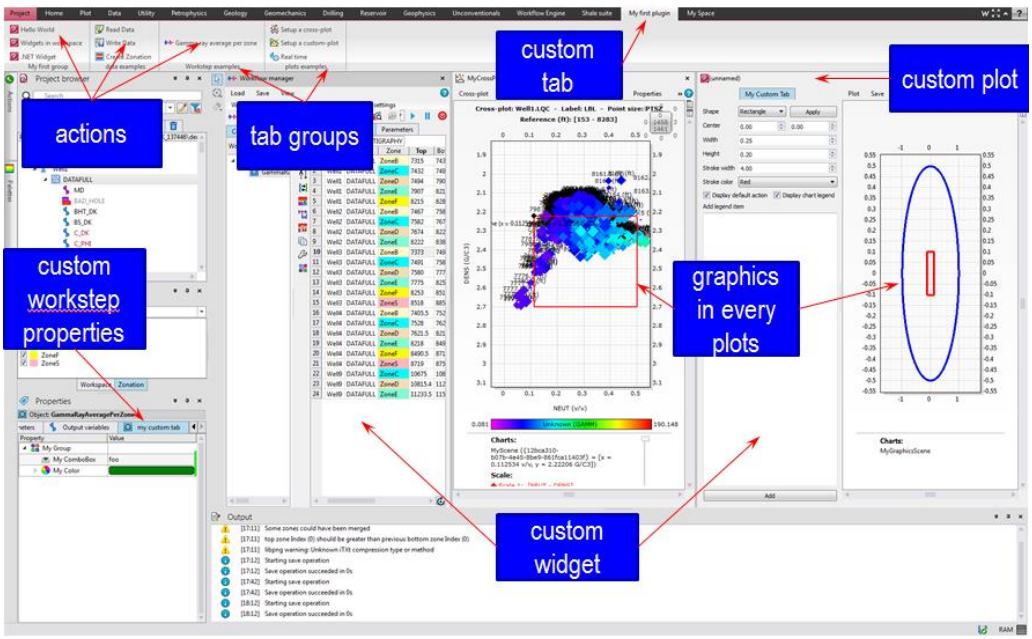


Figure 2: Techlog UI extensibility

Ocean provides the capability to extend Techlog's user interface functionality for the GUI to be tailored to the needs of new applications. Figure 2 shows some examples of what is customizable with the Ocean API:

1. Menu bar extensions:
    - Adding new tabs, groups and menus to the tbar (Techlog ribbon) or extending native Techlog menus
2. Windows:
    - Adding custom windows (Qt widgets) in the Techlog workspace
3. Plots:
    - Adding custom plots
    - Customizing native and custom plots by adding graphic items
    - Adding user interactions through graphic items
    - Extending the menu bar, tool bar and context menu of native and custom plots with custom tools
4. Workflow manager
    - Adding a custom user interface to an Ocean workstep
    - Extending workstep properties (Techlog properties editor) with a custom properties tab
5. Importer and Exporter
    - Extending import and export functionalities of the Techlog platform

Ocean for Techlog plug-in identity and activities

The **PluginIdentity** is an interface that the developer implements to declare some information about the plug-in, its list of activities, and the menu items used to trigger those activities.

This is the main entry point class of the plug-in and this class, compiled into the library, provides identity and support information on the plug-in.

Once the plug-in library is deployed into the **Extensions** folder of Techlog (it could be in the **Techlog**, **Company** or **User** folder), the end-user can enable or disable it in the Techlog module manager accessible through the **Navigation pane** > **Licensing** menu and click on the **Open the module manager** button.

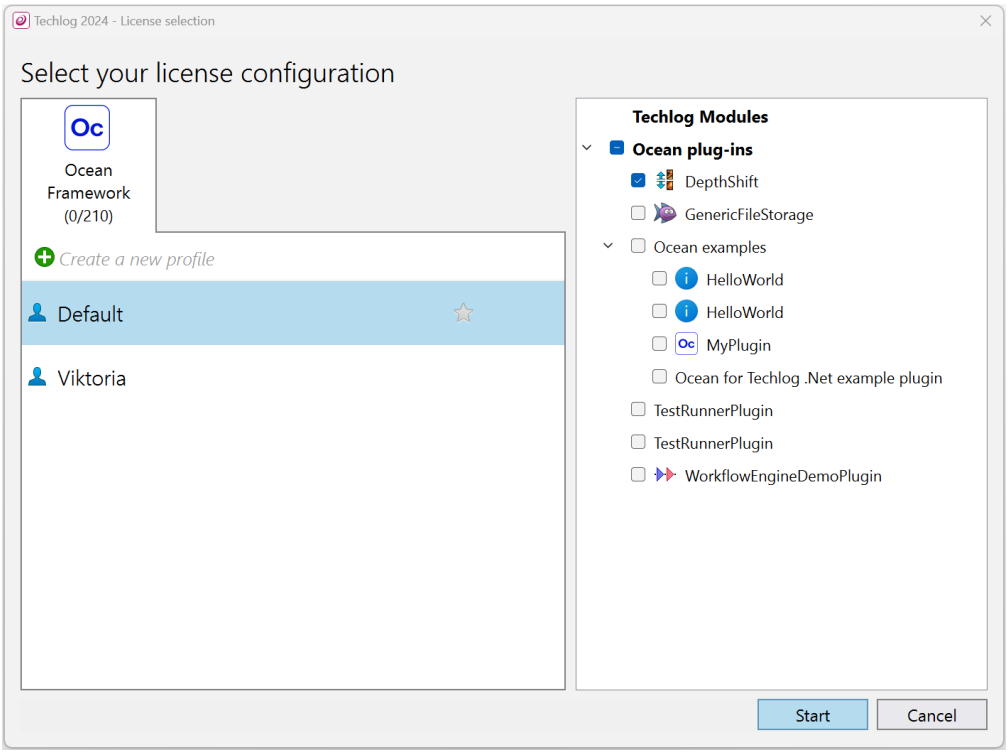


Figure 3: Techlog module manager

The module manager in Techlog manages the integration of the plug-in activities into Techlog: it loads and queries the plug-in, creates actions that launch the plug-in activities, and links them to menu items in Techlog.

In Techlog, a module is a set of functionalities associated with a license feature. A plug-in can contribute its activities into some Techlog modules. For instance, a plug-in can contribute an environmental correction workstep (associated with the environmental correction license) and can also add some geology-related processing to the WBI (wellbore imaging) module, that is available only if the user has also a WBI license. This means that the integration into the Techlog menus is dynamic, based on the Techlog modules enabled by the user, and therefore subject to license checks.

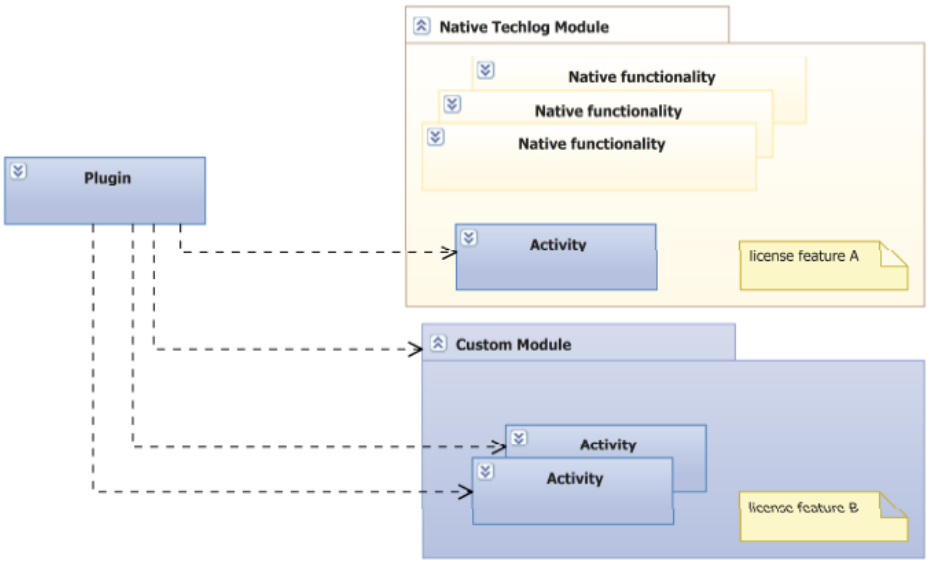


Figure 4: Plug-ins contribute activities to modules (native or custom); modules can be licensed

All the activities of a given plug-in run in a single process, and multiple instances of a given activity can run in that same process. This way, activities within a given plug-in can communicate between each other (for instance, multiple worksteps forming a workflow).

Ocean framework license

Ocean for Techlog is sold under a license feature called **Ocean_Framework** that makes **tlBase**, **tlAdvancedPlotting** and **tlPython** modules available.

Ocean_**Framework** license feature gives access to the Ocean for Petrel and Ocean for Studio development frameworks as well. You need to provide a dongle id when you order the license through the Ocean store.

Creating or opening a Techlog project with an Ocean framework license marks the project as tainted.

- Plots and reports accessing data from a tainted project have a watermark.
- Data export is prohibited.
- Once the project is tainted it can't be open with a regular Techlog license.

Qt LGPL notice

The Ocean framework is distributed with Qt LGPL 5.15.16 libraries. Per requirement of LGPL components used, you must provide with your plug-in a notice that LGPL code is being used. Do this by deploying with your plug-in dll (plug-in folder) the **LGPL.txt** file and the appropriately edited **README.txt** file that are shipped with the Ocean framework.

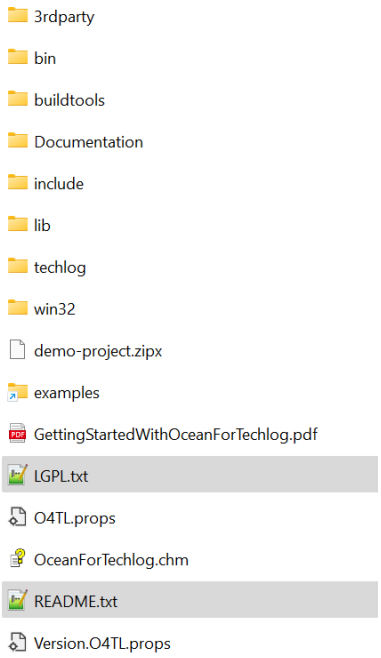


Figure 5: LGPL notice files

See the "Deploy folders and files with the plug-in dll" section for more information on how to deploy additional files in the plug-in folder.

Open and modify the **README.txt** files before deploying it with your plug-in, changing the "Ocean for Techlog Software" with the name of the plug-in at the beginning of the file.

Microsoft Visual Studio compiler supported with Ocean

The Ocean framework is deployed with libraries built with msvc143 (Visual Studio 2022) compiler version. This means that you can build your Ocean for Techlog plug-in with only the Visual Studio compiler 2022 version.

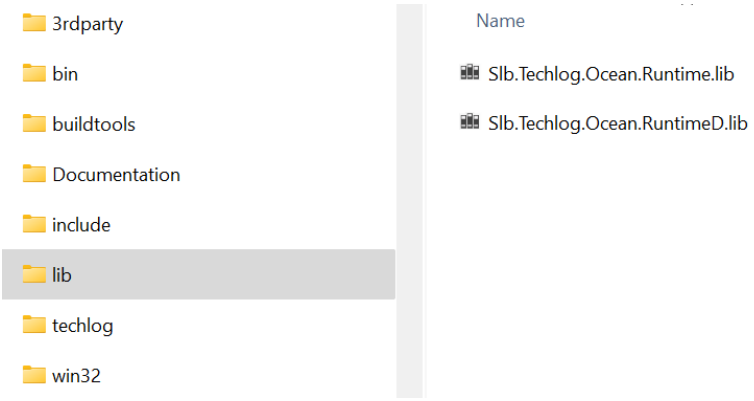


Figure 6: Ocean libraries

Binary compatibility

The commercial Ocean APIs from a Techlog major version release are binary compatible with all of the Techlog minor version releases. The best practice is to build your plug-in on the Ocean Framework major version so it will run on any minor releases of the same major Techlog version.

Note: If the plug-in depends on a new feature or a fix in a minor release you can build your plug-in against that Ocean framework minor release, but users of your plug-in will then have to adopt the corresponding Techlog minor release in order to run your plug-in.

Stability promise

The stability promise says that the Ocean API will be stable for minimum two years or two release cycles.

In order to implement new features or adopt new designs, the Ocean API may change over time. Any APIs that are going to be removed are marked with the "Deprecated" attribute and include when the deprecation happened and what the replacement API is. The API remains available as "Deprecated" for a minimum of one release cycle before the API is fully removed, and the plug-in code must be updated in this time frame.

For example, the **worksteps** method of the **Workflow** class was deprecated in the 2020.1 release.

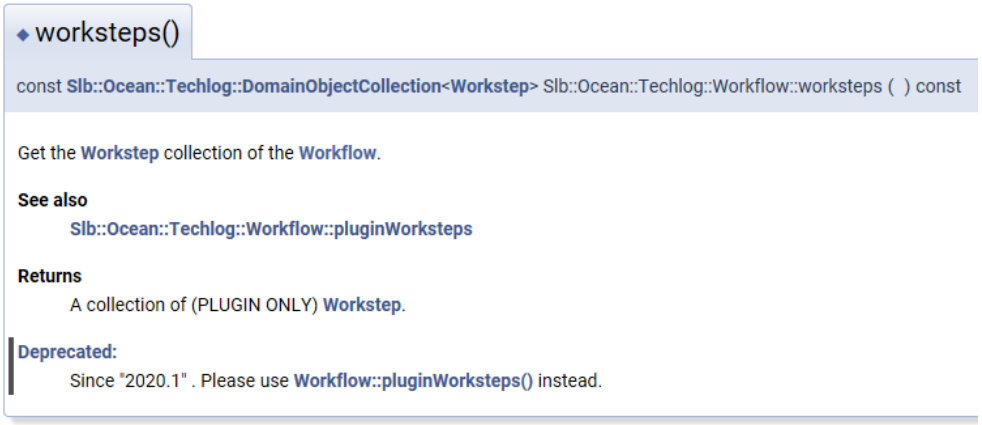


Figure 7: Deprecated API

Any plug-ins using the deprecated worksteps method must be changed to use the replacement method **pluginWorksteps** instead.

Since you might not have the time or resources to change your plug-in immediately, in the 2020.1 release both the new API **pluginWorksteps** and the old API **worksteps** are available. In the 2024.4 release, the old API **worksteps** is removed and is no longer available for use.

2.Install and setup the Ocean for Techlog development environment

Ocean for Techlog installation

The Ocean development environment is setup by Ocean for Techlog installer.

The installer first checks if the Techlog version corresponding to the Ocean Framework is installed on the user machine. The Ocean for Techlog package may be located anywhere on the disk.

1. Click Elevate button to run the Ocean installer with administrator privilege. (See Figure 8)

Figure 8: Ocean for Techlog administrator elevation

2. Browse the installation folder and click Next in the dialog window. (See Figure 9.)

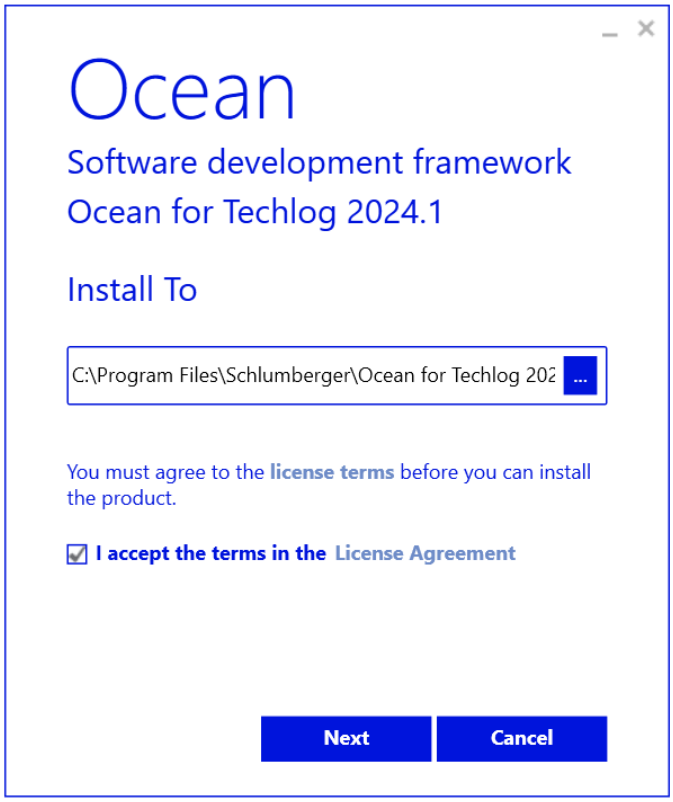


Figure 9: Ocean for Techlog install location

The installer checks:

- corresponding Techlog version is installed
- Visual Studio 2022 is installed with v143 compiler
- .NET Framework 4.8 is installed.

3. Click **Next** in the dialog window. (See Figure 10.)

Figure 10: Techlog, VS 2022, .NET Framework 4.8 are installed

If you have already a Techlog user folder defined on your system (**TLUSERDIR** environment variable), the sample plug-ins are deployed to this folder. Otherwise, the installer deploys sample plug-ins to the user profile's AppData in order to avoid any UAC (User Account Control) issues.

See the "Ocean for Techlog environment variables" section for more information on how to setup Ocean environment variables.

The installer shows Visual Studio components installed with Ocean.

4. Select all components and click Install in the dialog window. (See Figure 11.)

Figure 11: Visual Studio components

5. After the installation a reboot may be required to get all Ocean for Techlog Visual Studio extensions properly installed. (See Figure 12.)

Figure 12: Setup successful

Ocean for Techlog package content

The Ocean for Techlog Framework is deployed by the installer. The Ocean for Techlog package has this folders hierarchy installed on your disk when installed:

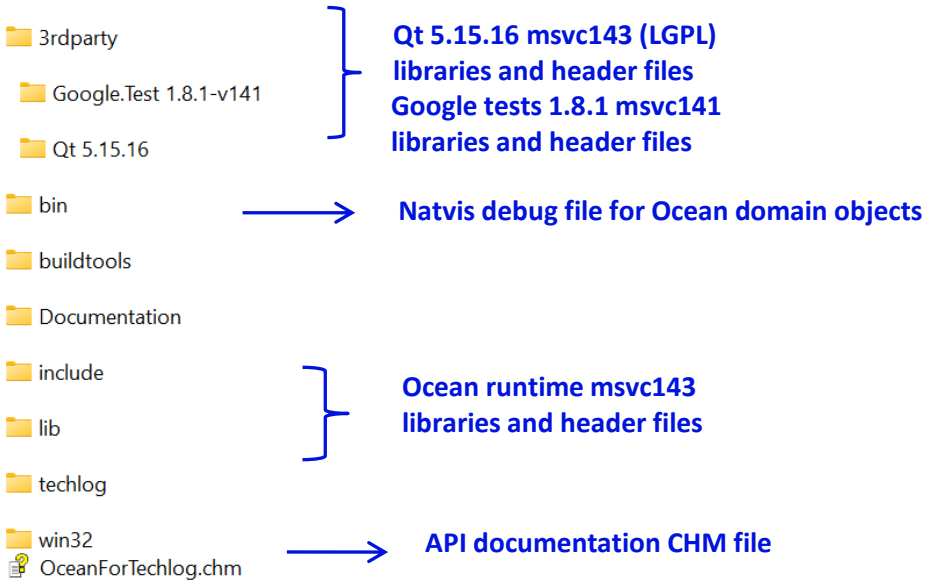


Figure 13: Ocean for Techlog package content

Plug-ins are built on Qt. The Ocean for Techlog Framework installer comes with the Qt LGPL 5.15.16 version. It contains QtCore and QtGui libraries (the 2 most basic Qt libraries). Those libraries are compatible with the Visual Studio compiler v143 (Visual Studio 2022).
The Ocean for Techlog API exposes these objects:

- Base classes: **QObject** (plug-in classes are **QObject** and in particular they expose their event handlers as Qt's slots methods), **QWidget** (a simple way of providing a custom GUI is by implementing a **QWidget**)
- Basic types: QString, QVariant, QImage, QColor, etc.
- Containers: **QList**, **QMap**, **QHash**, etc.
- Enums: **Qt::PenStyle**, etc.

Google.Test 1.8.1 x64 libraries are provided with Ocean framework in order to create an Ocean test plug-in. See the "Create unit tests for your plug-in" section for more information on how to create Google tests for an Ocean plug-in.
The Ocean framework libraries are built with Visual Studio compiler v143 with which the plug-in links to build with the v143 compiler.

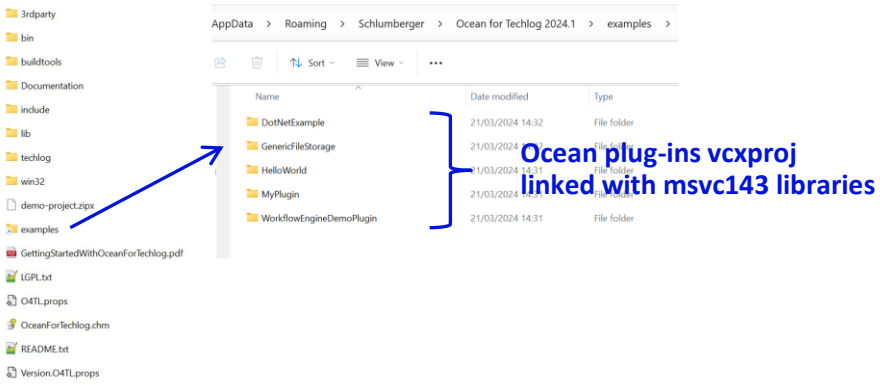


Figure 14: Ocean plug-in examples

The Ocean framework installer deploys in the Ocean package home folder an example folder link. The "examples" points to Visual Studio plug-in projects that link with Ocean libraries and use the Visual Studio compiler v143. All these examples are deployed in the **%appdata%\Schlumberger\Ocean for Techlog 2024.4** directory.

The examples folder includes these plug-ins:

1. **HelloWorld**: a simple plug-in useful to test your Ocean for Techlog development environment.
2. **DotNetExample**: shows how to integrate a .NET library in Techlog using Qt and Ocean framework.
3. **MyPlugin**: some code examples of each API exposed in Ocean for Techlog
    - Read and write data access
    - Create a workstep, add it and run it in a Techlog workflow
    - Plot examples as Logview, cross-plots, custom plots
    - Custom UI examples

4. **GenericFileStorage**: some plug-in domain objects (custom domain objects) code samples

The plug-in examples listed previously are built in release mode and deployed by the Ocean framework installer into the **Extensions** folder of Techlog User folder. You can modify the path to the Techlog **User** folder through the **TLUSERDIR** environment variable or directly in Techlog through the **Options** window dialog.

This is described in the "Ocean for Techlog environment variables" section.
The same known **Extensions** location can be added within Techlog's multi-level folder organization: **Techlog** and **Company**. This allows the plug-in to be deployed along with the Techlog installation, or on the Company's shared drive to reach many users.


It is not recommended for a plug-in developer to deploy a plug-in directly at the company or Techlog level for these reasons:

- content of the Company folder is usually handled by a dedicated team within the company
- Techlog extensions folder hosts plug-ins deployed with the Techlog baseline as native Techlog modules

Ocean for Techlog environment variables

In order to build your plug-ins, the Ocean installer sets the TechlogSDKHome2024_4 environment variable which is the root folder path where the Ocean for Techlog framework is installer on your disk (e.g. **C:\Program Files\Schlumberger\Ocean for Techlog 2024.4**).

To see the demo plug-ins installed with the Ocean package in the Techlog module manager, the user folder parameter is needed to say where deployed the plug-ins are.

If there is no user folder set on your machine, the installer sets the **%AppData%\Schlumberger\Ocean for Techlog 2024.4\techlog** folder as user folder and deploys the sample plug-ins in this folder. Sample plug-in code is also deployed in the **%AppData%\Schlumberger\Ocean for Techlog 2024.4\examples**

You can change it anytime through Techlog **Options** window (**Navigation pane > Options > Folders**).

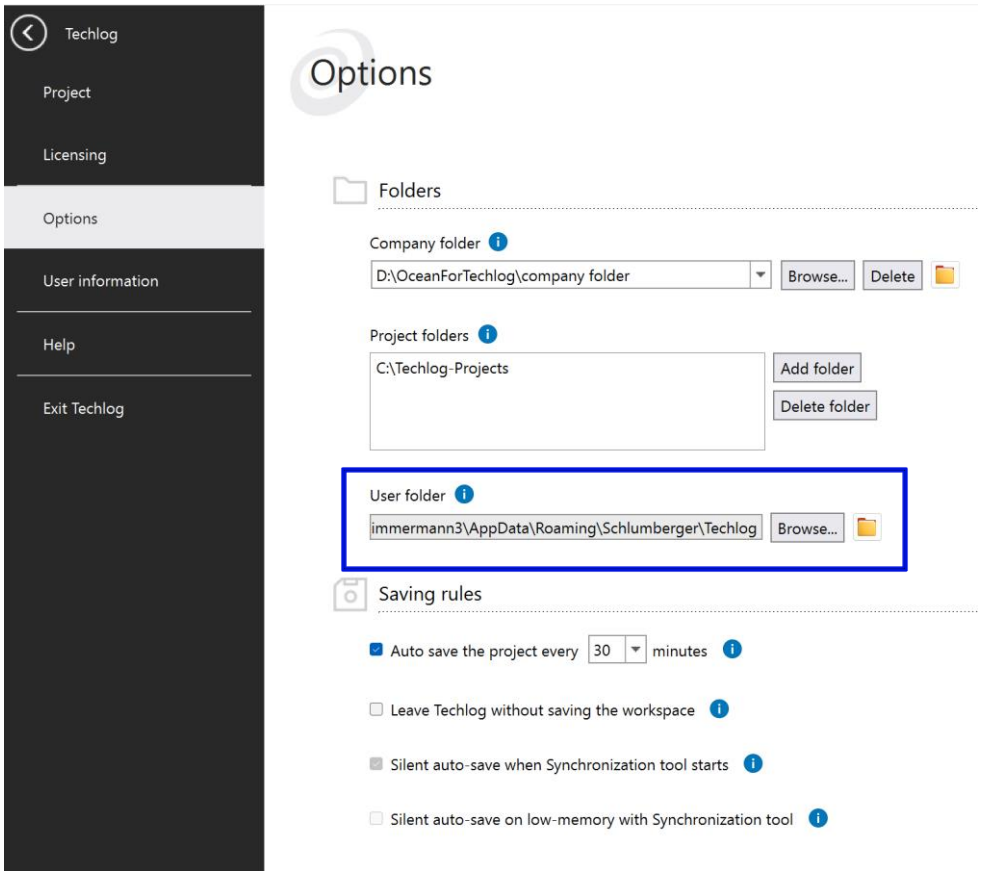


Figure 15: Techlog user folder

This parameter is set through the **TLUSERDIR** environment variable:

- **TLUSERDIR**=%TechlogSDKHome%\techlog if you want to use the User folder as your target build area.

Close and re-open any explorer window to propagate the new environment variable settings.

Ocean for Techlog property sheets

Property sheets (.props files) are deployed with the Ocean framework and are used to setup Ocean and Qt property values needed by the plug-in project to be built in Visual Studio.

Two main property sheets deployed at the **TechlogSDKHome2024_4** root folder:

Version.O4TL.props defines:

- The Techlog version (2024.4) used by all plug-ins built with this Ocean framework package.
- The path to **QTDIR** folder that contains the version of Qt libraries shipped with the Ocean for Techlog framework.
- The path to **GTests** folder that contains the version of GTests libraries shipped with the Ocean for Techlog framework.

O4TL.props includes other property sheets as:

- **AdditionalDependencies.O4TL.props** and **Include.O4TL.props** deployed in **TechlogSDKHome2024_4\bin** folder and used to defines Ocean for Techlog libraries and header files in the Visual Studio plug-in project.
- **AdditionalDependencies.QT.props** and **Include.QT.props** deployed in **TechlogSDKHome2024_4\3rdparty\Qt 5.15.16** folder and used to defines Qt libraries and header files in the Visual Studio plug-in project.
- **AdditionalDependencies.GTests.props** and **Include.GTests.props** deployed in **TechlogSDKHome2024_4\3rdparty\Google.Test 1.8.1-v143** folder and used to defines Google Tests libraries and header files in the Visual Studio test plug-in project.

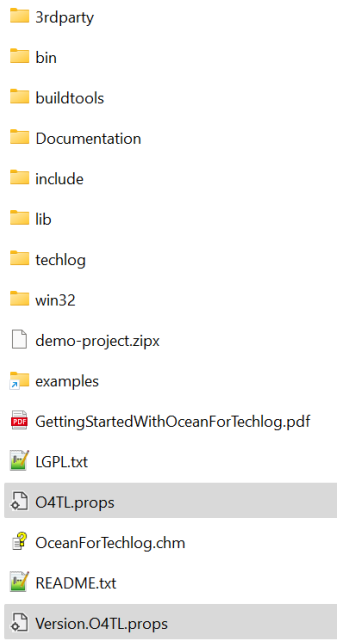


Figure 16: Ocean for Techlog main property sheets

Note: The main Ocean for Techlog property sheets are included to the Visual Studio plug-in project template and wizard installed with the Ocean framework to Visual Studio. Please don't modify those files as they contain the needed Ocean and Qt property values needed to build a valid plug-in for Techlog 2024.4 release. See the "Creating the Plug-in and Activity with Visual Studio" section for more information on how to create a plug-in by template in Visual Studio.

Test the Ocean for Techlog development environment

First test if **TechlogSDKHome** is properly set up and the Techlog user folder is pointing to the **Extensions** folder of the Ocean for Techlog development package. Perform these steps:

1. Run Techlog and click on the **Open the module manager** button from the **Navigation pane > Licensing** menu:

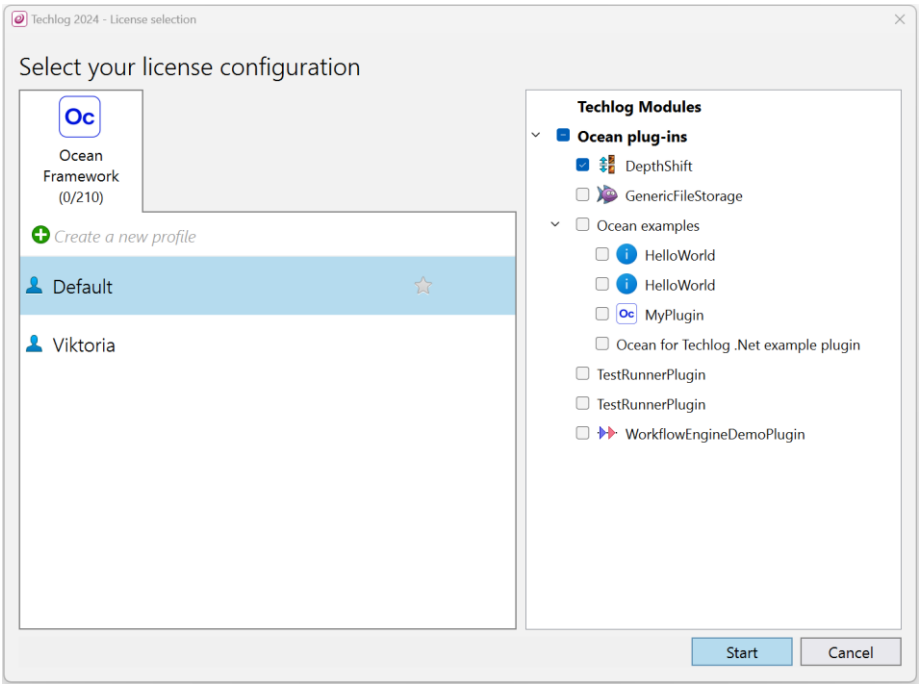


Figure 17: Release plug-ins deployed with Ocean Framework

The module manager scans the **Extensions** folder of the Ocean for Techlog package and the example plug-ins built in release mode and shipped with the Ocean framework are displayed as in Figure 17.

1. Go to %TechlogSDKHome2024_4%\examples\HelloWorld folder.

2. **Open HelloWorld.vcxproj** with Visual Studio and build the project in debug x64 mode.

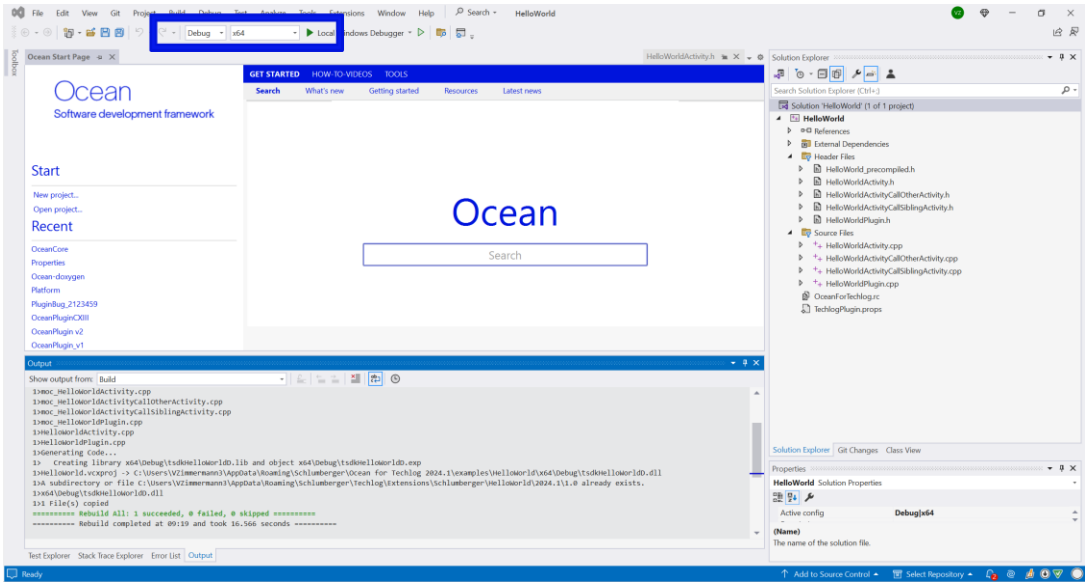


Figure 18: HelloWorld build in debug x64

The project must build successfully and a new debug x64 library of the HelloWorld project is generated in the **Extensions** folder of the Techlog user folder.

Note: In this screenshot you can see that the expected plug-in structure folder is **VendorName/PluginName/TechlogVersion/PluginVersion/**. If this structure folder is not respected the plug-in is not loaded in Techlog.

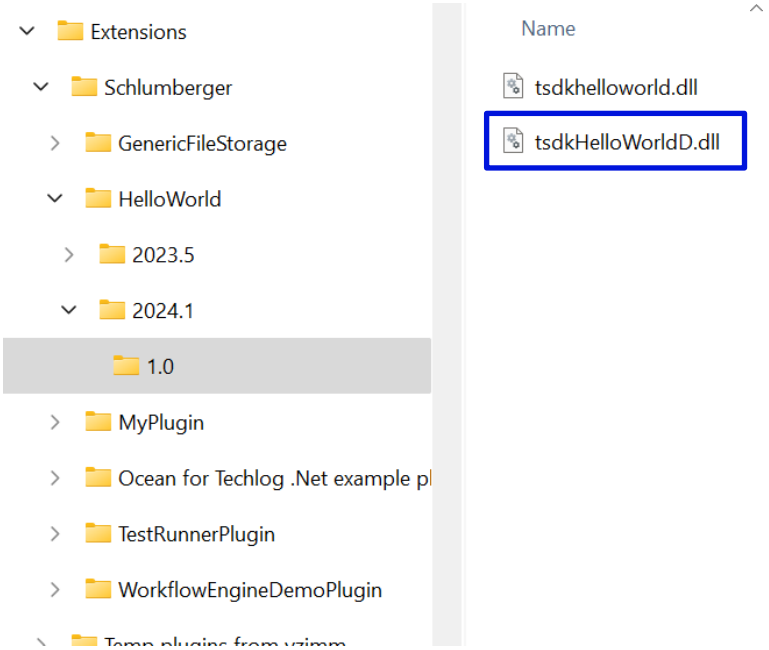


Figure 19: HelloWorld debug x64 library

3. In Techlog, open the module manager, right-click on **Ocean plug-ins** and click Refresh plug-ins in the context menu. The new **HelloWorld debug x64** plug-in appears in the **Ocean plug-ins** group as in Figure 20. Since there is one HelloWorld plug-in dll deployed by the Ocean framework installer in the same plug-in folder (HelloWorld release plug-in v143), once the HelloWorld plug-in is built in debug v143 and deployed, you see two HelloWorld plug-ins in the module manager with the same name "HelloWorld". Next to the plug-in you can click on the information icon and check the corresponding plug-in dll name in the information pane of the module manager.

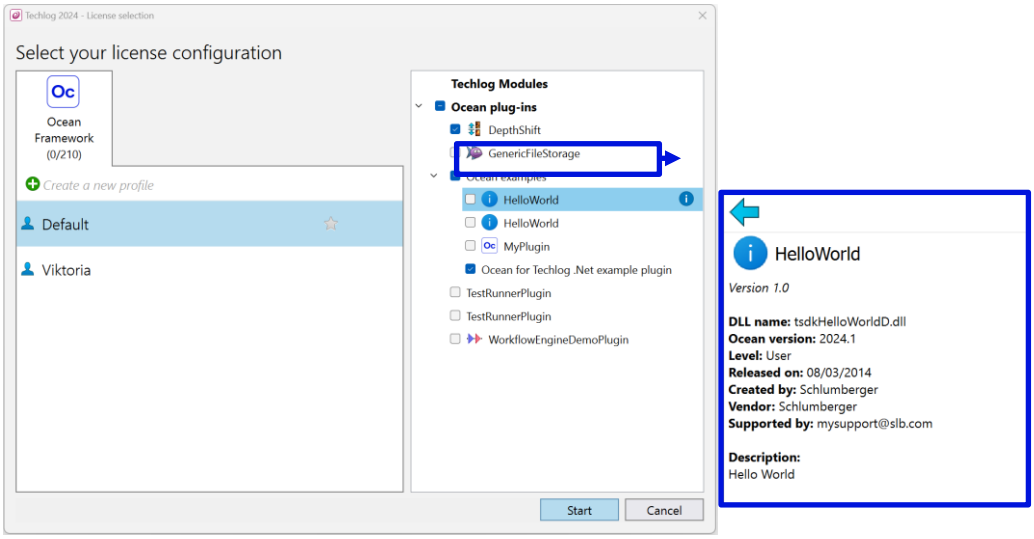


Figure 20: HelloWorld64D plug-in

After you refresh the list of plug-ins in the module manager, if you get these error messages for some plug-ins built in debug mode, it means that the Techlog plug-in debug host process executable and its dependencies have not been deployed properly in **bin64/pluginhost** folder of Techlog installation folder. Please re-install the Ocean framework.

Error: Plugin 'tsdkmypluginD.dll': can't find corresponding plugin host file.
Error: Can't launch plugin host for plugin 'tsdkmypluginD.dll': host process not running.

4. Enable the plug-in and click the Hello World action menu in the new HelloWorld plug-in added in Techlog. "Hello Plugin World!" is displayed in the Techlog **Output** console.

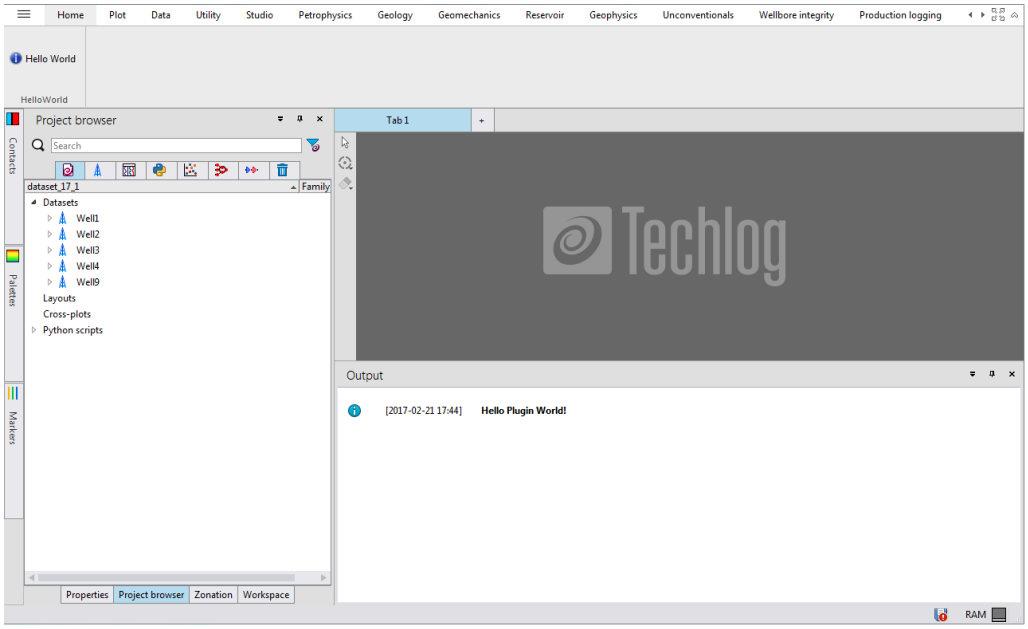


Figure 21: HelloWorldD activity running

3.Writing your first plug-in

The Ocean for Techlog framework provides a development and runtime environment for wellbore centric data manipulation, interpretation, and visualization applications. You have the ability to create workflows that interoperate with or extend the commercial Techlog Interactive Suite and the capability to extend the scope of Techlog to address new petrotechnical domains. This chapter describes the procedure of creating a simple plug-in.

Writing the plug-in

In your first plug-in you will add a new menu item into a new tab and group in Techlog. Clicking on this menu item will trigger an activity that prints all the well, dataset and variable names found in the current project.

There are three main steps for creating your first plug-in. Each step will be detailed in the sections that follow. The steps are:

1. Run the Ocean for Techlog Plug-in Wizard in Visual Studio to create the plug-in.
2. Inspect the files created by the Wizard.
3. Modify the code to add the processing logic.

Creating the Plug-in and Activity with Visual Studio

To create the project, plug-in, and activity using Visual Studio:

1. Start Visual Studio.

2. Create a new project by selecting File > New Project.
3. In the Project types area, under Visual C++ project type, select Ocean > Techlog 2024.4.
4. Select the Ocean Plug-in template.
5. Provide the name "MyFirstPlugin" for the project.
6. Click OK to start the Wizard.

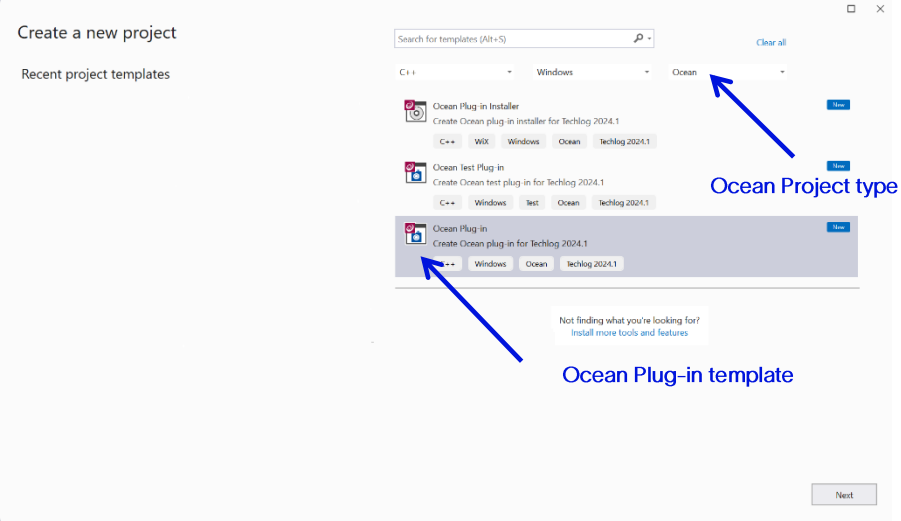


Figure 22: New project window

It is generally a good practice to use a descriptive plug-in name.

7. Change the name of your plug-in to "MyFirstPlugin".
8. Change the "Vendor name", "Plug-in version", "Support e-mail", "Crash dump e-mail" and "Description" fields as appropriate (See Figure 23).
9. Note that "Vendor name", "Plug-in name" and "Plug-in version" are mandatory plug-in information.
10. Click Finish.

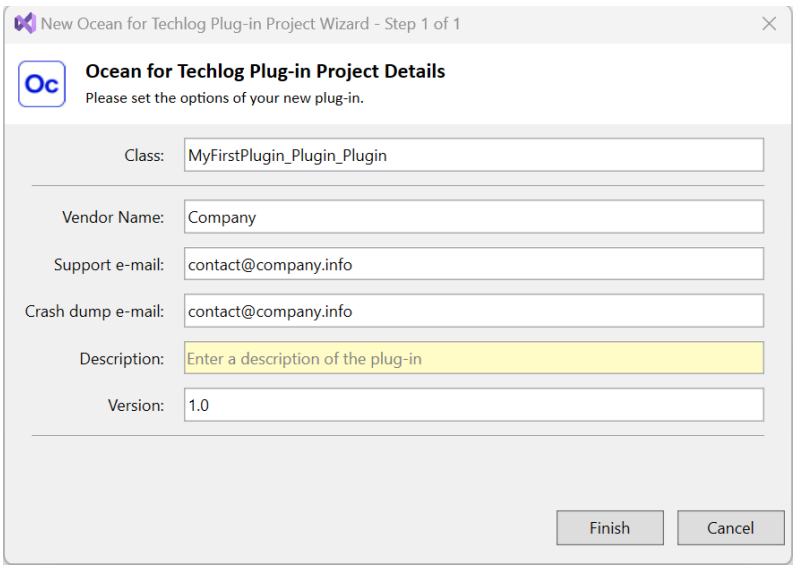


Figure 23: Plug-in wizard

The wizard creates the project with the main plug-in class.

1. Add a new plug-in activity by right-clicking on the project in the Solution Explorer and selecting **Add > New Item** in the context menu.
2. In the Item types area, under **Visual C++** item type, select **Ocean > Techlog 2024.4**.
3. Select the **Ocean Activity** template.
4. Provide the name "ReadDataActivity" for the activity.
5. Click **Add** in the dialog (See Figure 24.)

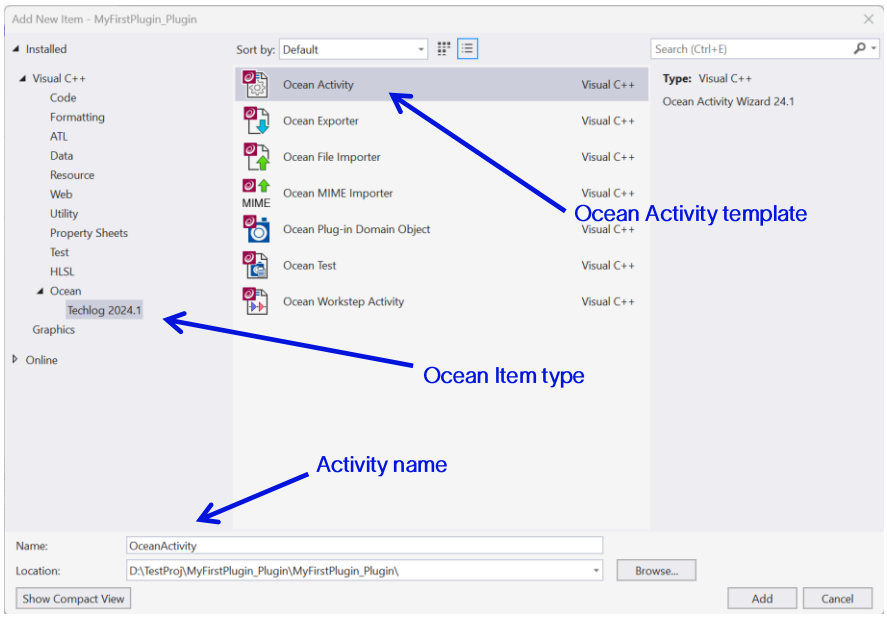


Figure 24: New activity window

Note: You also have the ability to create:

- An **Ocean Exporter**. This adds to the project the skeleton code of a plug-in exporter. See the "Exporter implementation" section in Ocean for Techlog Developer Guide - Plugin Domain Object - Importer&Exporter for more information on how to implement a plug-in domain object with Ocean.

- An **Ocean File Importer**. This adds to the project the skeleton code of a plug-in file importer. See the "FileImporter implementation" section in Ocean for Techlog Developer Guide - Plugin Domain Object - Importer&Exporter for more information on how to implement a plug-in domain object with Ocean.

- An **Ocean MIME Importer**. This adds to the project the skeleton code of a plug-in MIME importer. See the "MimeImporter implementation" section in Ocean for Techlog Developer Guide - Plugin Domain Object - Importer&Exporter for more information on how to implement a plug-in domain object with Ocean.

- An **Ocean Plug-in Domain Object**. This adds to the project the skeleton code of a plug-in domain object. See the "Plug-in domain object" section in Ocean for Techlog Developer Guide - Plugin Domain Object - Importer&Exporter for more information on how to implement a plug-in domain object with Ocean.

- An **Ocean Workstep Activity**. This adds to the project an activity class that instantiates a Workstep in the Techlog Application Workflow Interface with its signals and slots. See the "Workflow and worksteps" section in *Ocean for Techlog Developer Guide – Basics* for more information on how to implement an Ocean workstep.

6. Change the "Tab title", "Group title" and optionally the "SubGroup title" fields as appropriate (See Figure 25) and click Next. These fields are used to create the plug-in menu tab, group and sub-group in the Techlog toolbar that holds the Ocean menu action.

Note: The wizard allows you to add the plug-in activity to a tab, a group and optionally a sub-group that are already declared in the plug-in code. Existing tabs, groups and sub-groups are listed into the corresponding tab, group and sub-group drop down lists.

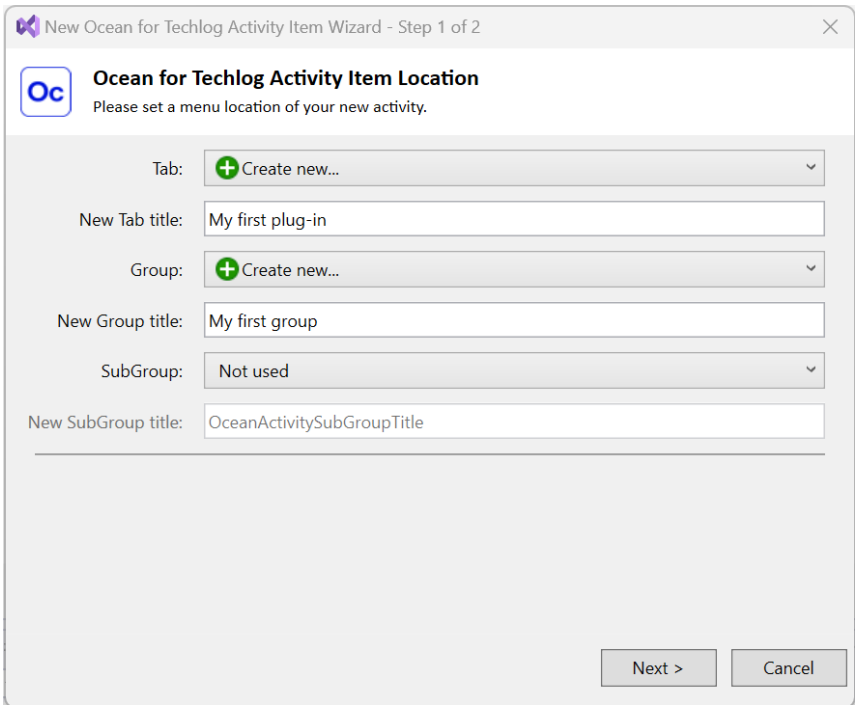


Figure 25: New activity wizard: menu tab and group creation

7. Choose the type of action that you want to create. See the "Tab, group and action created by the plug-in" section in Ocean for Techlog Developer Guide – Basics for more information on how to create a menu action. Change the "Action menu text" and "Action menu tooltip" fields as appropriate (See Figure 26) and click **Finish**. These fields are used to create the plug-in menu action in the Techlog toolbar that triggers the Ocean activity.

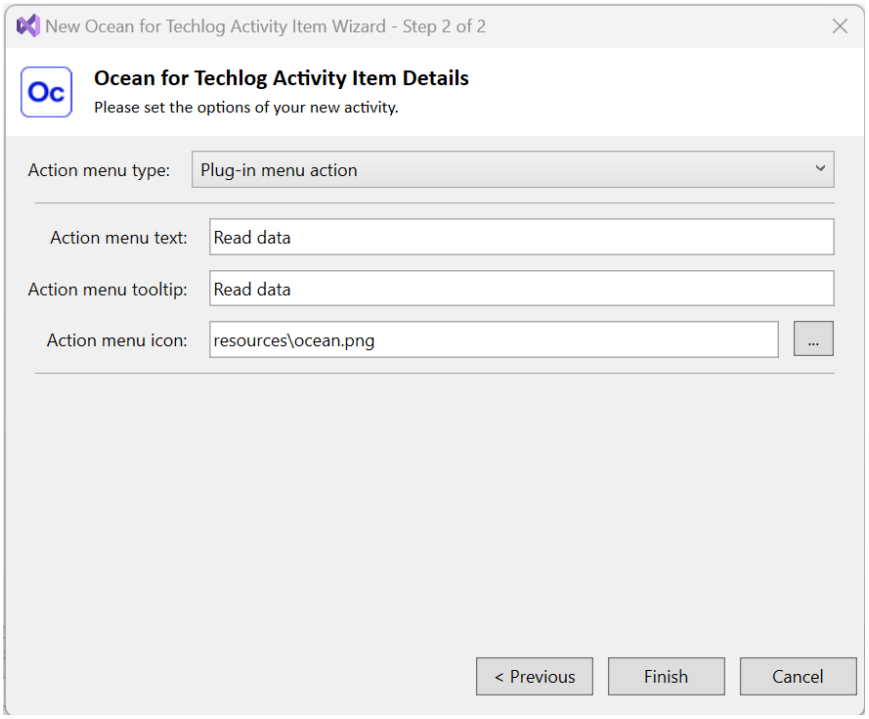


Figure 26: New activity wizard: plug-in menu action creation

The wizard adds the activity class to the project.

⚠ If Intellisense is disabled in Visual Studio, Ocean template items are not accessible and an error message is raised. In Tools > Options menu of Visual Studio, Disable database has to be turned off.

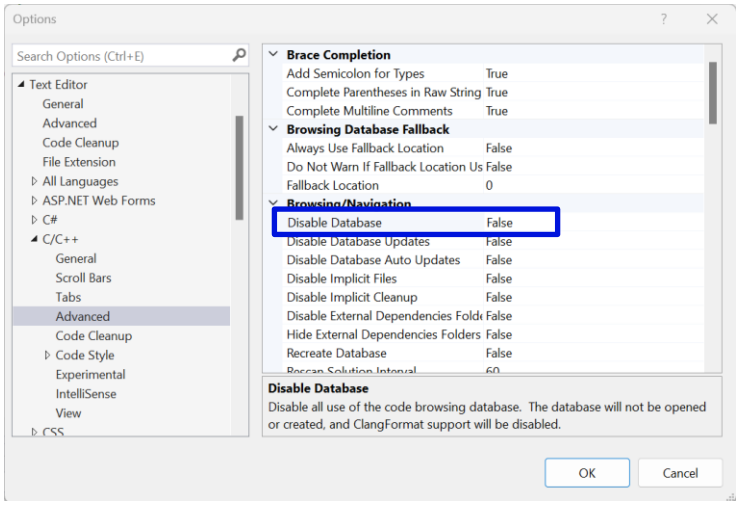


Figure 27: Disable database

Inspecting the files

The Ocean for Techlog Wizard creates a solution named "MyFirstPlugin" with a project named "MyFirstPlugin" in the Visual Studio Solution Explorer. The project will contain header and source file for the **Plugin** class that was created, and the **Activity** class (See Figure 28).

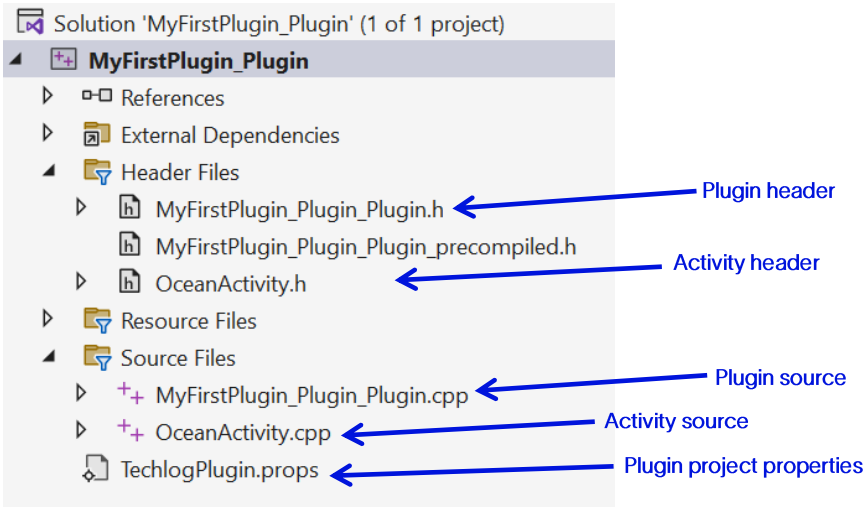


Figure 28: Example project header and source files in Solution Explorer

Plugin

The main plug-in class derives from **PluginIdentity** interface class. **PluginIdentity** is derived from **IPlugin** class (plug-in interface) that exposes these virtual methods:

```
class IPlugin
{
public:
  virtual void getInformation(PluginInformation
  &pluginInformation) const = 0;

  virtual void getActivities(PluginActivities
  &activities) const = 0;

  virtual void getMenu(PluginMenu &menu) const = 0;
};
```

Implement the plug-in identity interface to declare:

- Information about the plug-in (**getInformation**)
- A list of activities (**getActivities**)
- Menu items used to trigger those activities (**getMenu**)

```
#pragma once
#include "tsdkpluginidentity.h"

class MyFirstPlugin : public PluginIdentity
{
  Q_OBJECT
  Q_PLUGIN_METADATA( IID TSDK_PLUGIN_INTERFACE_ID )
public:
  virtual void getInformation(Slb::Ocean::Techlog::PluginInformation&
  pluginInformation)
  const override;
  virtual void getActivities(Slb::Ocean::Techlog::PluginActivities&
  activities)
  const override;
  virtual void getMenu(Slb::Ocean::Techlog::PluginMenu& menu) const
  override;
};
```

These three methods must be implemented in the source file that first includes the plug-in and activity header files and Slb::Ocean::Techlog namespace at the beginning of **MyFirstPlugin.cpp** file.

```
#pragma once
#include "tsdkpluginidentity.h"

class MyFirstPlugin : public PluginIdentity
{
  Q_OBJECT
  Q_PLUGIN_METADATA( IID TSDK_PLUGIN_INTERFACE_ID )
public:
  virtual void getInformation(Slb::Ocean::Techlog::PluginInformation&
  pluginInformation)
  const override;
  virtual void getActivities(Slb::Ocean::Techlog::PluginActivities&
  activities)
  const override;
  virtual void getMenu(Slb::Ocean::Techlog::PluginMenu& menu) const
  override;
```

These three methods must be implemented in the source file that first includes the plug-in and activity header files and **Slb::Ocean::Techlog** namespace at the beginning of **MyFirstPlugin.cpp** file.

```
#include "tsdkplugininformation.h"
#include "tsdkpluginactivities.h"
#include "tsdkpluginmenu.h"
#include "tsdkpluginmenutab.h"
#include "tsdkpluginmenuaction.h"
#include "tsdkpluginmenugroup.h"
#include "MyFirstPlugin.h"

// Please include here your activity header files
#include "ReadDataActivity.h"
// #include "Activity.h"
/*****ACTIVITIES*INCLUDE*****/

using namespace Slb::Ocean::Techlog;

/*****ACTIVITIES*INCLUDE*****/

using namespace Slb::Ocean::Techlog;
```

The **`getInformation`** method contains properties which provide information to the plug-in. These include vendor name, plug-in name, plug-in version, description, release date, plug-in icon, creator, support email, crash dump email, plug-in license feature and Techlog license feature dependencies. The contents of **`getInformation`** should look something like:

```
void MyFirstPlugin::getInformation(PluginInformation& pluginInformation) const
{
  pluginInformation.setVendorName(PLUGIN_VENDOR_NAME);
  pluginInformation.setName(PLUGIN_NAME);
  pluginInformation.setVersion(PLUGIN_VERSION);
  pluginInformation.setDescription("This is my first plug-in");
```

```
  pluginInformation.setReleaseDate("16/03/2024");
  pluginInformation.setIcon(QIcon("ocean.png"));
  pluginInformation.setCreator(PLUGIN_VENDOR_NAME);
  pluginInformation.setSupportEmail("jsmith@slb.com");
  pluginInformation.setCrashDumpEmail("jsmith@slb.com");
}
```

The plug-in property values as PLUGIN_VENDOR_NAME, PLUGIN_NAME and PLUGIN_VERSION are stored in the **TechlogPlugin.props** file. This property sheet contains Techlog plug-in properties stored at the Visual Studio project level. Those values can be changed directly editing the file.

Note: The plug-in vendor's name, name and version values passed to the **setVendorName**, **setName** and **setVersion** functions of the **`PluginInformation`** class have to match the plug-in structure folder names **VendorName/PluginName/TechlogVersion/PluginVersion/**. If this structure folder is not respected the plug-in is not loaded by the Techlog module manager.

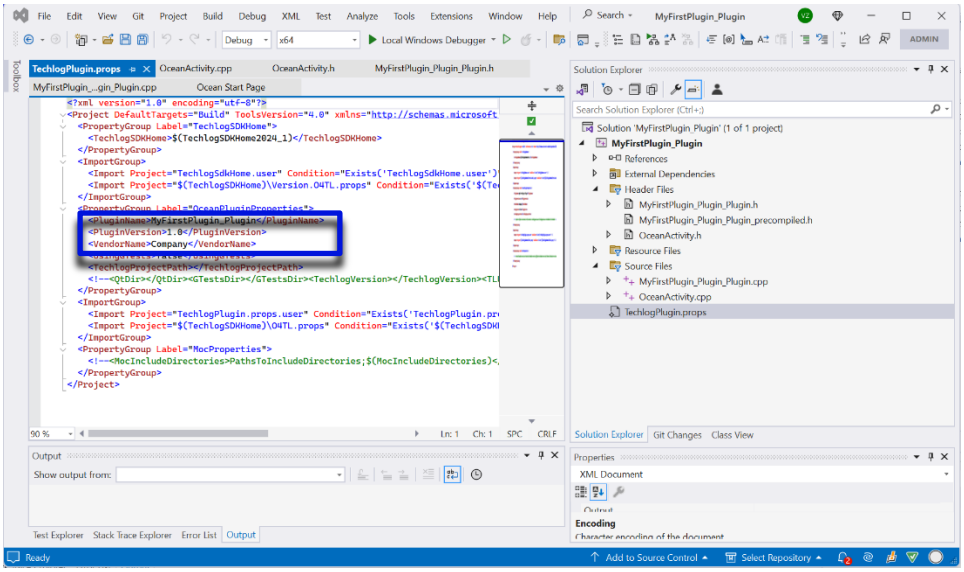


Figure 29: Ocean plug-in project properties

By right-clicking on the project in the Solution Explorer and selecting **Ocean plug-in properties** item in the context menu, Ocean plug-in project property values can be overridden at the Visual Studio user level through the **Ocean plug-in properties** dialog window.

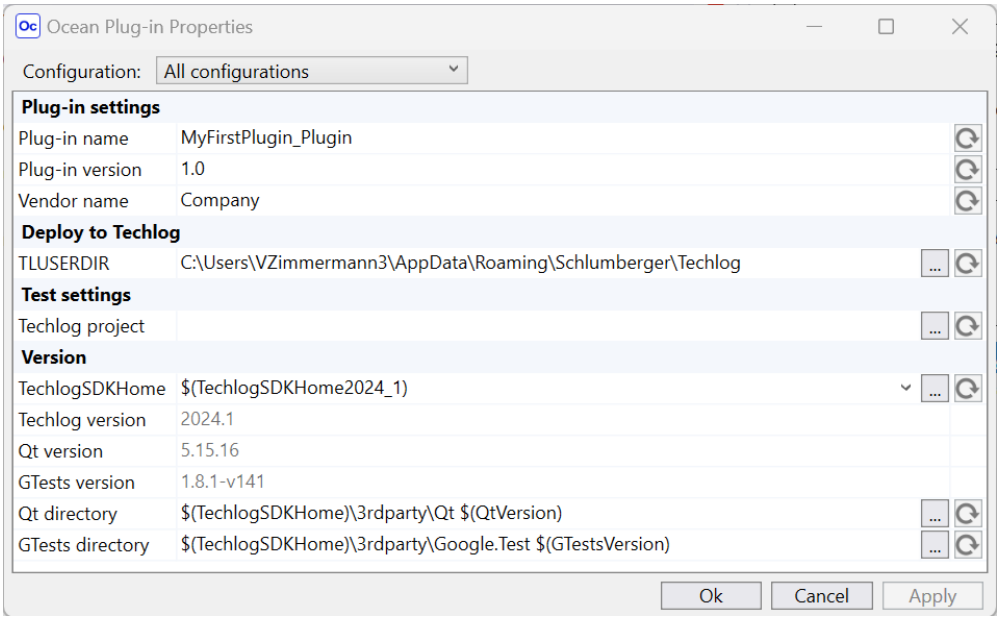


Figure 30: Ocean plug-in properties dialog window

Project property values modified through the **Ocean plug-in properties** dialog window doesn't change the default project property values stored in the **TechlogPlugin.props** file. A new property sheet named **TechlogPlugin.props.user** is created in the project directory that contains those new values.

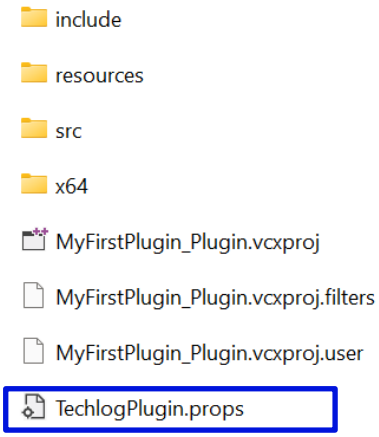


Figure 31: Ocean plug-in user property sheet

Property values stored in the **TechlogPlugin.props.user** file have the highest priority at the build time. Property values defined at the user level can be rollback to the project level property values using  buttons in the **Ocean plug-in properties** dialog window.

Note: The best practice is to share only the **TechlogPlugin.props** in a source control common to all members of a team that are collaborating on a plug-in project development.

In the **getActivities** method, **ReadDataActivity** is added to the plug-in activity. The wizard had declared for this activity a unique id (GUID) and **ReadDataActivity** is identified as unique by its GUID in the list of activities of the plug-in.

```
static QString ReadDataActivityId(QLatin1String("f1007f1e1ce3477ea47fd91f4e7e1b7b"));

void MyFirstPlugin::getActivities(PluginActivities& activities) const
{
// Please fill this method with your activities with lines like this:
  activities.add( TSDK_ACTIVITY( ReadDataActivity, ReadDataActivityId ) );
  // activities.add(TSDK_ACTIVITY(Activity, actionId));
  /*****ACTIVITIES*REGISTRATION*PLACE*****/
}
```

The wizard implements the **getMenu** method in **MyFirstPlugin.cpp**; this method is used to add custom menus to Techlog.

Menu items are used to trigger activities.

The sequence to customize the TBar (Ribbon) is summarized using the **PluginMenu** API exposed with Ocean:

1. **PluginMenuTab:** create new menu area for the plug-in
2. **PluginMenuGroup:** new menu group created and added to the new **PluginMenuTab** object
3. **PluginMenuAction:** new menu action created and added to the new **PluginMenuGroup** object and instantiated with an action id
4. **PluginMenu:** new **PluginMenuTab** object added the Techlog main menu

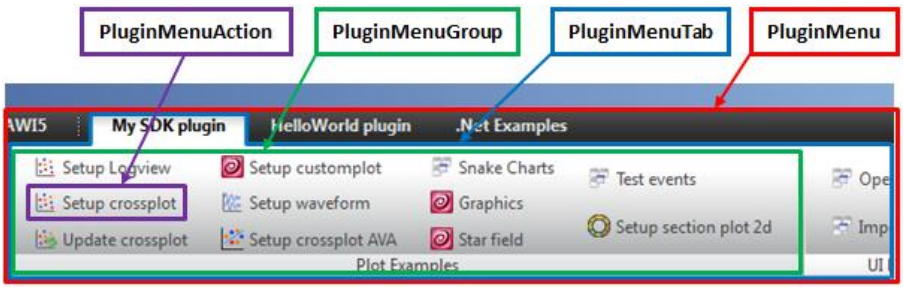


Figure 32: Plug-in menu classes

See the "Plug-in information and menu" section in *Ocean for Techlog Developer Guide – Basics* for more information on how to extend Techlog menus.

To link **ReadDataActivity** with the **PluginMenuAction** that triggers this activity, the wizard instantiates the **PluginMenuAction** object passing to the constructor the unique identifier (GUID) of the activity declared at the beginning of **MyFirstPlugin.cpp.**

```
void MyFirstPlugin::getMenu( PluginMenu& menu ) const
{
  PluginMenuTab tab ("63c99cc4e766412d84a7827e3b238a58");
```

```
  tab.setTitle("My first plug-in");

  PluginMenuGroup group ("e7e3b9553d64417283eb7b1083789e96");
  group.setTitle("My first group");

  PluginMenuAction actionReadData (ReadDataActivityId);
  actionReadData.setText("Read data");

  group.addAction(actionReadData);
  tab.addGroup(group);
  menu.addTab(tab);
}
```

Activity

This new action menu triggers the **ReadDataActivity**. This class inherits from the **AbstractActivity** interface class which is the base class for any Ocean for Techlog plug-in activity.

```
class AbstractActivity : QObject
{
public:
  virtual void run() = 0;
  virtual void dispose();
  ...
};
```

The **run** method is the main method of an activity, called when the user clicks on the corresponding menu item.

Override the **dispose** method if you need to cleanup resources before the activity is unloaded.

The **AbstractActivity** is a **QObject** so every activity declared in a plug-in is a **QObject,** but you need to add a **Q_OBJECT** macro in your activity class to tell the meta-object compiler to compile the signals and slots.

```
class ReadDataActivity : public Slb::Ocean::Techlog::AbstractActivity
{
  Q_OBJECT;

private:
  void run();
};
```

Writing the algorithm code

Once the skeleton of the plug-in has been created, you need to implement the plug-in logic that will be triggered when the user clicks on the action menu declared in the **getMenu** method of the plug-in identity class (main plug-in class).
You add the custom algorithm code overriding the **run** method of the **AbstractActivity** interface.

```
#include "ReadDataActivity.h"

using namespace Slb::Ocean::Techlog;

void ReadDataActivity::run()
{
  // TODO: Implement the action menu logic here.
}
```

To write the algorithm code:

Access the APIs from the **Slb::Ocean::Techlog** namespace.

Code the **run** method. The work for the activity is completed:

Read the current main project using the **Session::current().mainProject()** API. The **Project** class exposes a function **wells**, which provides navigation to the well collections in the project. Parse through all the wells and for each well parse through all the datasets using the **datasets** public function exposed in the **Well** class.

Get for each dataset from the corresponding properties exposed in the **Dataset** class:

- Its name
- Its size; use the rowCount public method which returns the number of rows of the dataset (and therefore of all its variables)

Print the well name, dataset name and size from the main Techlog project using **Session::current().currentWorkspace()** API. The **Workspace** class exposes the **logEvent** method to print message into Techlog output console with some different output levels listed in **LogLevel** enumeration class:

- **Debug**
- **Information**
- **Warning**
- **Error**

For each dataset parse through all its variables using the **variables** public function exposed in the **Dataset** class.

Get for each variable from the corresponding properties exposed in the **`Variable`** class:

- Its **name**
- Its **unit**
- Its **family**

And print its property values in the Techlog Output console using the **logEvent** method of the current **Workspace** with a **LogLevel** set to **Information**.

Call **`stop`** method inherited from **AbtractActivity** interface class at the end of your activity **`run`** method to stop the plug-in activity. Otherwise, the plug-in will stay in the background until the user manually stops the plug-in task in the workspace manager of Techlog (Figure 33) or stops Techlog.

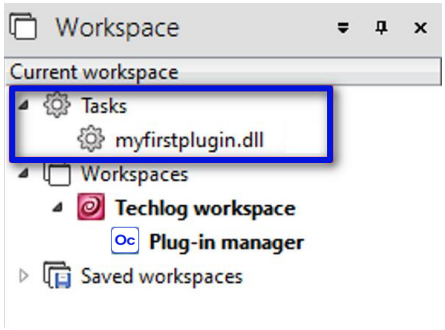


Figure 33: Techlog workspace manager

This example shows the complete activity source code:

```
#include "ReadDataActivity.h"

#include "tsdklock.h"
#include "tsdkloglevel.h"
#include "tsdkvariableenums.h"

using namespace Slb::Ocean::Techlog;

void ReadDataActivity::run()
{
  // TODO: Implement the action menu logic here.

  // Lock all
  Lock lock = LOCK_CREATE_AND_ACQUIRE_ALL(lock);

  // Get the current workspace from the current session
  Workspace workspace = Session::current().currentWorkspace();
  // Get the main project from the current session
  Project proj = Session::current().mainProject();

  // Iterate on all the wells in the project
  foreach (Well well, proj.wells())
  {
    // iterate on all the datasets of the current well in the loop
    foreach(Dataset dataset, well.datasets())
    {
      // Get the name and size of the current dataset in the loop
      QString datasetName = dataset.name();
```

```
      QString datasetSize = QString::number(dataset.rowCount());
      // Display well name and dataset infos in Techlog output console
      workspace.logEvent(LogLevelInformation,
      QString("<b>Well name = %1, Dataset name = %2, Dataset size = %3</b>")
     .arg(well.name()).arg(datasetName).arg(datasetSize));

     // iterate on all the variables of the current dataset in the loop
     foreach(Variable var, dataset.variables())
     {
       // Get the name, unit and family of the current variable in the loop
       QString varName = var.name();
       QString varUnit = var.unit();
       QString varFamily = var.family();
       // Display variable infos in Techlog output console
       workspace.logEvent(LogLevelInformation,
       QString("Variable name = %1,Variable unit = %2,Variable family = %3")
       .arg(varName).arg(varUnit).arg(varFamily));
     }
    }
   }
   // release objects locked
   lock.release();

   // Stop the plug-in activity
   stop();
}
```

Running the plug-in

You have just completed the modification of the **`run`** method. In this section, you will finish building the solution and run your plug-in in Techlog.

Build your solution in Visual Studio in release 64 bit. This creates a new folder for the plug-in in the deployment folder (**Extensions** folder) of the Ocean framework. This plug-in folder contains the new plug-in library. When it starts the module manager scans the **Extensions** folder and shows the new library in the list of available plug-ins. The plug-in menu is added to Techlog when the plug-in is enabled in the module manager. The activity runs as a separate process when the user clicks on the action menu; at this moment the plug-in appears as a new task in the list of tasks of the current workspace of Techlog.

Open the Techlog module manager and enable **MyFirstPlugin. My first plug-in** tab is added to the Techlog native tabs. This tab contains only one group **My first group** and this group contains only one action menu **Read Data** (Figure 34).

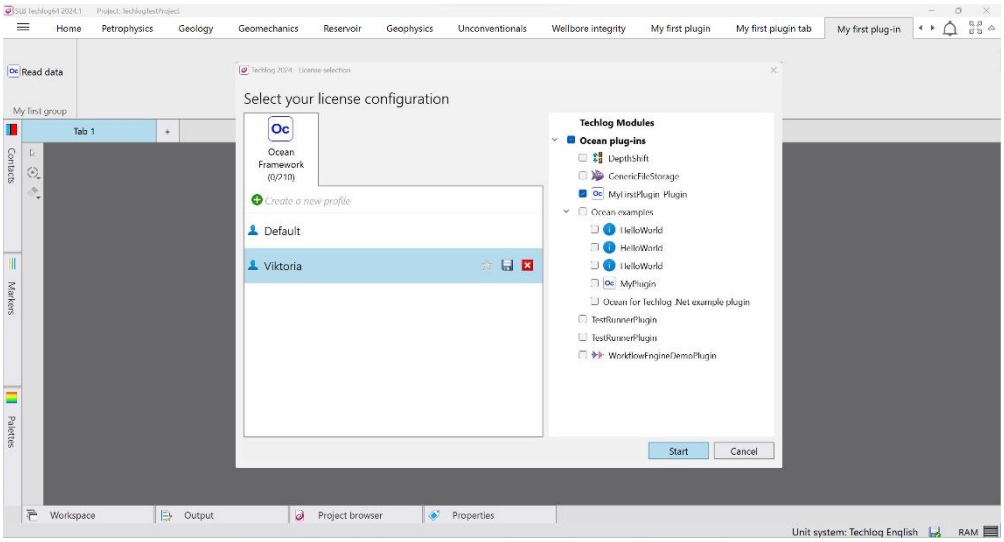


Figure 34: Enable MyFirstPlugin in the module manager

Import the Techlog fundamentals dataset deployed with the Ocean framework (*%TechlogSDKHome%\demo-project*) and click on the **Read Data** action menu. The **Read Data** activity shows all the wells, datasets and variables in the Techlog message log (Figure 35).

Note: Opening a Techlog project with an Ocean framework license will taint the project.

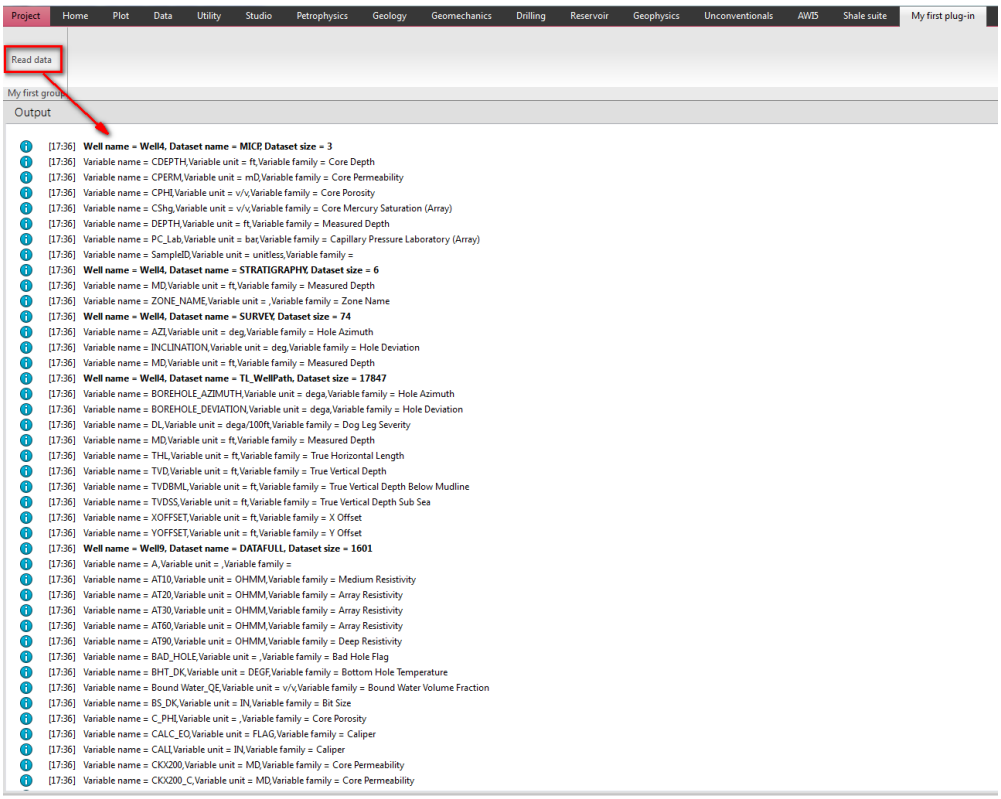


Figure 35: Read Data activity output messages

You have now written, built, and run your first Ocean for Techlog plug-in.

Debug the plug-in

To debug the plug-in you have to build it in debug mode. Go to the Visual Studio solution and change the build mode from release x64 to debug x64. Still in Visual Studio open the **ReadDataActivity.cpp** file and into the **run** method of the activity add a breakpoint on the first line.

Re-build the solution, close and reopen Techlog.

A new library called **MyFirstPlugin.dll** is generated in the plug-in folder. Then go back to Techlog, open the module manager and refresh the plug-in list (right click on **Ocean plug-ins**). The new plug-in for debugging appears in the module manager in the **Ocean plug-ins**. Disable the release version and enable the debug one.

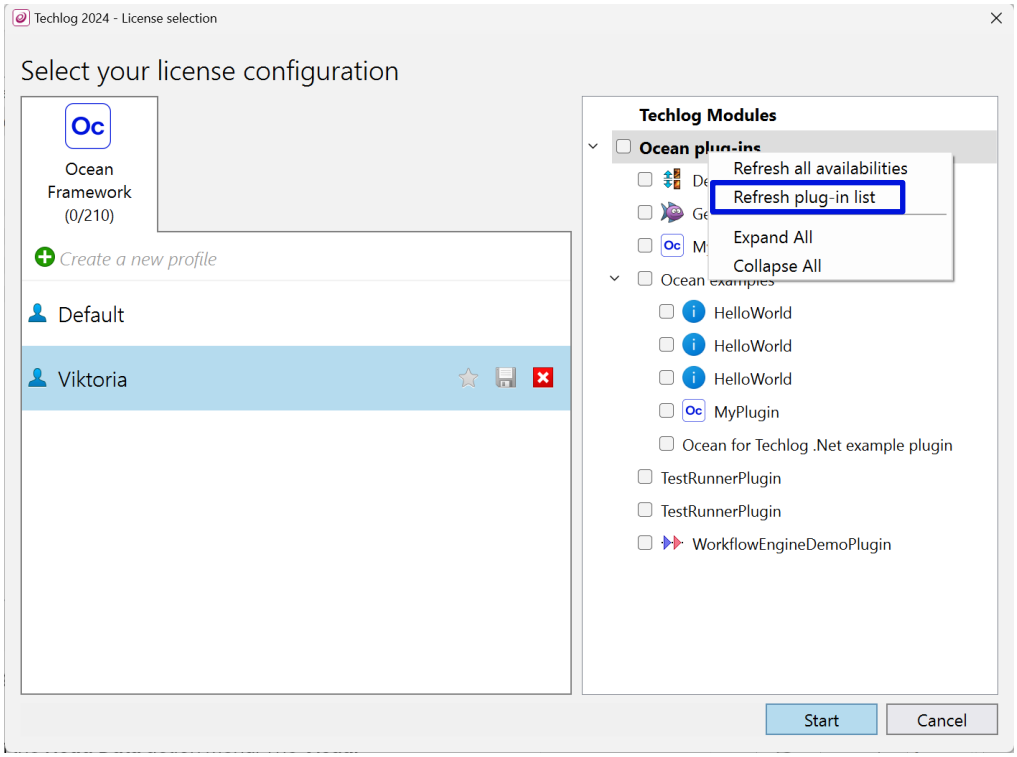


Figure 36: Refresh plug-in list

Press the **Ctrl+Alt** keys and click on the **Read Data** action menu. The **Visual Studio Just-In-Time debugger** pops up and asks you to select from the list a Visual Studio solution debugger to attach to the plug-in host, which for a plug-in built in 64 bit is **techlogpluginhostD.exe**. Select **MyFirstPlugin** in the list and click **Yes** as shown in Figure 37.

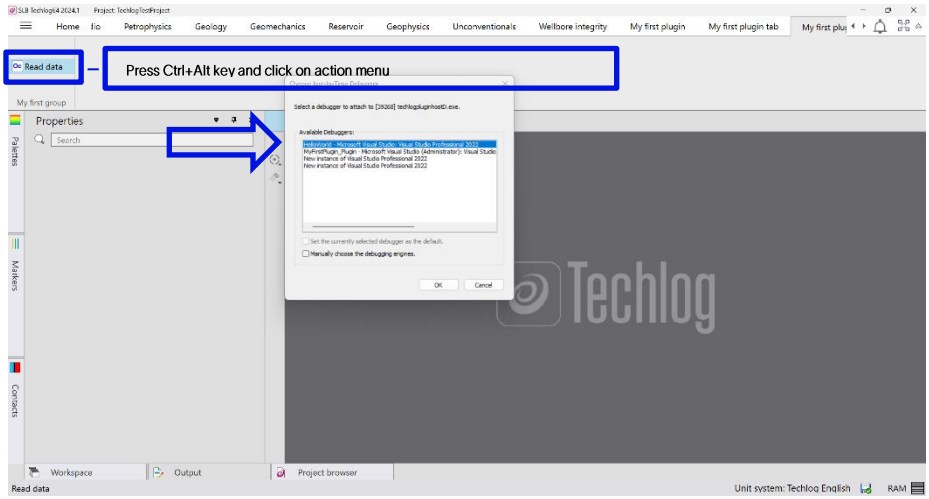


Figure 37: Debug the plug-in

The debugger stops on the first line of the **`run`** activity method where the breakpoint has been added.
If Visual Studio complains about a **Managed** application please **Manually choose the debugging engines** turning on this option in the **Visual Studio Just-In-Time debugger** window. A popup shows up listing all the available debugger engines, enable the **Managed** debugger for which version of the .NET framework you want to debug. Unless you're debugging a .NET based plug-in, you cannot attach the .NET/Managed debugger at all.

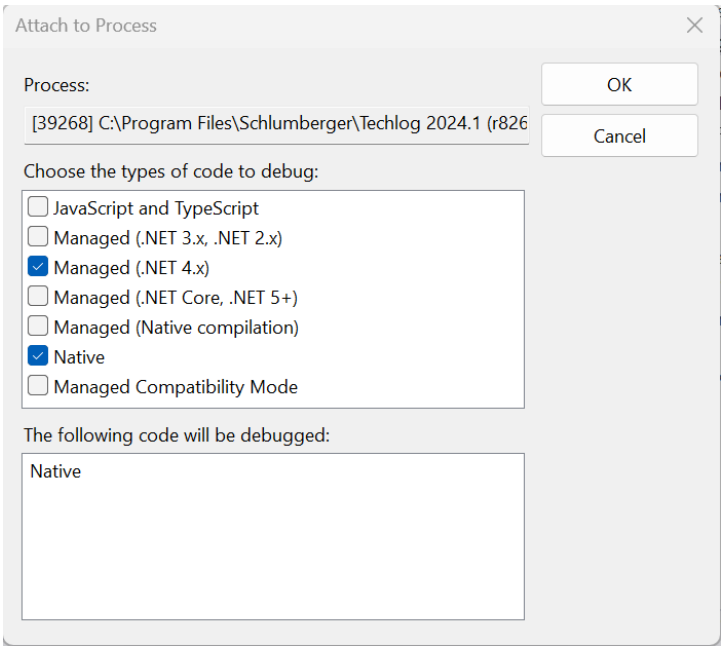


Figure 38:Visual Studio debugger engines

Auto enabled the plug-in

Enable the plug-in at Techlog start-up by adding a file named **auto_enabled** (no extension) into the plug-in folder that contains the plug-in dll.

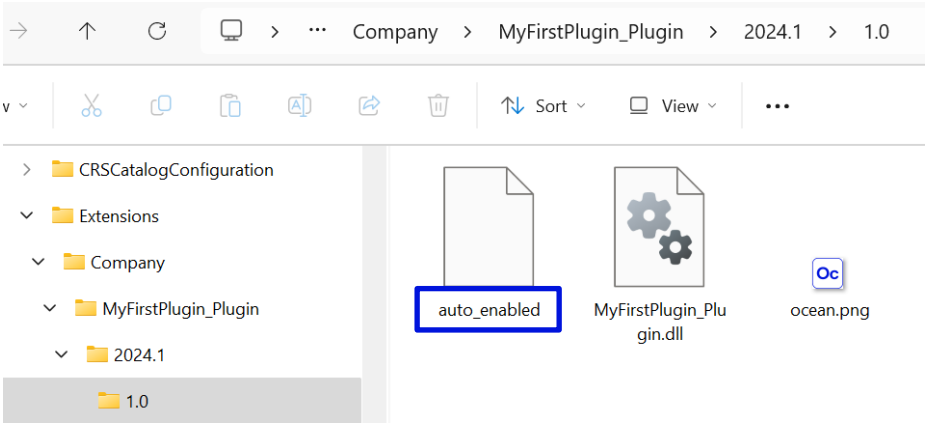


Figure 39: auto_enabled file

This file is a flag that tells Techlog to enable the plug-in in the Techlog module manager. The auto enabled plug-in no longer appears in the module manager under the **Ocean plug-ins** node and all the plug-in menus are added automatically to the Techlog ribbon when Techlog starts.

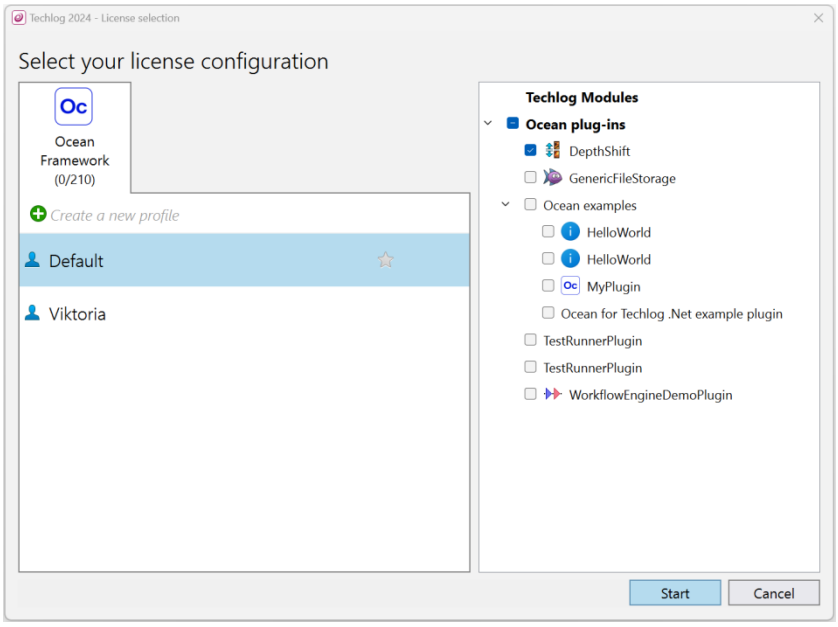


Figure 40: auto enabled plug-in not visible in module manager

Note: The Techlog plug-in host process on which the auto enabled plug-in runs doesn't appear anymore in the list of processes of the Windows task manager.

Auto start plug-in activities

A file named **plugin_config.json** deployed to the plug-in folder allows to run a list of plug-in activities at the Techlog startup .

The JSON file contains two parameters:

- **AutoStartActivies** that allows to pass the list of activity ids to be run
- **auto_enabled** that tells Techlog to enable the plug-in at the Techlog startup

```
{
    "AutoStartActivities" : ["vshgr","workstep by zone"],
    "auto_enabled" : "true"
}
```

Techlog Viewer plug-in development (SLB Internal only)

The Techlog Viewer is software to facilitate display and interaction with data.

Techlog Viewer specific

To allow your plug-in to run on Techlog Viewer, call the **setTechlogViewerActivity** function in the **PluginInformation** class, with the activity ID as the parameter.

```
class PluginInformation
{
public:
  ...
  void setTechlogViewerActivity(const QString
  &techlogViewerActivity)
};
This is an example:
static QString ReadDataActivityId(QLatin1String("f1007f1e-1ce3-477e-a47f-d91f4e7e1b7b"));
void MyFirstPlugin::getInformation(PluginInformation& pluginInformation) const
{
  pluginInformation.setTechlogViewerActivity(ReadDataActivityId);
}
```

Note: Techlog Viewer is single well by design; developing a plug-in that uses several wells will result in an assert being displayed in the Techlog Viewer output window.

Signed plug-ins

Having a signature on the plug-in is not necessary for internal development only.

However a .sign file is mandatory if any external deployment is planned.

To generate the signature file, please contact the Wellbore Product Champion Edgard Rivollier - erivollier@slb.com

Create unit tests for your plug-in

By exposing a couple of basic concepts, Ocean for Techlog enables plug-in developers to write and run automated tests using their unit testing framework of choice while still giving the unit tests access to the full functionality of Ocean for Techlog. The tutorial "Unit Testing Techlog Plug-ins" in the **OceanForTechlog.chm** file shipped with the Ocean package outlines how to get started and how to integrate the tests into a continuous integration environment.

Please refer to this tutorial for more details on how to create unit tests with Ocean for Techlog.

Creating a Test plug-in with Visual Studio

To create a Test plug-in project using Visual Studio:

Add a new test plug-in project to the solution that contains an Ocean plug-in project by right clicking on the solution in the Solution Explorer and selecting **Add > New Project** in the context menu. In the Project types area, under **Visual C++** project type, select **Ocean > Techlog 2024.4**. Then select the **Ocean Test Plug-in** template.

Note: A test project cannot be created in an empty Visual Studio solution. The test project wizard uses the main plug-in project in the solution.

Provide the name "TestMyFirstPlugin" for the project. Click **OK** to start the Wizard (see Figure 41).

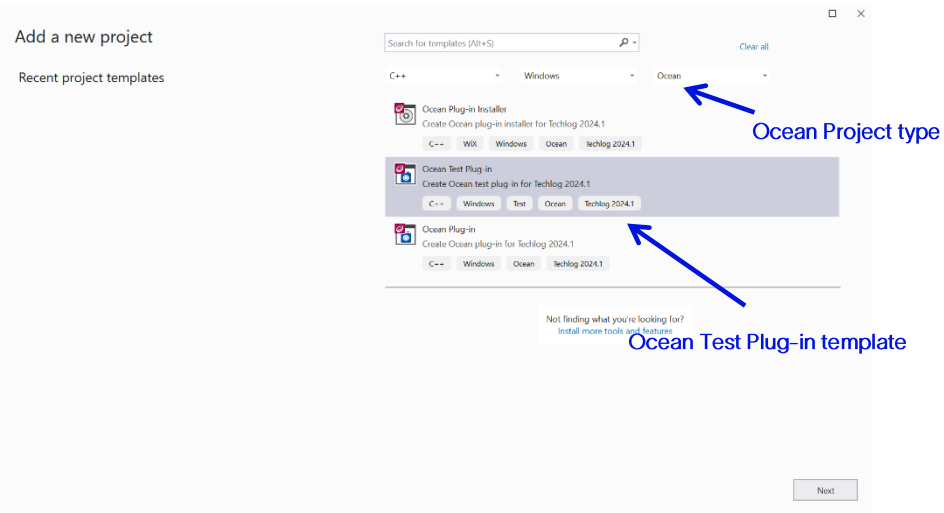


Figure 41: New project window

The test plug-in wizard opens (see Figure 42).
Set these inputs:

- **Class:** test plug-in class name
- **Vendor name:** the name of the plug-in owner
- **Version:** the plug-in version

- **Techlog project:** the Techlog project data that you want to use to run your unit tests. It is an optional input. If you don't specify any Techlog project, tests will be run in Techlog with a temporary project.
- **Main project:** select the Ocean plug-in in the solution that you want to test. The Ocean plug-in will be a dependent library of the Test plug-in.

Click **Finish** in the dialog.

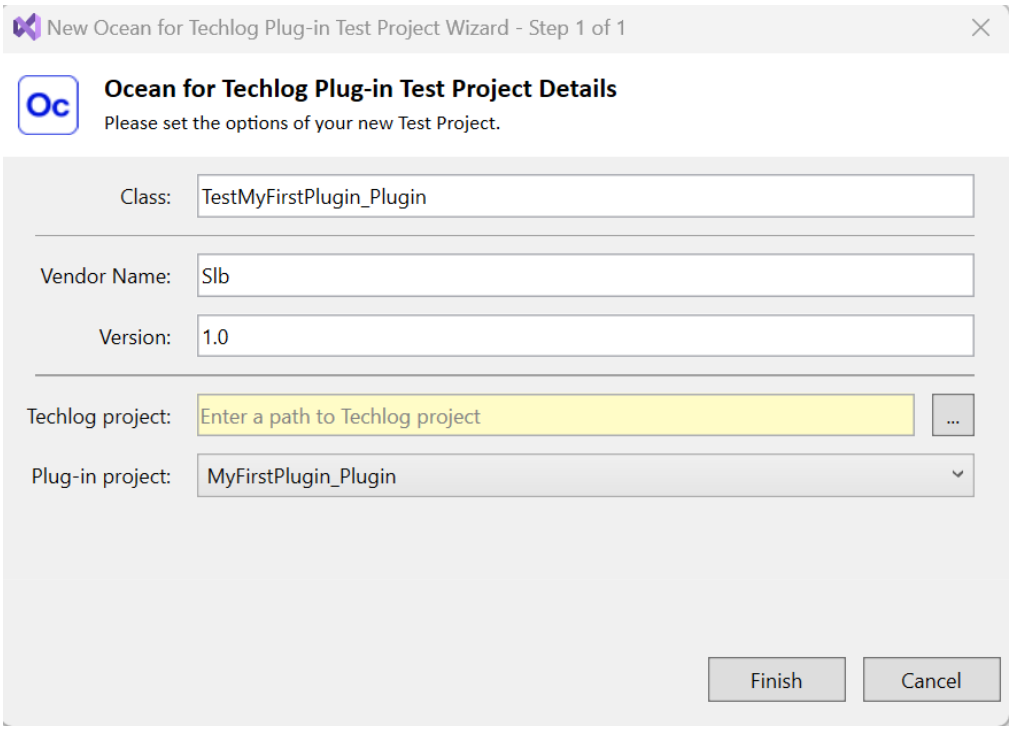


Figure 42: Test Plug-in wizard

Note: The path to the "3rdparty" folder of the Ocean package that contains Google test libraries (debug and release folders) and header files (include folder) is added to the project by the wizard.

See the "Techlog Test Adapter" section for more information on how to access and modify test settings and run plug-in unit tests with the Techlog Test Adapter.

Inspecting the files

The Ocean Test Plug-in Wizard adds a project named "TestMyFirstPlugin" in the Visual Studio Solution Explorer. The project contains header and source file for the Test Plugin class that was created, and the test activity and runner classes (see Figure 43.)

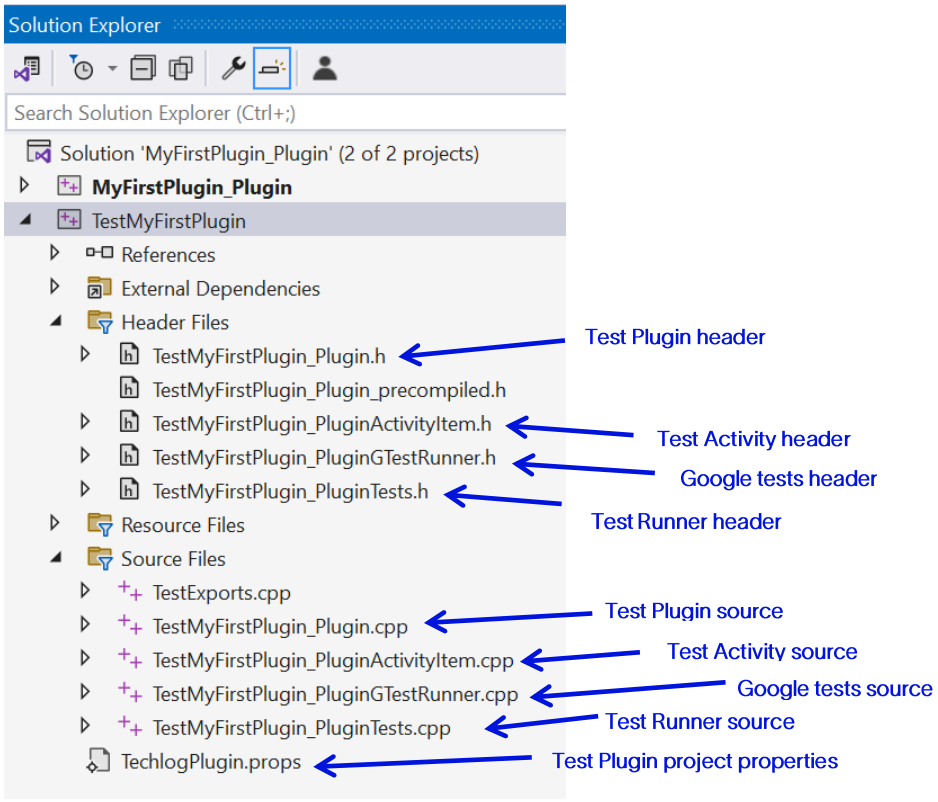


Figure 43: Test plug-in header and source files in Solution Explorer

The Test Activity runs all the Google tests implemented in the Test plug-in through the Test runner utility class.

Implement the tests

The Google test API provides a number of options you may consider depending on your requirements. Refer to the official Google test online documentation at http://code.google.com/p/googletest/.

In **TestMyFirstPluginTests.cpp** file, there are two types of Google tests created by the wizard.

The first one uses the **TEST** macro to define the test.

TEST has two parameters: the test case name and the test name. After using the macro, define your test logic between a pair of braces. Use a bunch of macros to indicate the success or failure of a test.

In this example, the test creates a well in Techlog project, sets its color property to blue and checks if the color is correctly set.

```
TEST(GTestName1, OkTest)
{
  Lock lock = LOCK_CREATE_AND_ACQUIRE_ALL(lock);
  Project project = Session::current().mainProject();
```

```
  Well well = Well::create("MyWell", project);
  Droid wellDroid = well.droid();
  well.setColor(Qt::blue);
  lock.release();

  lock = LOCK_CREATE_THEN_ACQUIRE_OR_RETURN(lock, wellDroid);
  well = DomainObject::get(wellDroid).tryCast<Well>();

  if (well.isNull())
  {
    ASSERT_FALSE(well.isNull());
    lock.release();
    return;
  }

  EXPECT_EQ(well.color(), Qt::blue);

  lock.release();

  lock = LOCK_CREATE_AND_ACQUIRE_ALL(lock);
  well.erase();
  lock.release();
}
```

The second one uses the **TEST_F** macro that defines a Google test fixture.

A test fixture is a place to hold objects and functions shared by all tests in a test case. Using a test fixture avoids duplicating the test code necessary to initialize and cleanup those common objects for each test. It is also useful for defining commonly used sub-routines that your tests may need.

In this example "MyWell" is initialized in **SetUp** method called before the test is run. Check in the test fixture if the color of "MyWell" is blue. "MyWell" is erased in **TearDown** method called after the test is run.

```
void TestMyFirstPluginTest::SetUp()
{
  Lock lock = LOCK_CREATE_AND_ACQUIRE_ALL(lock);
  Project project = Session::current().mainProject();
  Well well = Well::create("MyWell", project);
  well.setColor(Qt::blue);
  lock.release();
}

void TestMyFirstPluginTest::TearDown()
{
  Lock lock = LOCK_CREATE_AND_ACQUIRE_ALL(lock);
  Project project = Session::current().mainProject();
  Well well = project.wells().get("MyWell");
  well.erase();
  lock.release();
```

```
}


TEST_F(TestMyFirstPluginTest, WellColor)
{
  Lock lock = LOCK_CREATE_AND_ACQUIRE_ALL(lock);
  Project project = Session::current().mainProject();
  Well well = project.wells().get("MyWell");

  if (well.isNull())
  {
    ASSERT_FALSE(well.isNull());
    lock.release();
    return;
  }

  EXPECT_EQ(well.color(), Qt::blue);

  lock.release();
}
```

The Ocean test plug-in project is created in a Visual Studio solution that already hosts an Ocean plug-in project to allow the developer to make calls to the Ocean plug-in methods in Google tests.

In this example, **ReadDataActivity** of **MyFirstPlugin** has a public method to remove from a Techlog **Variable** all the missing values. Test this plug-in functionality by calling it from a Google test in my test plug-in.

```
Variable ReadDataActivity::removeMissingValues(Variable variable)
{
  Lock lock = LOCK_CREATE_AND_ACQUIRE_ALL(lock);
  Dataset dataset = variable.dataset();
  Well well = dataset.well();

  Variable ref = dataset.findReferenceVariable();

  QVector<float> resultVarValues;
  QVector<float> resultRefValues;

  int rowCount = variable.rowCount();

  for (int i = 0; i < rowCount; i++)
  {
    if (variable.getFloatValue(i) != Absent::MissingValue)
    {
      resultVarValues.append(variable.getFloatValue(i));
      resultRefValues.append(ref.getFloatValue(i));
    }
  }
```

```
    Dataset resultDataset =
  Dataset::create(QString("%1_result").arg(dataset.name()),
  ref.name(), ref.format(), resultVarValues.count(), well);

  Variable resultRef = resultDataset.findReferenceVariable();
  resultRef.setFamily(ref.family());
  resultRef.setUnit(ref.unit());
  resultRef.setFloatValues(resultRefValues);

  Variable resultVar =
  Variable::create(variable.name(), resultDataset,
  variable.format(), VariableTypeContinuous, 1);
  resultVar.setFamily(variable.family());
  resultVar.setUnit(variable.unit());
  resultVar.setFloatValues(resultVarValues);

  lock.release();

  return resultVar;
}
```

The first thing to do is to export the **ReadDataActivity** class when **MyFirstPlugin** is built and import **ReadDataActivity** class when **TestMyFirstPlugin** is built. Do this by adding a conditional compilation tag in **C/C++ > Preprocessor > Preprocessor Definitions** to **MyFirstPlugin** project settings (see Figure 44).

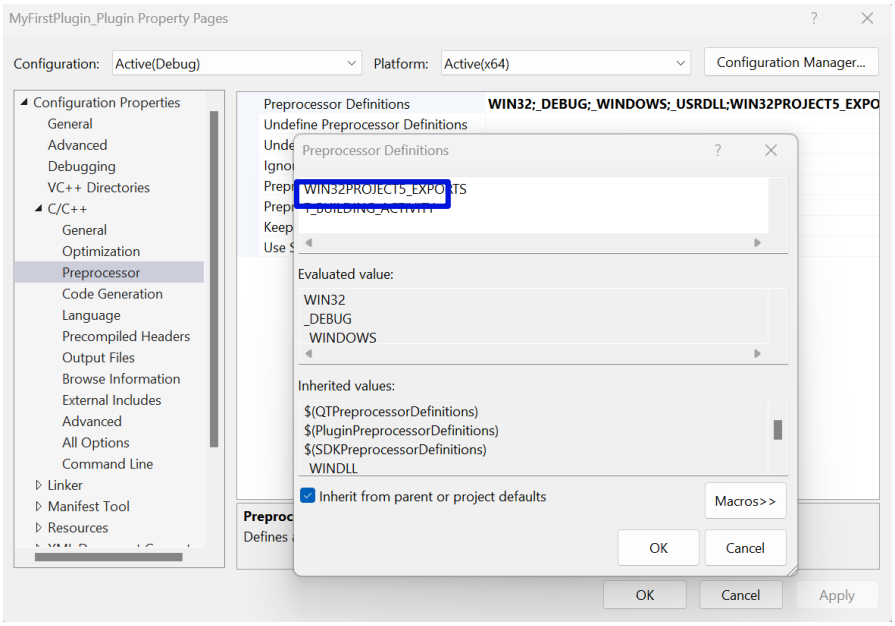


Figure 44: Add conditional compilation tag in Ocean plug-in settings

Then in **ReadDataActivity** header file, add this code:

```
#   define DllExport __declspec( dllexport )
```

```
#else
#   define DllExport __declspec( dllimport)
#endif

class DllExport ReadDataActivity : public Slb::Ocean::Techlog::AbstractActivity
{
  Q_OBJECT;

private:
  void run();

public:
  Slb::Ocean::Techlog::Variable
  removeMissingValues(Slb::Ocean::Techlog::Variable variable);
};
```

The removeMissingValues function is imported by the Test plug-in; call it in a Google test.

```
TEST_F(TestMyFirstPluginTest, RemoveMissingValues)
{
  Lock lock = LOCK_CREATE_AND_ACQUIRE_ALL(lock);

  Project project = Session::current().mainProject();

  Variable variable =
  project.wells().get("Well1").datasets().get("DATAFULL").
  variables().get("GR");

  lock.release();

  ReadDataActivity *readDataActivity = new ReadDataActivity();
  Variable resultVar = readDataActivity->removeMissingValues(variable);

  lock = LOCK_CREATE_THEN_ACQUIRE_OR_RETURN(lock, resultVar);

  for (int i = 0; i < resultVar.rowCount(); i++)
  {
    if (resultVar.getDoubleValue(i) == Absent::MissingValue)
    {
      ASSERT_FALSE(true);
      lock.release();
      return;
    }
  }

  ASSERT_TRUE(true);

  lock.release();
}
```

TearDown function the result dataset is erased after the test.

```
void TestMyFirstPluginTest::TearDown()
{
  Lock lock1 = LOCK_CREATE_AND_ACQUIRE_ALL(lock1);
  Dataset dataset =
  Session::current().mainProject().wells().get("Well1").datasets().
  find("DATAFULL_result");

  if (!dataset.isNull())
    dataset.erase();
  lock1.release();
}
```

Note: The test fixture **`TestWorkstep`** created by the Ocean Test plug-in wizard shows how to test AWI workstep method, waiting for the end of the processing in order to assert the results.

Run the tests

Once the solution is built **MyFirstPlugin** and **TestMyFirstPlugin** are listed by the Techlog module manager.
Open the module manager, refresh the list of plug-ins and enable **TestMyFirstPlugin**.

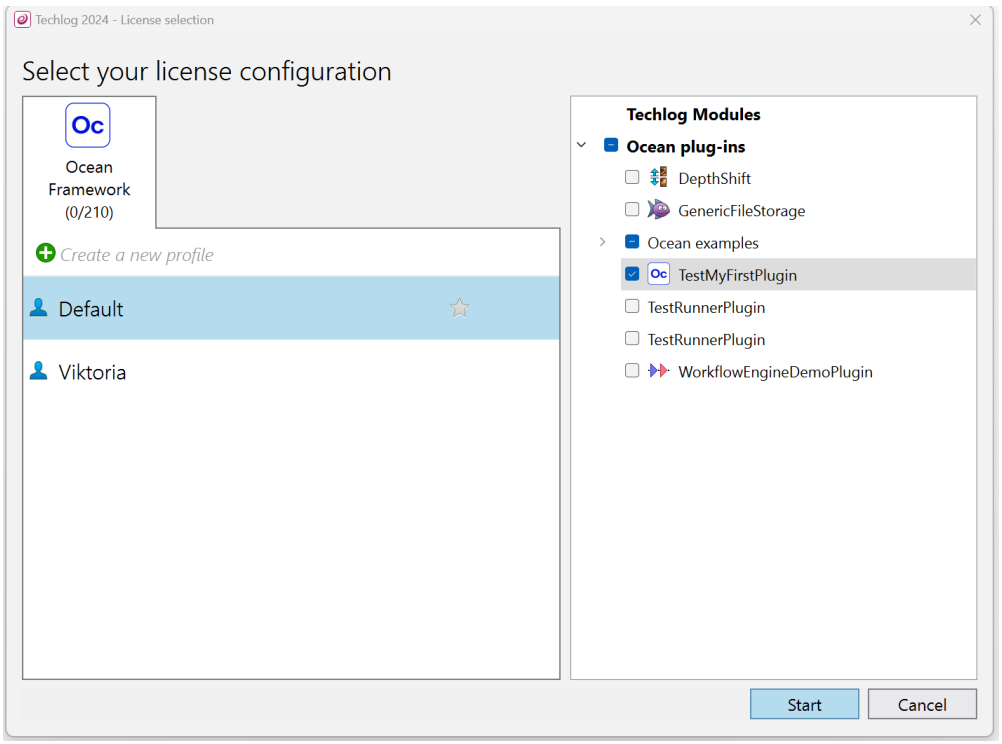


Figure 45: Enable test plug-in

GTest tab is added to the Techlog menus that contain a **gTest** action item from which the test are run in the Techlog context. When the tests have finished running, the user opens the result tests log file directly from the Techlog output console (see Figure 46).

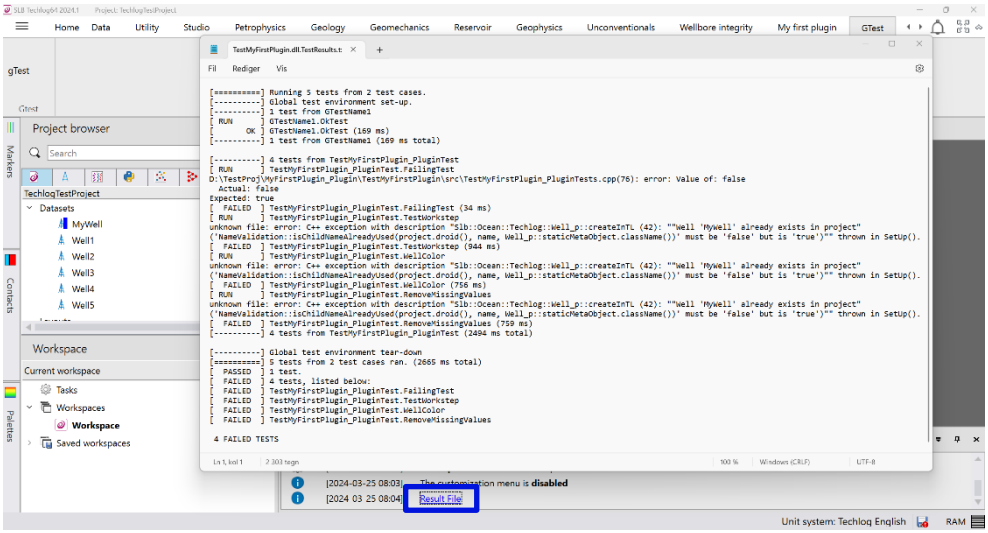


Figure 46: Gtest plug-in runs in Techlog context

You can also run the tests directly from Visual Studio through the Techlog Test Adapter.
See the "Techlog Test Adapter" section for more information on how to run plug-in unit tests with the Techlog Test Adapter.

4.Techlog Test Adapter

You can use the Techlog Test Adapter to run Ocean plug-in tests directly from the Visual Studio development environment in the Techlog context. This chapter describes the Techlog Test Adapter functionalities. It also describes the different options available to plug-in developers that enable Ocean plug-in tests to be run through the Techlog Test Adapter directly into the Visual Studio development environment, or on a build agent as Azure DevOps or TeamCity.

Run tests in Visual Studio

Once Google tests are implemented (see the "Create unit tests for your plug-in" section) in the Ocean test plug-in, you need to configure your **Plug-in test environment** to run tests.

There are two options to configure the Plug-in test environment in Visual Studio:

- Using the Ocean menu options (and Ocean plug-in properties).
- Using the Ocean RunSettings file

Configure the Plug-in test environment using Ocean menu options

The configuration options are available in the **Ocean > Techlog Test Adapter options** menu.

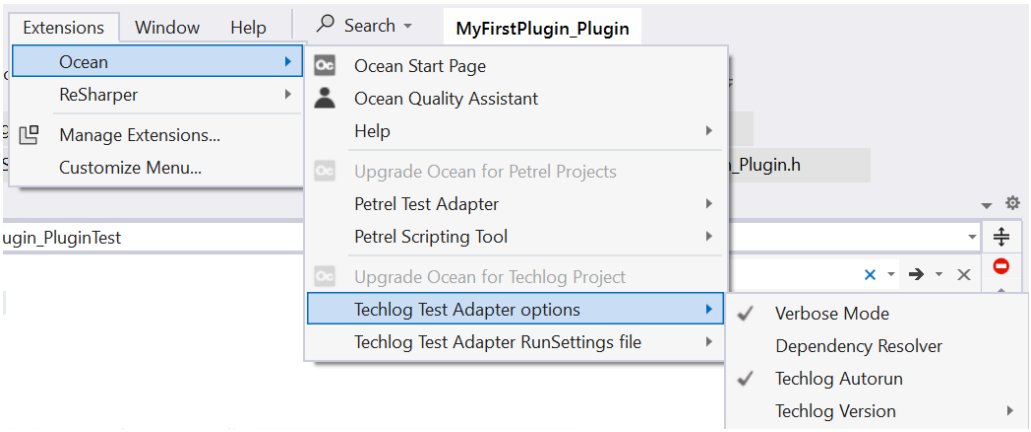


Figure 47: Ocean menu options

Techlog autorun

When tests are run, the Techlog Test Adapter can start Techlog to run plug-in unit tests in the Techlog context with some Techlog data. This only happens if the **Techlog Autorun** option is enabled.

Techlog version

You can use this option to select against which version of Techlog tests run.

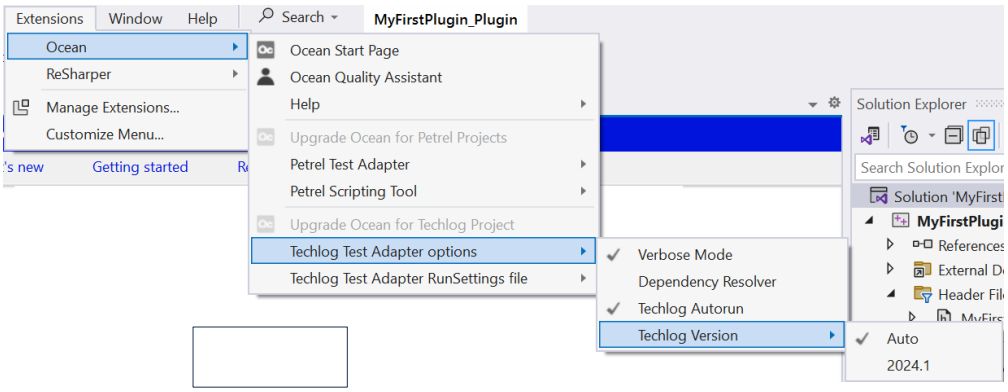


Figure 48: Techlog Version option

By default, the **Techlog Version** is set to **Auto**. This means that the Techlog Test Adapter automatically uses the appropriate Techlog version to run Ocean plug-in tests. The Techlog version auto detection is done through properties that you can set to the Visual Studio solution at different priority levels. If the Techlog autorun option is enabled, each priority level tells the Techlog Test Adapter to run a Test plug-in .dll with a specified Techlog project.

The first place (priority level) where the Techlog Test Adapter searches for the Techlog version and Techlog project path is a .runsettings file set to the Visual Studio solution. Use of this level is not recommended in the current "not Ocean RunSettings" workflow. It is supported for backward compatibility purposes only.

The second, but main, place where the Techlog Test Adapter searches for Techlog version and project path to run Ocean plug-in tests is at the Test plug-in project properties level.

From the **Ocean plug-in properties** dialog window, you can define the **Techlog project** path. The **Techlog version** can't be modified through the **Ocean plug-in properties** dialog window as it is defined by an Ocean for Techlog property sheet deployed with the Ocean framework.

See the "Ocean for Techlog property sheets" section for more information on Ocean properties related to the Techlog version.

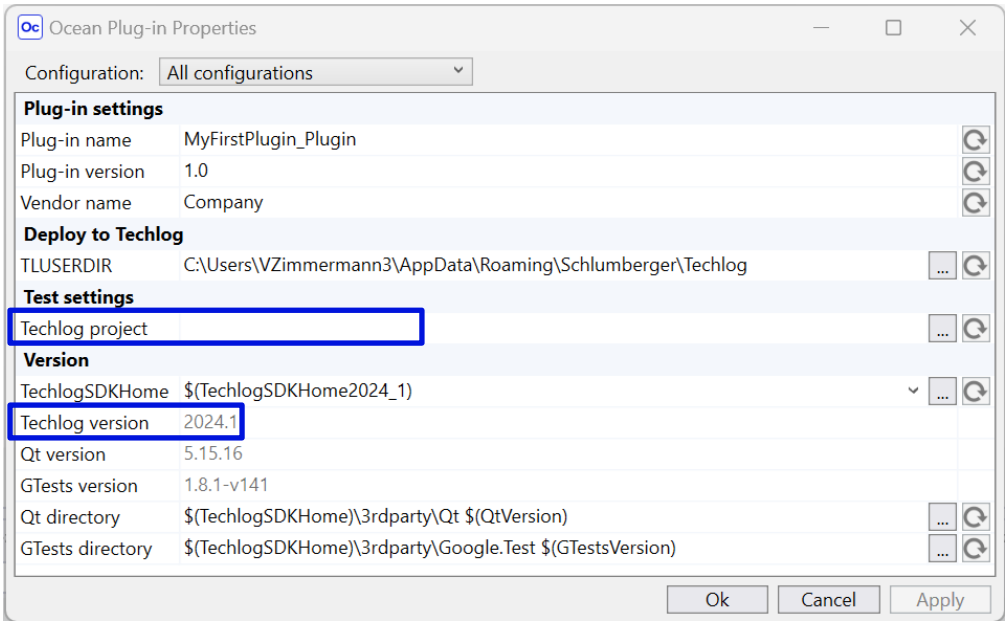
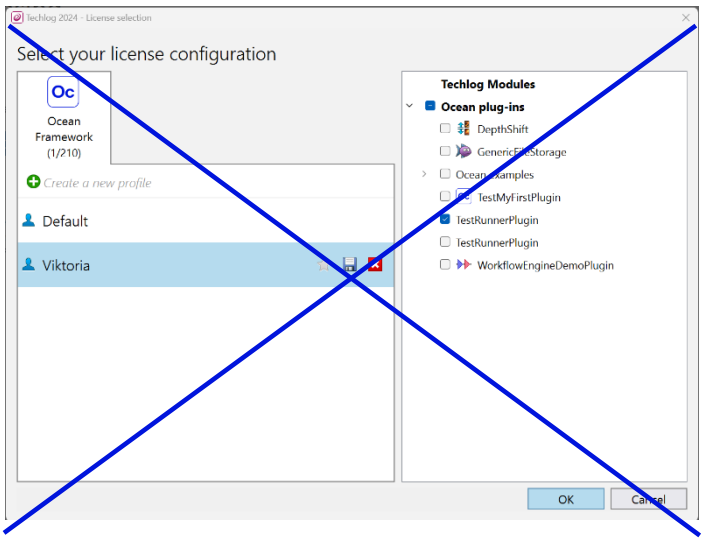


Figure 49: Ocean plug-in properties window

Note: If the Techlog autorun option is enabled, the Techlog Test Adapter starts the Techlog with the right version, runs the Ocean plug-in tests, and then closes Techlog when all tests have been run.

Ocean tests are run in Techlog through the **TestRunnerPlugin** that is shipped in debug and release modes with the Ocean framework. If one of these plug-ins is activated and saved in a favourite Techlog profile, then Techlog Test Adapter wouldn't be able to enable the right **TestRunnerPlugin** at the Techlog start and run Ocean tests. Because of this, please don't activate by default a **TestRunnerPlugin** in your favourite Techlog profile.



Dependency Resolver

When you enable the **Dependency Resolver** option, the missing plug-in dependencies are copied to the test assembly folder before running tests. This option is enabled by default.
Warning: We do not recommend using the Dependency resolver option because this option is not available on a build agent.

Verbose Mode

You can use this option to provide additional details in the Test output console of Visual Studio as to what the Techlog Test Adapter is doing.

Custom Code Coverage

This option is only available in Visual Studio Enterprise and is disabled by default. The option must be disabled if you want to use Native code coverage (see "Code coverage options" for more details).

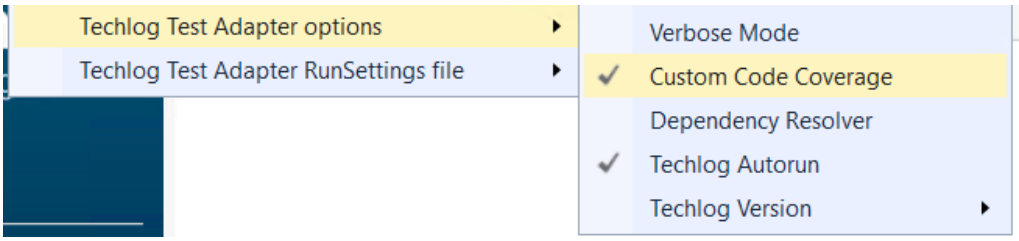


Figure 50: Custom Code Coverage option

Discover and run tests

Once the Plug-in test environment is set up (using the Ocean menu options or an Ocean RunSettings file), tests can be discovered by the Techlog Test Adapter and displayed in the Visual Studio Test Explorer as follows:

1. Set the Default Processor Architecture option to **x64 in TEST > Test Settings >** Default Processor Architecture menu
2. Open the Visual Studio Test Explorer window from the **TEST > Windows** menu
3. Build the Visual Studio solution

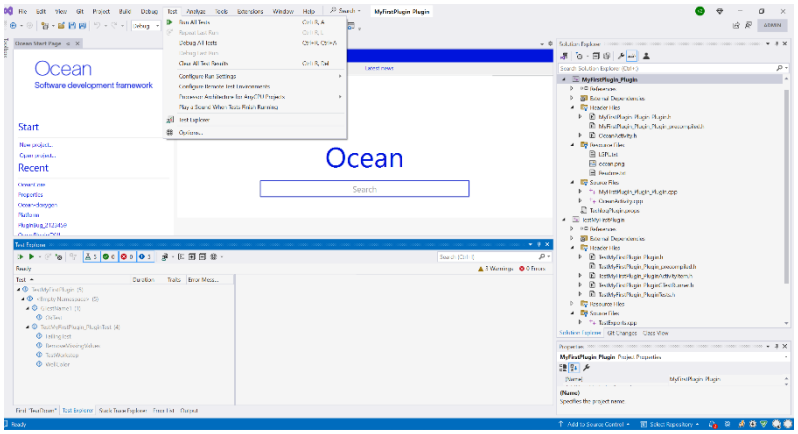


Figure 51: Tests in Test Explorer

To run tests, click **Run all** or **Run selected tests**.

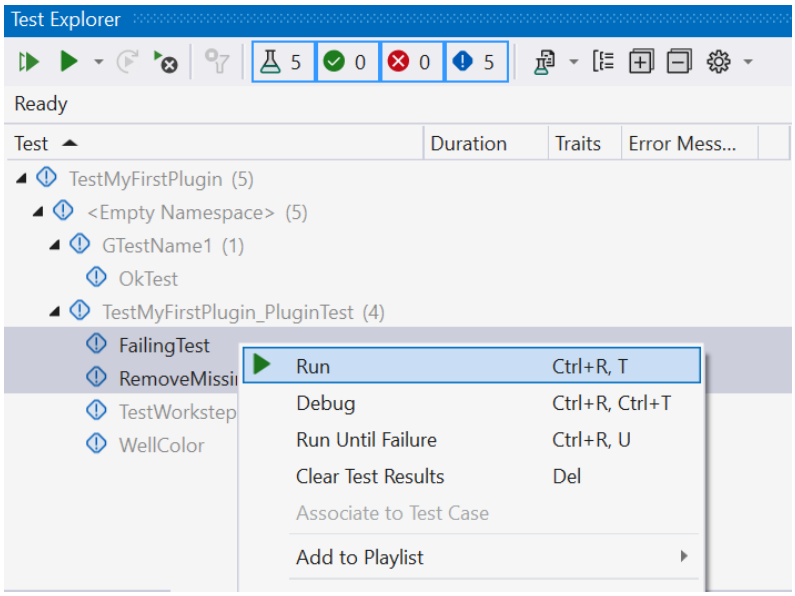


Figure 52: Run tests

Code coverage options

There are two different types of the code coverage available: Native code coverage and Custom code coverage (see the "Code Coverage types" for differences).

Run tests with Native code coverage

To run tests with Native code coverage:

1. Depending on the Plug-in test environment used:

- In the Ocean menu, disable Custom Code Coverage option
- In the Ocean RunSettings file, select Native code coverage type

2. Optional: set Profile of C++ projects (that you want to cover) to **Yes** (but not always necessary: please check the need for this flag experimentally)

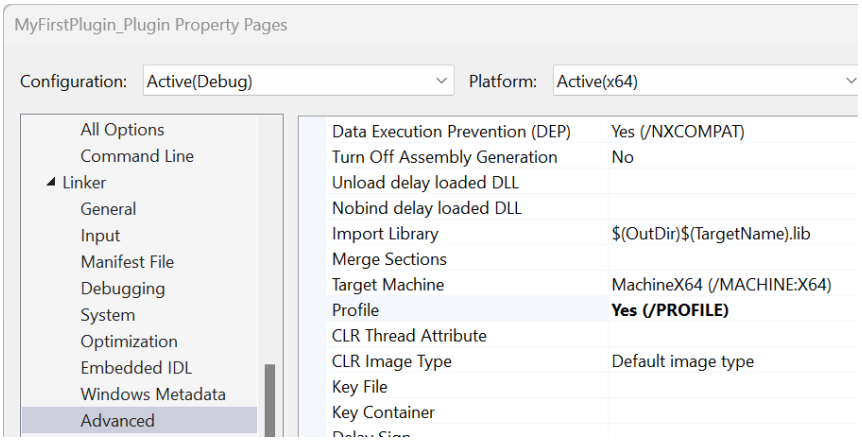


Figure 53: Profile property

3. Click **Analyze Code Coverage** in the Test Explorer of Visual Studio to start tests with Native code coverage

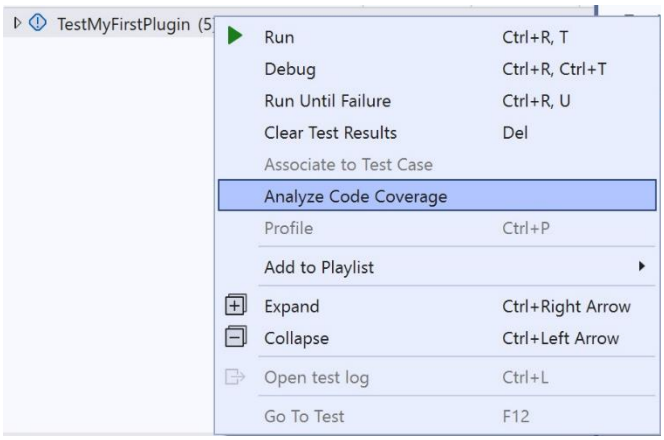


Figure 54: Start tests with Native code coverage

Run tests with Custom code coverage

To run test with Custom code coverage:

1. Depending on the Plug-in test environment used:
   - In the Ocean menu, enable **Custom Code Coverage** option
   - In the Ocean RunSettings file, select **Custom** code coverage type

2. Set the Profile of C++ projects that you want to cover to Yes (Figure 53)
3. To start tests with Custom code coverage, in the Visual Studio interface, click **Run**

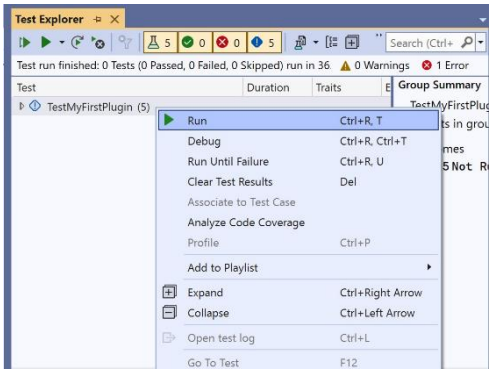


Figure 55: Start tests with Custom code coverage

Code coverage results

At the end of testing, the Techlog Test Adapter produces .coverage file that contains detailed results on the plug-in code covered by the unit tests.

The .coverage file is read by Visual Studio and displayed in the Code Coverage Results window.

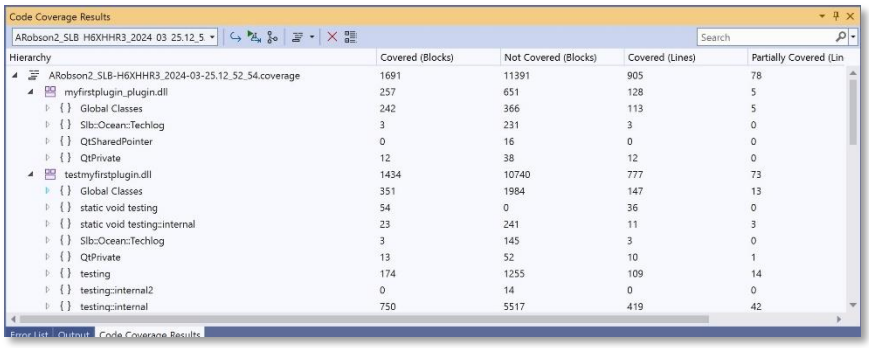


Figure 56: Code coverage results in Visual Studio

Configure the Plug-in test environment using Ocean RunSettings file

You can use a special Ocean RunSettings file and its editor to prepare the Plug-in test environment for the Visual Studio (instead of the Ocean menu options and Ocean plug-in properties) and for a build agent. The settings configurable using the editor are fully described in section "Configure the Plug-in test environment using Ocean menu options".

Create an Ocean RunSettings file

To create an Ocean RunSettings file:

1. Open Visual Studio with your Plug-in solution
2. Go to **Ocean > Techlog Test Adapter RunSettings file** Visual Studio Extensions menu and click **New**.

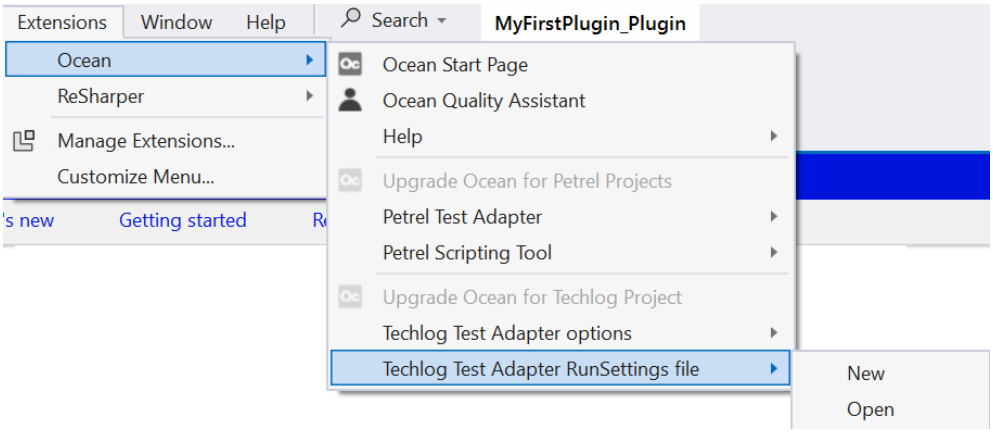


Figure 57: Create Ocean RunSettings file

The editor opens:

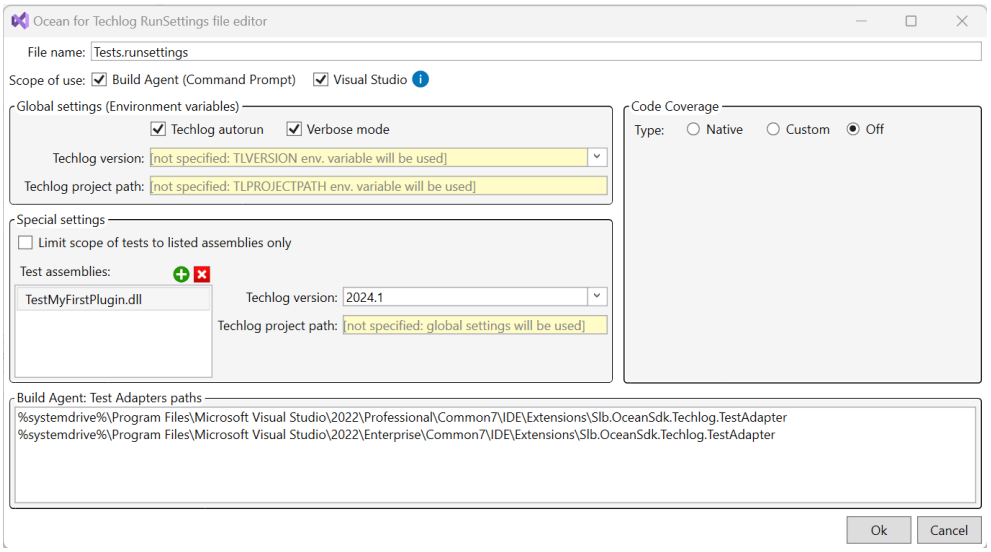


Figure 58: Ocean RunSettings file editor

In the **File name** input field you can specify the name of your new settings file with the .runsettings extension.

Select the **Build Agent** checkbox if you want to use this file on a build agent or in the Command Prompt.

Select **Visual Studio** if you want to use the Ocean RunSettings file in Visual Studio.

If you select the Visual Studio checkbox, all Ocean menu options and Ocean plug-in properties are ignored.

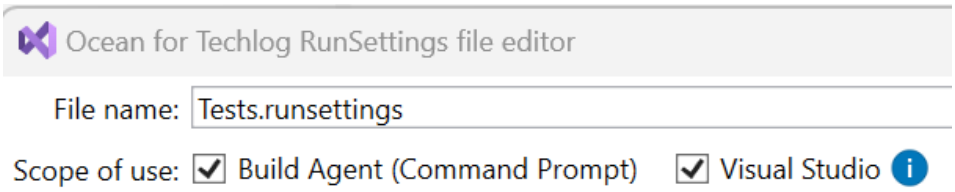


Figure 59: File name and Scope of use

There are four sections to set up your Plug-in test environment for Techlog Test Adapter unit tests.

- Global settings

In the Global settings section you can specify the global Techlog version and Techlog project path you want to use in test run. Techlog autorun and Verbose mode options are also available.

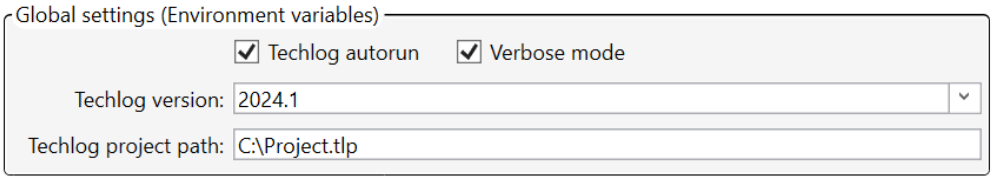


Figure 60: "Global settings" section

- Special settings

In the Special settings section you can specify which Techlog versions and Techlog project paths to use in the test run per Plug-in test assembly.

Note: Special settings override Global settings.

You can also filter input assemblies for the test run by selecting **Limit scope of tests to listed assemblies only**. By default, using this option on a build agent is recommended. If selected, only tests from listed assemblies are run using the Techlog Test Adapter. Any assemblies that aren't listed will not be analysed.

To rename an assembly double-click the assembly's name.

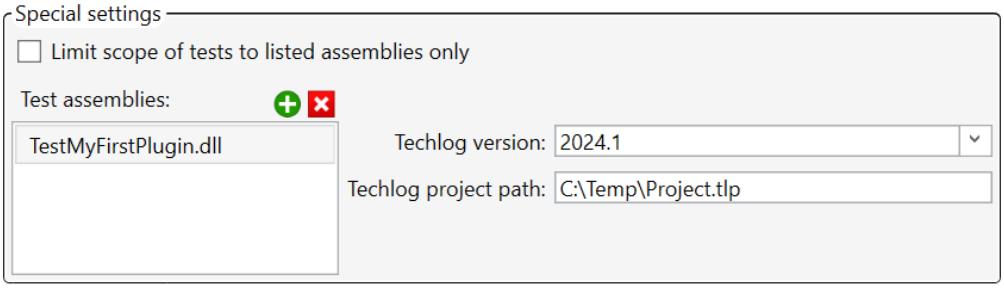


Figure 61: "Special settings" section

- Code coverage section

In the Code coverage section you can turn code coverage for tests on and off. Two code coverage types are available (see the "Code Coverage types" for details):

- **Native code coverage**. If you select **Native**, you must specify a list of assemblies you want to cover during the test run. You can also specify lists of functions and sources to exclude from the test run.


If the Native code coverage Assemblies list is empty, the test run will hang for a long time on the build agent and result in a timeout exception.

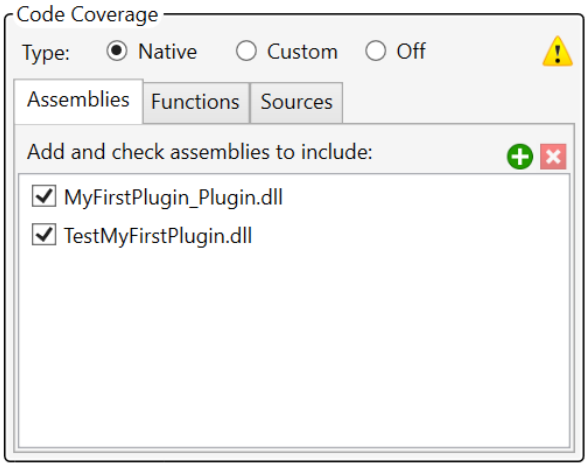


Figure 62: Native code coverage options

- Custom code coverage. If you select Custom, you can specify the output path for resulting XML file with Coverage results. Also, there are couple of settings for Visual Studio.

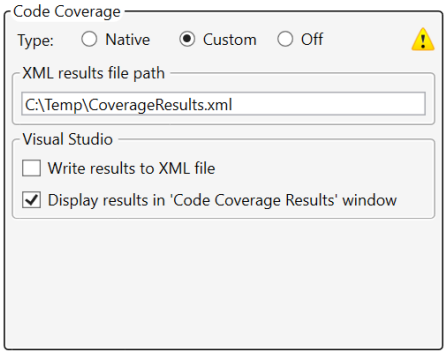


Figure 63: Custom code coverage options

- Test Adapters paths

In the "Build Agent: Test Adapters paths" section you must enter a path or paths to the Techlog Test Adapter. Make sure that you enter an existing path or paths on the target (build agent) machine.

The values are not used when you run tests from Visual Studio.

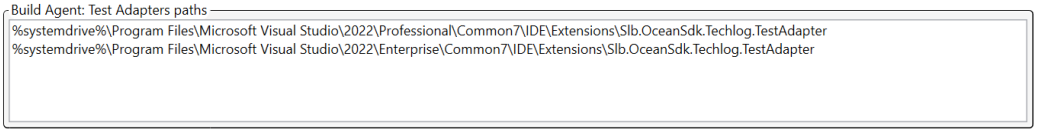


Figure 64: Test Adapters paths

Finally, to create the RunSettings file, click **Ok**. The new file is placed under the **Settings** folder of the solution and opened in the Code view.

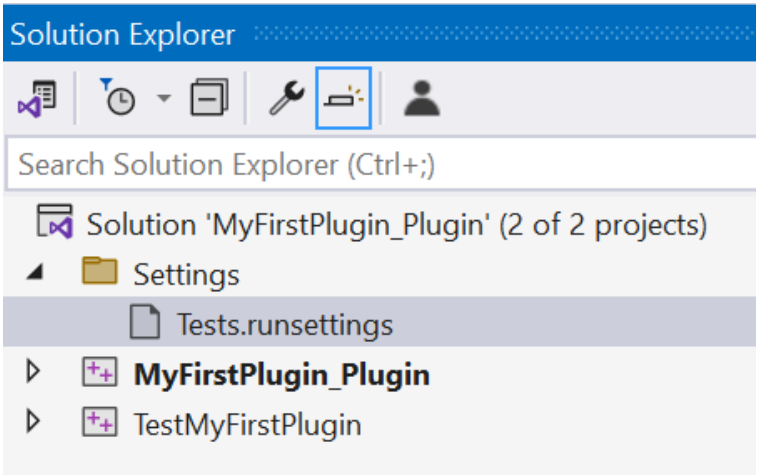


Figure 65: New Ocean RunSettings file in the solution

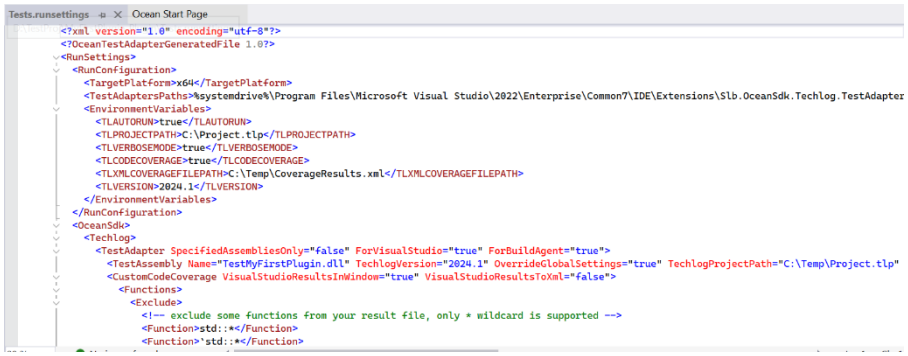


Figure 66: Ocean RunSettings file in the Code view

Modify existing Ocean RunSettings file

You can modify the created Ocean RunSettings file manually by using Code view, or by using the same Ocean RunSetting file editor. Which you can access through the context menu of the file, or through Visual Studio Ocean menu.

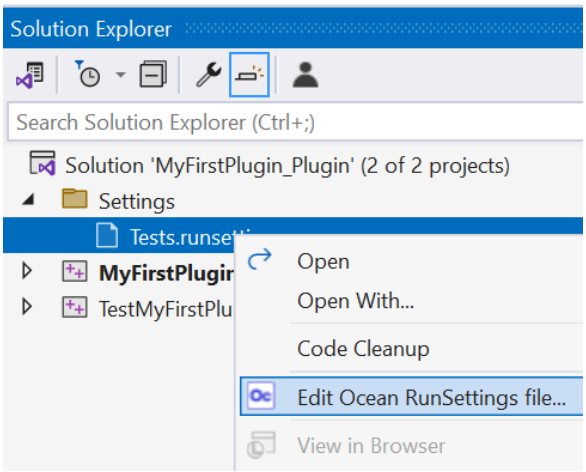


Figure 67: Edit existing Ocean RunSettings file

Apply your Ocean RunSettings file to current Plug-in solution

To use the Ocean RunSettings file in your Visual Studio plug-in solution, select it in the **Test Settings > Select Test Settings File** menu item (Figure 68 and Figure 69).

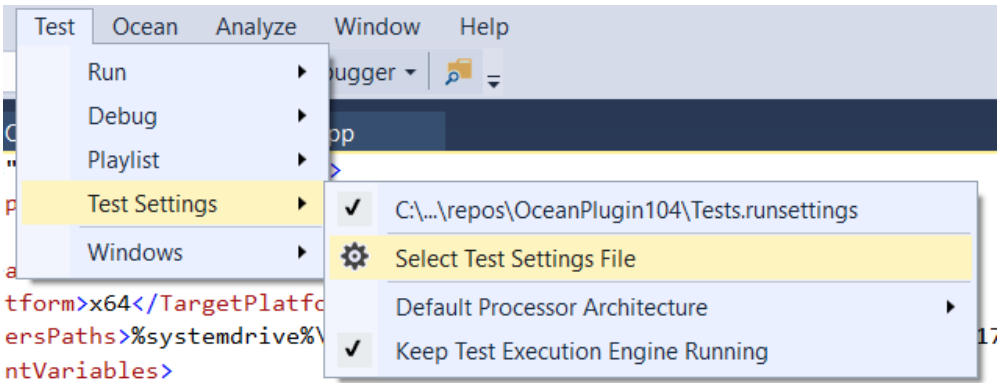


Figure 68: Select the .runsettings file in your Visual Studio 2022 solution

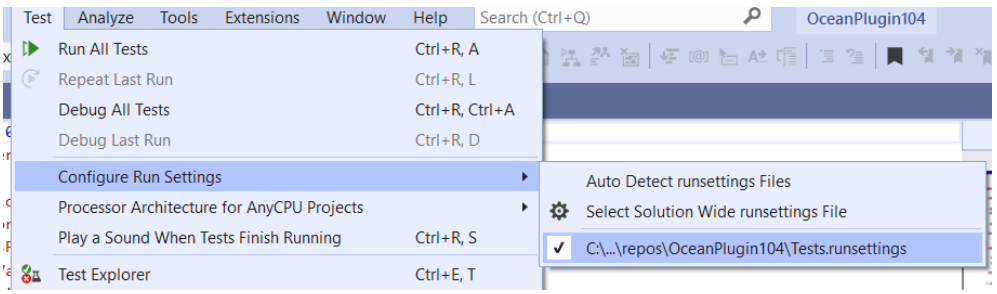


Figure 69: Select the .runsettings file in your Visual Studio 2022 solution

See the next "Setup Techlog Test Adapter on a build agent" chapter for how to use the Ocean RunSettings file on a build agent.

Setup Techlog Test Adapter on a build agent

Through the Techlog Test Adapter you can run plug-in tests on a build agent as Azure DevOps or TeamCity. This way you can set up a continuous integration environment for your plug-in through a build agent that is responsible to:

- Build the plug-in solution
- Run the unit tests associated to the plug-in through the Techlog Test Adapter
- With a targeted Techlog version
- With a targeted Techlog project or a temporary project
- Calculate Code Coverage and provide results

Build machine prerequisites

To create and configure pipeline (build definition) on an Azure DevOps build agent to run plug-in tests through the Techlog Test Adapter follow the steps below.

Note: You can also run Ocean plug-in tests with the Techlog Test Adapter on a TeamCity build agent.

The applications listed here must be installed on the build machine:

- Azure Pipelines Agent.
- Visual Studio Enterprise 2022 (with compiler v143).
- Schlumberger licensing tool with a valid Techlog license.
- Techlog 2024.4.
- Ocean for Techlog 2024.4.

Note: This document only describes how to setup a pipeline (build definition) with the Techlog Test Adapter on an Azure DevOps build agent. You can also run Ocean plug-in tests with the Techlog Test Adapter on a TeamCity build agent.

Prepare the Plug-in test environment

There are two main options to prepare the Plug-in test environment for a build agent:

- Use an Ocean RunSettings file. Modify existing Ocean RunSettings file and commit this file to your repository before going to Azure pipeline setup. **This option is recommended.**
- Directly use environment variables.

Warning: Native code coverage is only supported if the Ocean RunSettings file is used (for the workflows described).

Environment variables

This table describes all the environment variables provided by the Techlog Test Adapter. If you are not using the Ocean RunSettings file, or some fields in the RunSettings file are not specified, you can use these to configure the build agent with all Techlog Test Adapter options for the Visual Studio environment.

Variable	**Short description**
TLVERBOSEMODE	"Verbose Mode" option in the Visual Studio menu. Default value: "False"

TLCODECOVERAGE	"Custom Code Coverage" option in the Visual Studio menu. Default value: "False" Note: Only available if Visual Studio Enterprise is installed on the build agent.
TLAUTORUN	"Techlog Autorun" option in the Visual Studio menu. Default value: "True"
TLVERSION	"Techlog Version" option in the Visual Studio menu. Default value: "Auto" Note: If the value is "Auto" or empty, the Techlog version is resolved from a .runsettings file.
TLPROJECTPATH	Absolute path to a Techlog project to be used to run tests (only if TLAUTORUN is set to "True"). Default value: empty Note: If the value is empty, the Techlog project path is: Resolved from a .runsettings file. Not set and tests are run with a temporary project
TLXMLCOVERAGEFILEPATH	Only for Custom Code Coverage. Absolute path to a summary XML file that contains unit testing code coverage results. Default value: empty Note: Only available if Visual Studio Enterprise is installed on the build agent and TLCODECOVERAGE is set to "True".

Figure 70: Techlog Test Adapter environment variables

See the next "Azure pipeline setup" section for how to apply these variables for the build agent.

Azure pipeline setup

These steps explain how to create and configure a pipeline on Azure DevOps that builds the plug-in, runs the tests in Techlog using Techlog Test Adapter, and then computes code coverage results.

1. Connect to the Azure DevOps page and go to **Pipelines** menu
2. Click on **New pipeline** button and click **Use the classic editor**

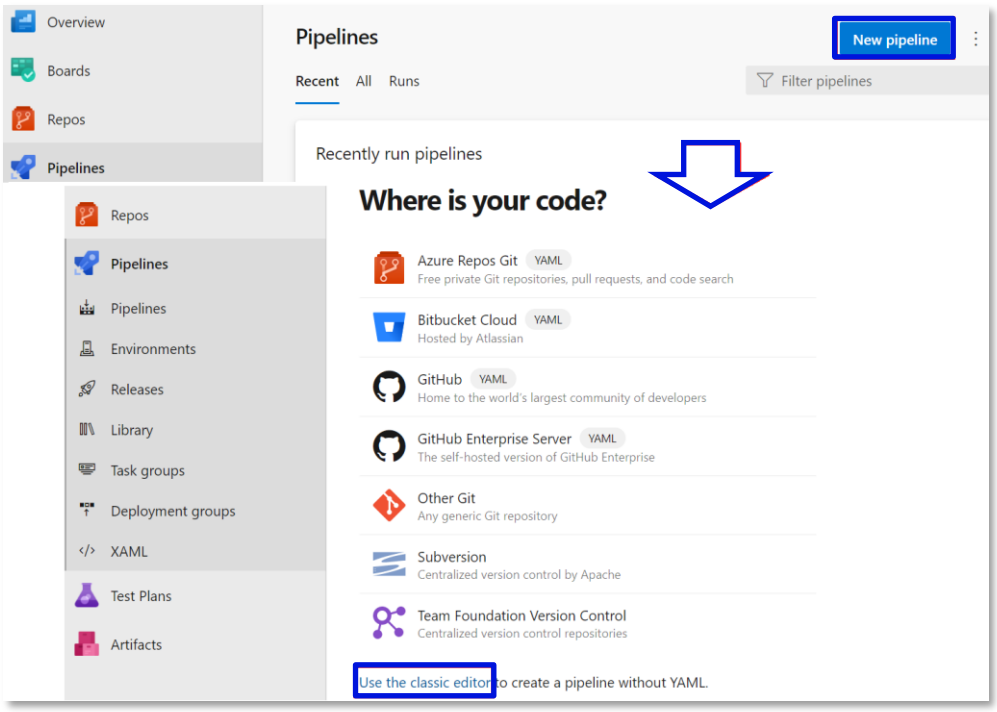


Figure 71: Create a new pipeline

3. Select the source control where the Ocean plug-in solution is hosted.
Azure Repos Git source:

- Select a **Team project** in the list to access Azure DevOps Git repositories.
- Select a **Repository** in the list that contains the plug-in solution.
- Select the **Default branch**.
- Click **Continue**.

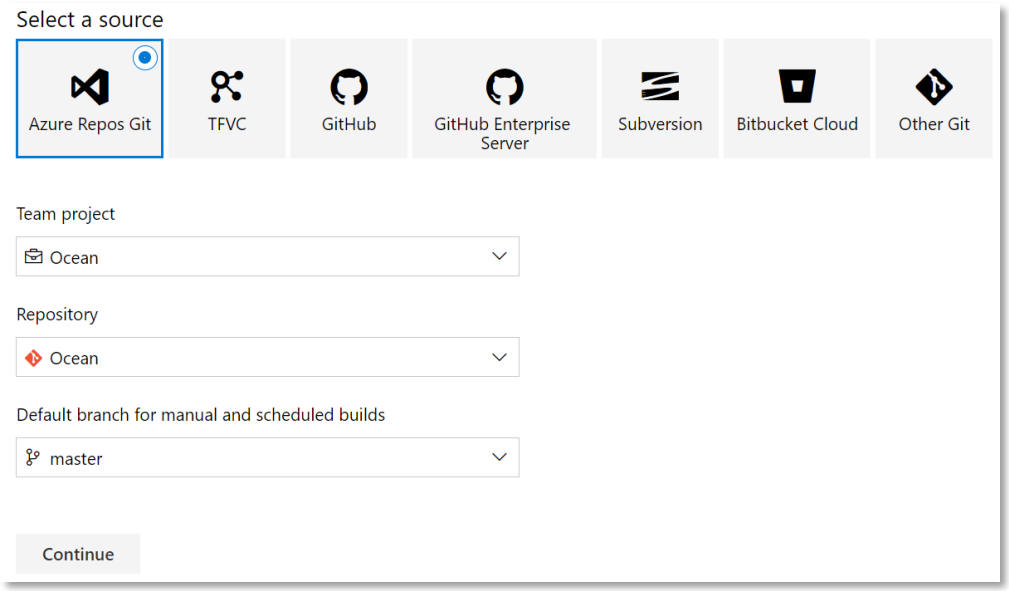


Figure 72: Azure Repos Git source

TFVC source:

- Enter the **Server path** to the repository that contains the plug-in solution.
- Enter the **Local path** where this plug-in solution must be copied to the build machine.
- Click **Continue**.

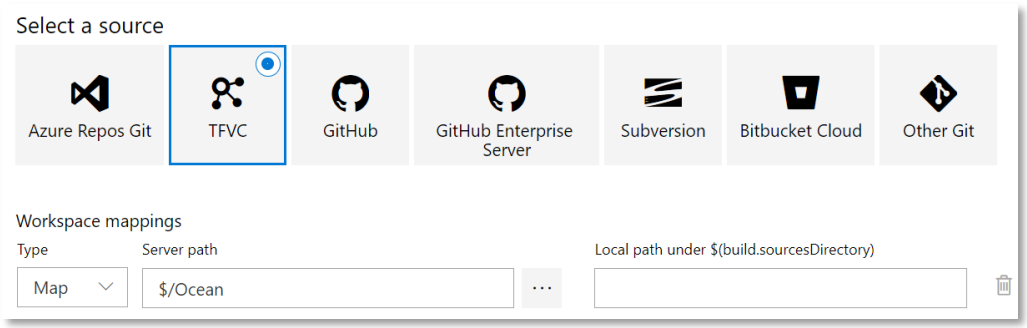


Figure 73: TFVC source

4. In the pipeline template page click **Empty job**

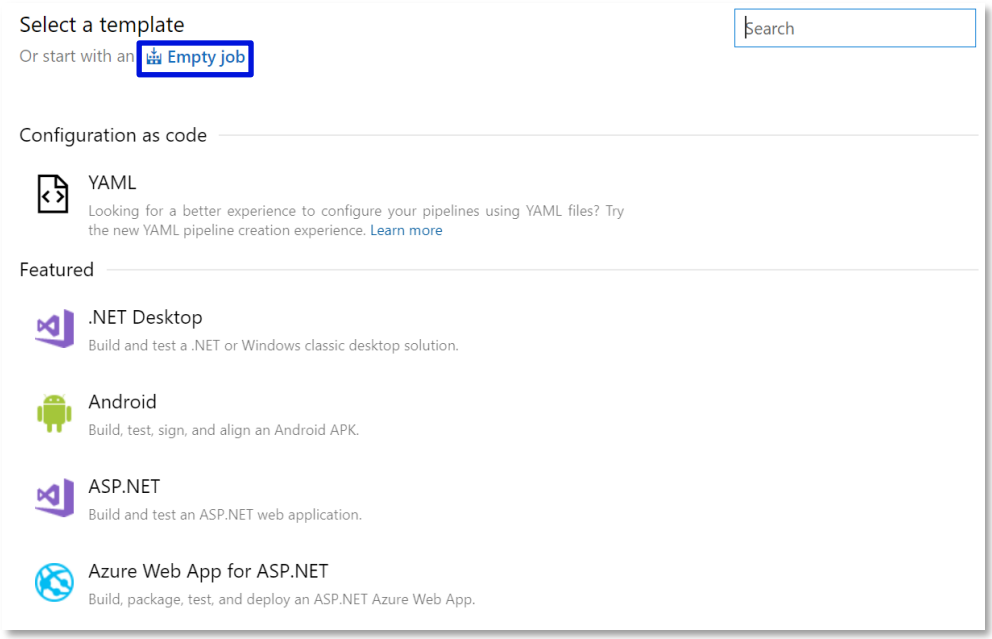


Figure 74: Pipeline template page

5. The pipeline is now created and you have to provide some settings to the process that will be run on the build agent:

- Provide a **Name** for the pipeline.
- Select an **Agent pool** in the list.

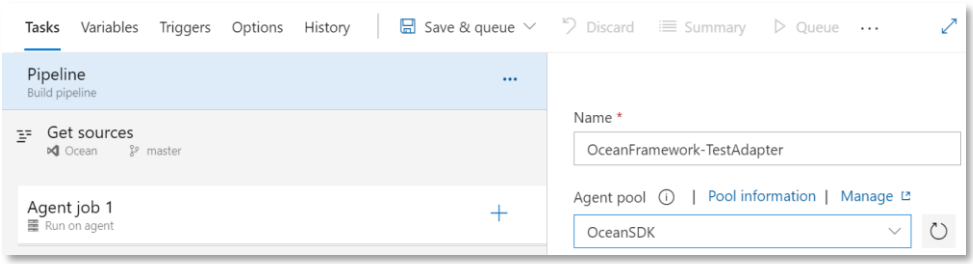


Figure 75: Pipeline settings

The pipeline is created in Azure with a default job named "Agent job 1". This is under this job that you will add tasks that which plug-in solution to build in Visual Studio, the plug-in tests to run with the Techlog Test Adapter for a given Techlog version / project and compute code coverage results.

6. Add a **Visual Studio build** task to the job that is responsible to build the Ocean plug-in Visual Studio solution.

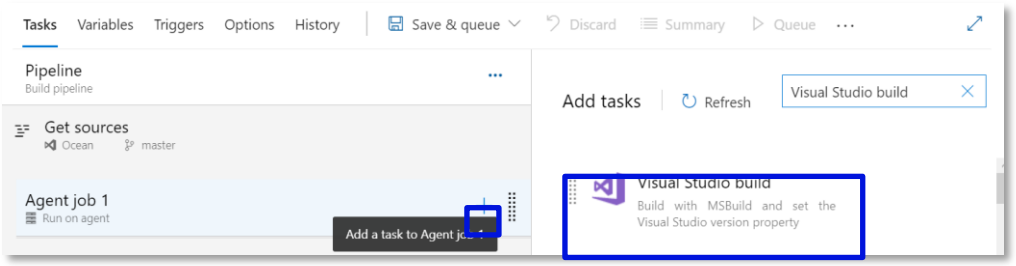


Figure 76: Add a Visual Studio build task

7. Define the **Visual Studio build** solution parameters:

- **Display name**.
- **Solution**: Relative path from repository root of the solution.
- **Visual Studio Version**.
- **MSBuild Arguments**: /p:Platform=x64.
- **Platform**: x64.
- **Configuration**: release or debug.

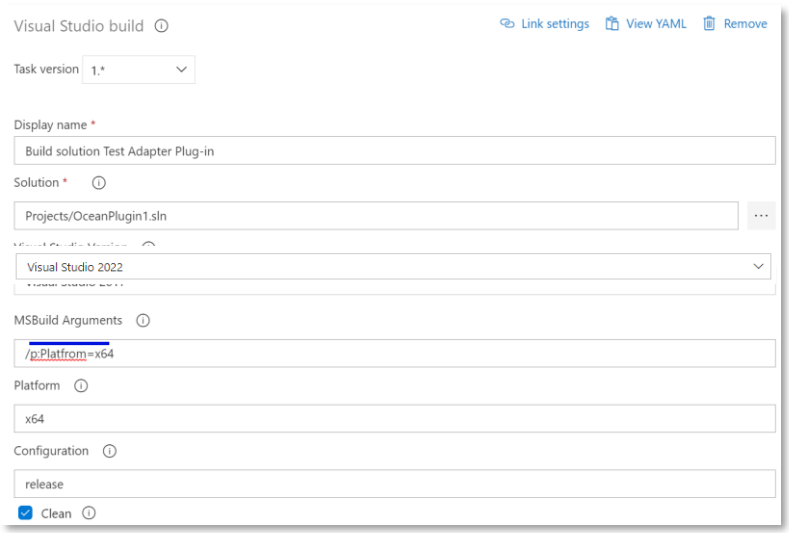


Figure 77: Visual Studio build solution parameters

8. An optional step is to add an **Extract files** task to the phase that is responsible to unzip a Techlog project hosted in your Ocean plug-in solution and used to run tests in Techlog.

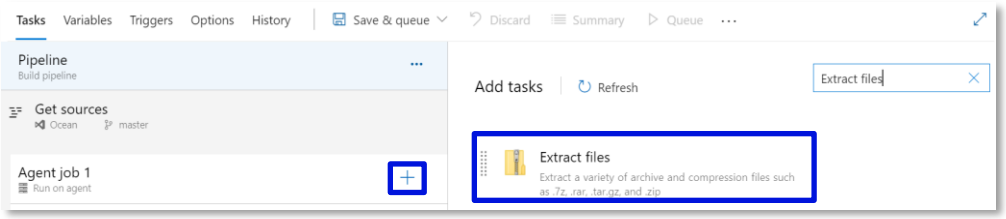


Figure 78: Add an Extract files task

9. Define the Extract files parameters:

- Display name.
- Archive file patterns**:** Relative path to the zip file from the plug-in solution repository.
- **Destination folder.**

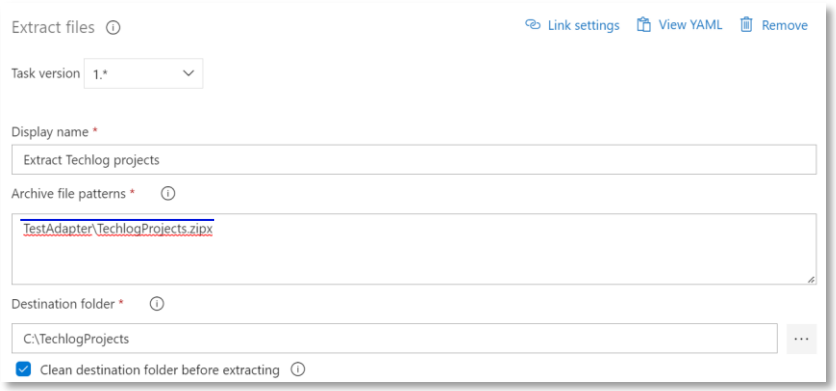


Figure 79: Extract files parameters

10. Add a **Visual Studio Test** task to the phase that is responsible to run the Ocean plug-in tests through the Techlog Test Adapter.

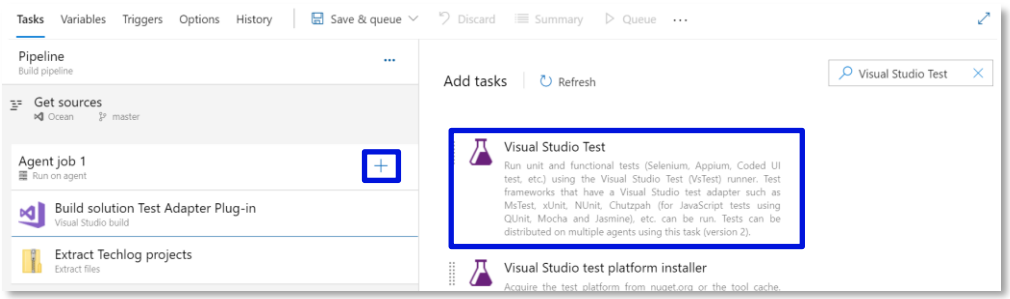


Figure 80: Add a Visual Studio Test task

11. Define the **Visual Studio Test** parameters:

- **Display name**.
- **Test assemblies**: relative path to test plug-ins output dll that contains Ocean unit tests. The file paths are relative to "Search folder" parameter. You can leave this field as default in a common case if Ocean RunSettings file is specified and "Limit scope of tests" option is enabled (see Figure 61) in the Ocean RunSettings file.
- **Search folder**: folder to search for "Test assemblies" (relative path to the local build folder).
- **Test platform version**: Visual Studio 2022.
- **Code coverage enable**: should be disabled (even if Code Coverage is enabled in the Ocean RunSettings file or in the Environment variables).

If you are going to use Ocean RunSettings file (Figure 81):

- **Settings file**: Path to your Ocean RunSettings file.
- **Path to custom test adapters**: This should be empty (default value) if specified in the Ocean RunSettings file.

If you are not going to use Ocean RunSettings file (Figure 82):

- **Settings file:** Optional, path to .runsettings file (without Ocean sections).
- **Path to custom test adapters**: Directory path to the custom Techlog Test Adapter (e.g. C:\Program Files (x86)\ Microsoft Visual Studio\2022\Enterprise \Common7\IDE\Extensions\Slb.OceanSdk.Techlog.TestAdapter).
- **Build platform**: Set to x64.
- **Other console options**: /Platform:x64.

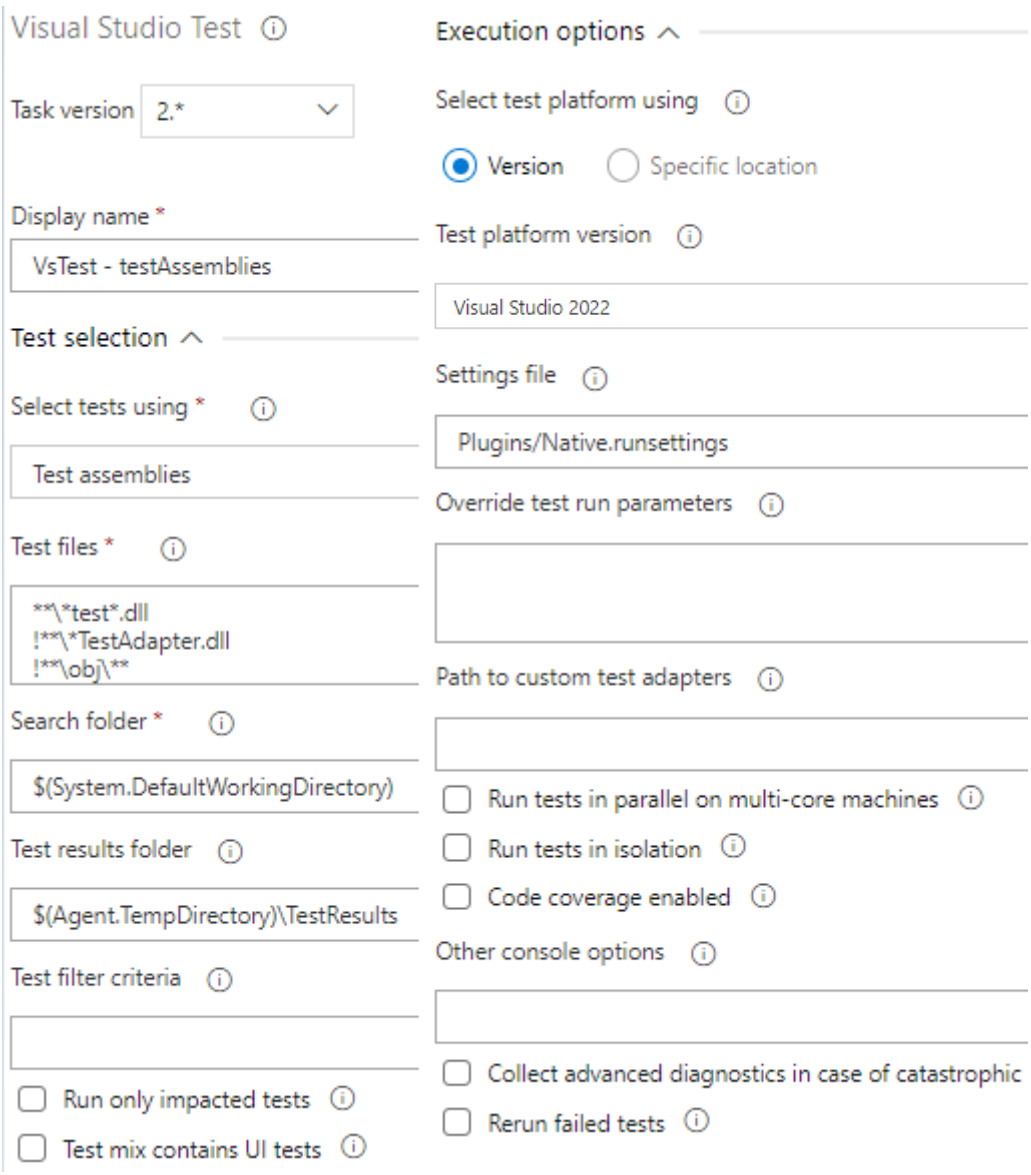


Figure 81: Visual Studio Test parameters (with Ocean RunSettings file)

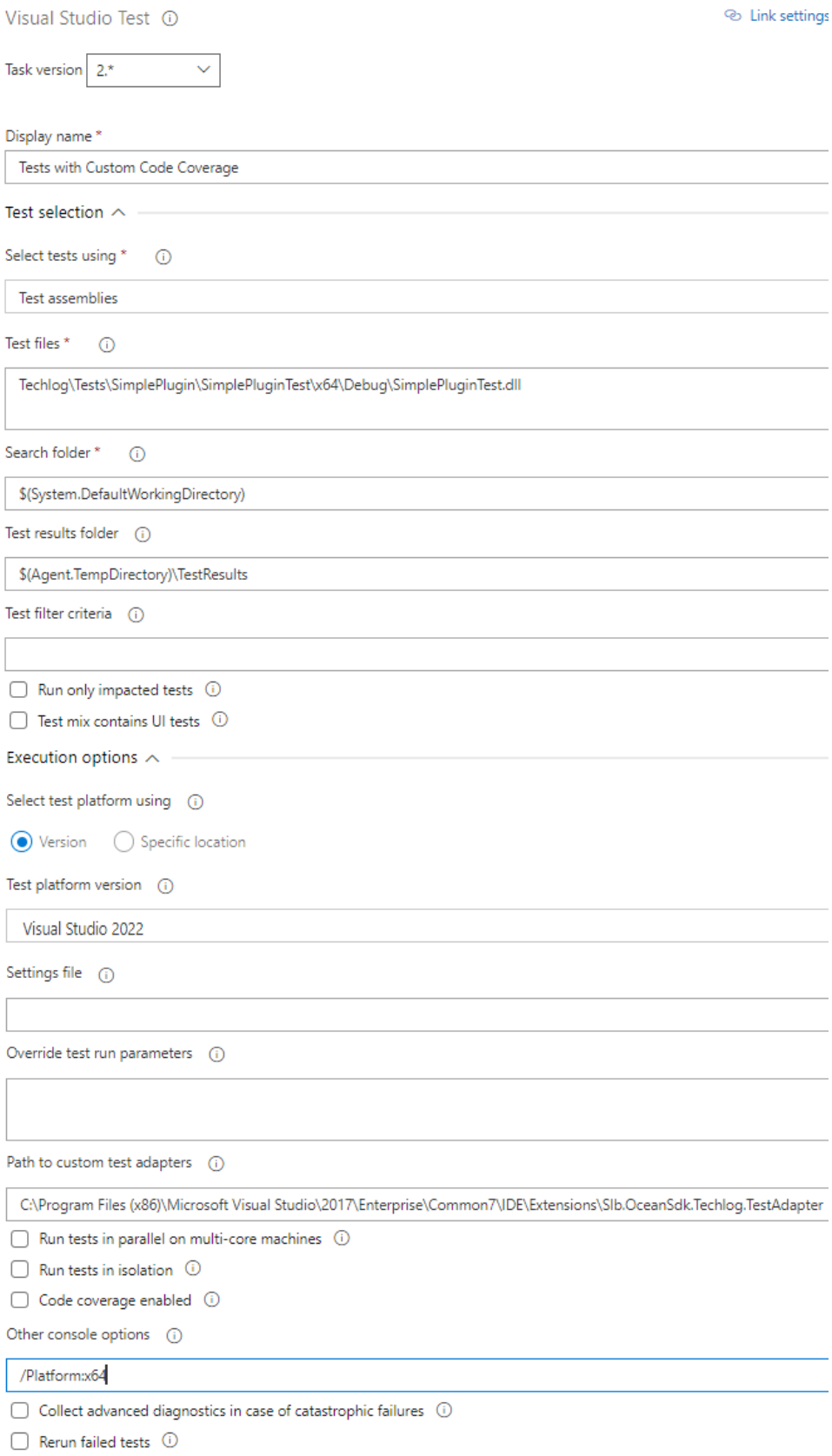


Figure 82: Visual Studio Test parameters without Ocean RunSettings file

12. You can skip this step if you are using Ocean RunSettings file and all required settings are specified in the Ocean RunSetting file:

Add Techlog Test Adapter environment variables to the pipeline (see Figure 70). Pipeline variables are available in all pipeline tasks created previously.

- Click the **Variables** tab of the pipeline.
- Add Techlog Test Adapter environment variables to the page that the build agent must use to run Ocean plug-in tests in Techlog.

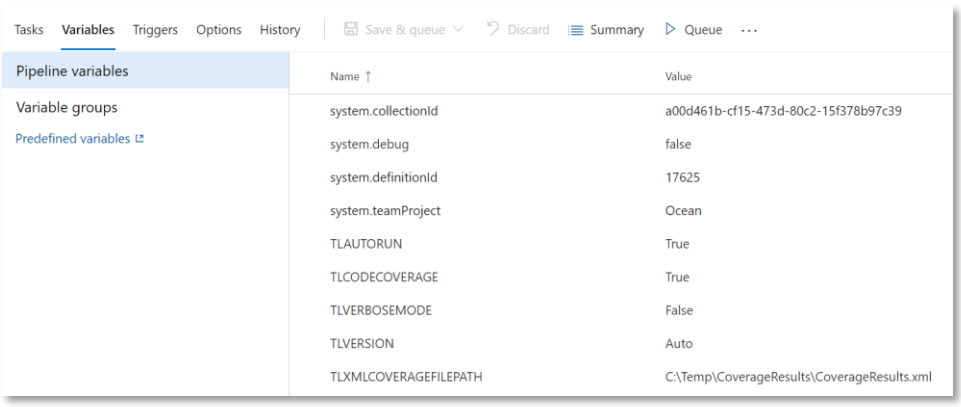


Figure 83: Pipeline environment variables

13. You can skip this step if Custom code coverage is disabled:

Add a **Publish build artifacts** task to the phase that is responsible to publish in artifacts of the build agent the results of the plug-in unit tests code coverage.

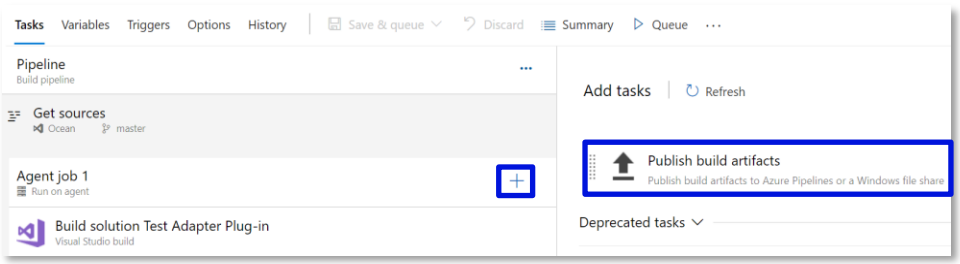


Figure 84: Add a Publish build artifacts task

14. You can skip this step if Custom code coverage is disabled:

Define **Publish build artifacts** parameters as:

- Display name.
- Path to publish: The folder path to publish the code coverage results on the build machine, or on the repository.
- Artifact name.

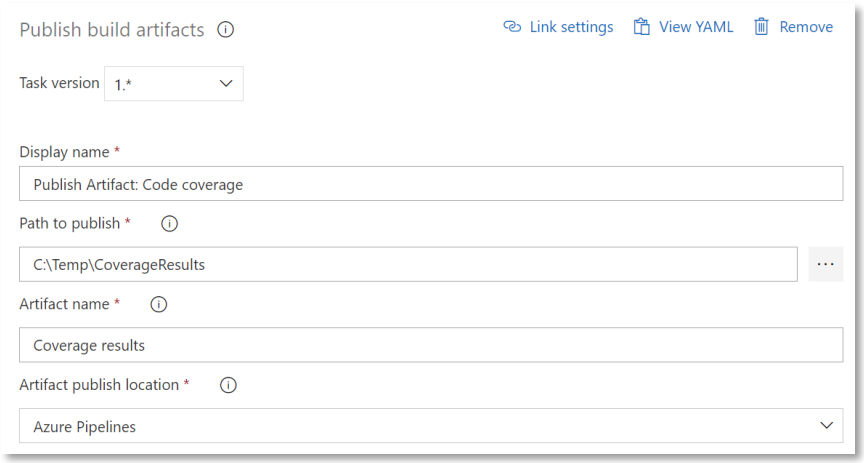


Figure 85: Publish build artifacts parameters

15. To queue the pipeline on the build agent, click **Save & queue**.

Unit test results

The pipeline runs on the build agent and creates a build result that is visualized in the **Runs** tab of the **Pipelines** menu. When you click a recently built item in the **Runs** list, it shows you the unit test results as shown in this example of the build with Custom code coverage enabled:

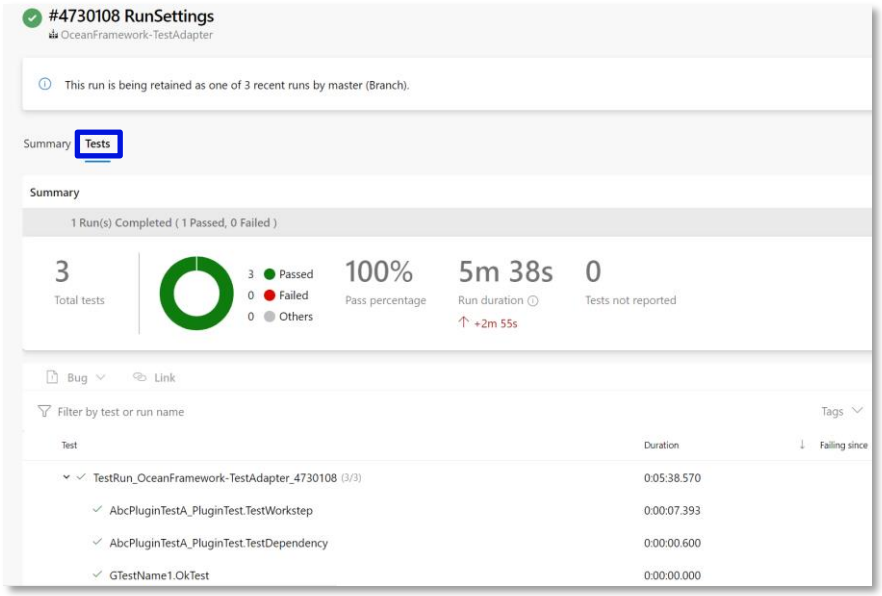


Figure 86: Pipeline unit test results

As you can see in Figure 86 the Custom code coverage summary XML file generated by the Techlog Test Adapter isn't exploitable directly by Azure DevOps and can't be displayed in this dashboard.

Note: Native code coverage results are exploitable by Azure DevOps.

The Custom code coverage summary XML file is created by the Techlog Test Adapter in the build artifacts.

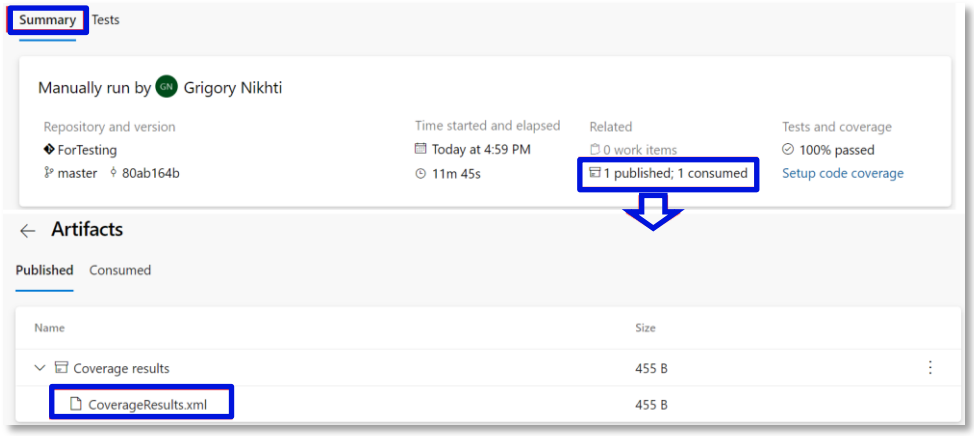


Figure 87: Custom code coverage result file in build artifacts

Code Coverage types

There are two different types or modes to cover your plugin code using unit tests provided by the Techlog Test Adapter:

- Native code coverage.
- Custom code coverage.

Native code coverage is recommended because it is the default Visual Studio code coverage.

Code coverage types differences

There are key differences between two coverage types:

	Custom code coverage	**Native code coverage**
Techlog Version support	2018.1+	2019.2+
Techlog Autorun option enabled	Yes	Yes
Techlog Autorun option disabled	Yes	Not supported
Build Agent: displays native code coverage results in build results (from VSTest task results)	No	Yes
Build Agent: uses special XML file to save coverage results	Yes	No (not necessary)
Supports RunSettings file in terms of Code Coverage	No. Only "Functions Exclude" area (with special syntax) is supported	Yes

Figure 88: Code coverage types differences

Set up and run tests with code coverage: short summary

This is a short summary how to setup and run tests with Code Coverage in Visual Studio and on a build agent:

	Custom code coverage	**Native code coverage**
Set "Linker -> Advanced -> Profile" property of C++ projects to "True"	Yes	Yes/No (depends on the project)
Visual Studio: how to start	Use "Run Tests" option (not "Analyze Code Coverage")	Use "Analyze Code Coverage" option
The following rows are relevant if the Ocean RunSettings file is not used:		
Visual Studio: Enable "Custom Code Coverage" option	Yes	No(should be disabled)
Visual Studio: Using RunSettings file	Optional	Optional
Build Agent: Set TLCODECOVERAGEenvironment variable	Set to "True"	Do not set or set to "False"
Build Agent: Enable "Code Coverage" option in VS Test task	No (should be disabled)	Any value (.runsettings file has a priority)
Build Agent: Using .runsettings file	Optional (to exclude some functions)	Necessary (all assemblies to cover must be placed to "ModulePaths -> Include" section)

Figure 89: Code coverage: setup and run summary

5.Create an installer for your plug-in

The Ocean for Techlog Visual Studio templates deployed by Ocean WIX installer provide a project template that allows plug-in developers to package their plug-ins through a WIX installer. You must have WIX 3.11 or a higher version installed to use the Ocean Plug-in installer template.

Create a plug-in installer

To create a plug-in installer project using Visual Studio:

Add a new plug-in installer project to the solution that contains the Ocean plug-in project that you want to package by right clicking on the solution in the Solution Explorer and selecting **Add > New Project**. In the Project types area, under **Visual C++** project type, select **Ocean > Techlog 2024.4**. Then select the **Ocean Plug-in Installer** template.

Note: An installer project cannot be created in an empty Visual Studio solution. The installer project wizard uses the main plug-in project in the solution.

Provide the name "MyFirstPluginInstaller" for the project. Click the **OK** button to start the Wizard. (See Figure 90)

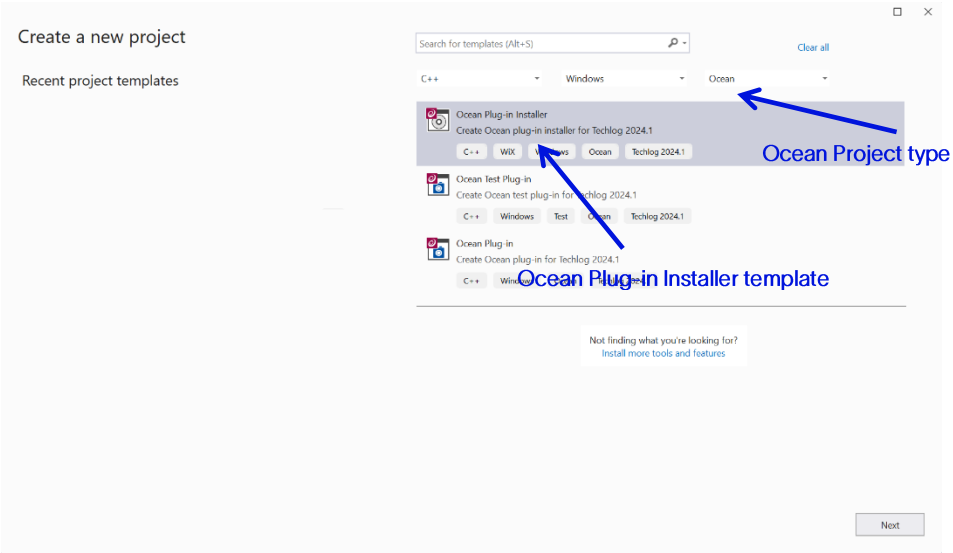


Figure 90: New project window

The plug-in installer wizard appears (See Figure 91).

Set these inputs:

a. **Title:** title of the Ocean plug-in to be installed
b. **Company:** company name that owns the Ocean plug-in; displayed during plug-in installation
c. **Description:** description of the Ocean plug-in; displayed during plug-in installation
d. **Projects:** select the Ocean plug-ins in the solution to package in the installer

Click **Finish** in the dialog.

Note: The plug-in dll and its resources built or copied at the post build in the project output directory are packaged in the plug-in WIX installer by the Ocean plug-in installer wizard.

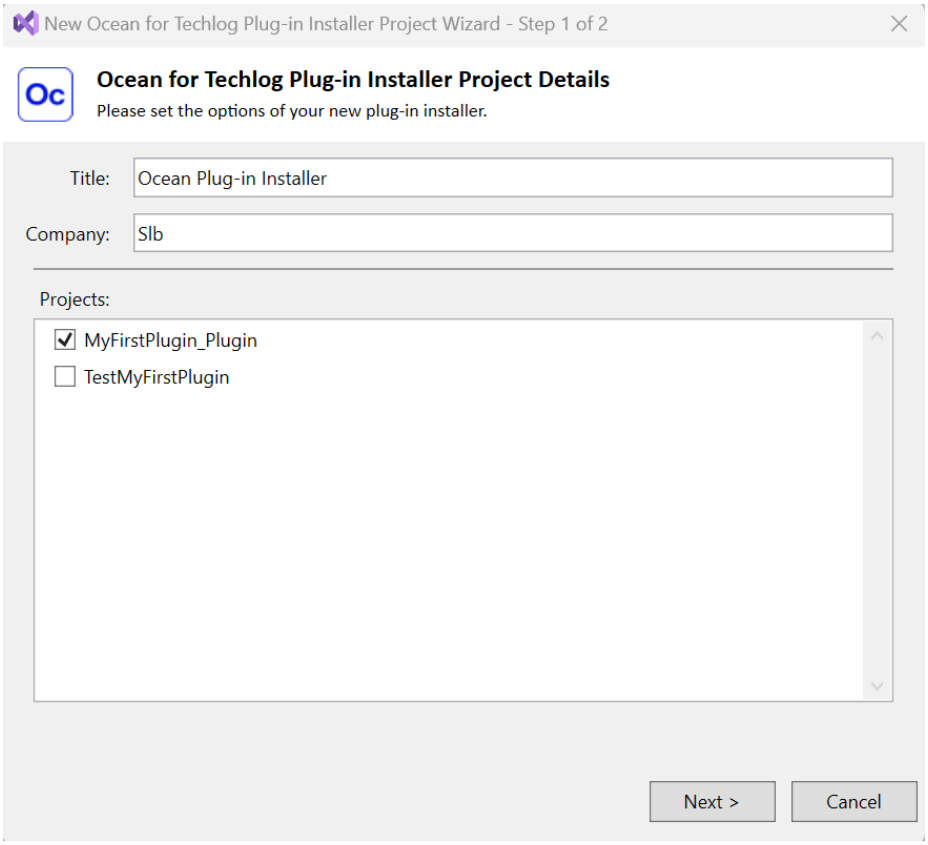


Figure 91: Plug-in installer wizard

The WIX installer project for "MyFirstPlugin" is added to the Visual Studio solution. Build the project; a MSI installer is generated in output of the build. Use it to deploy the plug-in in Techlog to the plug-in users.

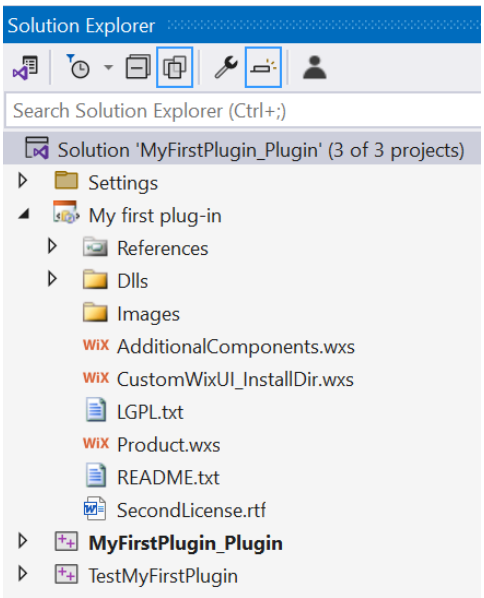


Figure 92: Plug-in installer project

Deploy folders and files with the plug-in dll

You may have to deploy additional files with your plug-in dll that follow a particular folder hierarchy. The WIX installer created through the Ocean plug-in installer template allows you to add those files by editing the **Product.wxs** file. Consider a plug-in activity that creates a **Logview** from a layout template stored at the plug-in level.

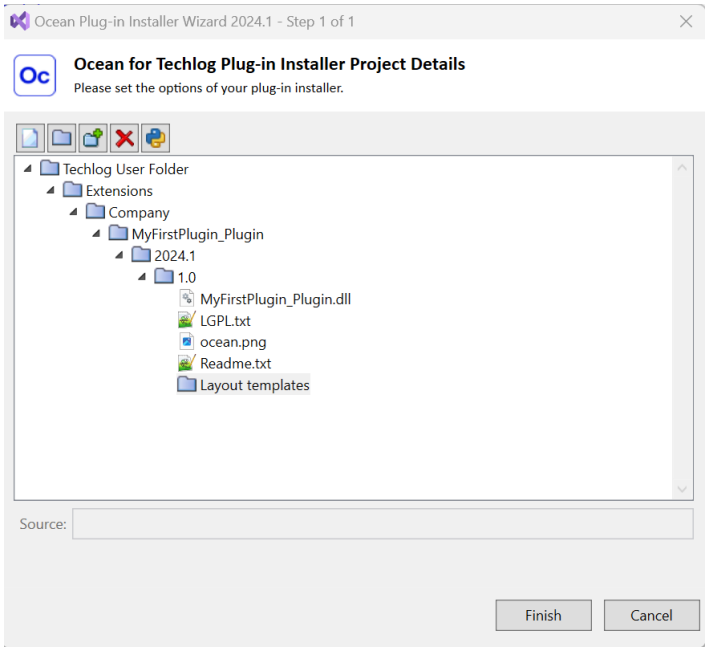
```
void SetupLogviewActivity::run()
{
  Lock lock = LOCK_CREATE_AND_ACQUIRE_ALL(lock);

  Project project = Session::current().mainProject();

  Workspace workspace = Session::current().currentWorkspace();

  // Apply the template for all the wells in the projects
  QList<Well> wells = project.wells().toList();

  LogviewTemplate logviewTemplate =
  LogviewTemplate::get(StorageLevelPlugin, "Well9_short");

  LogviewTemplateDataBinding logviewTemplateDataBinding =
  LogviewTemplateDataBinding(logviewTemplate, wells);

  Logview logview =
  Logview::create(workspace, logviewTemplateDataBinding);

  lock.release();

  stop();
}
```

The layout template **Well9_short.xml** must be deployed with the plug-in dll in a folder named "LayoutTemplates".

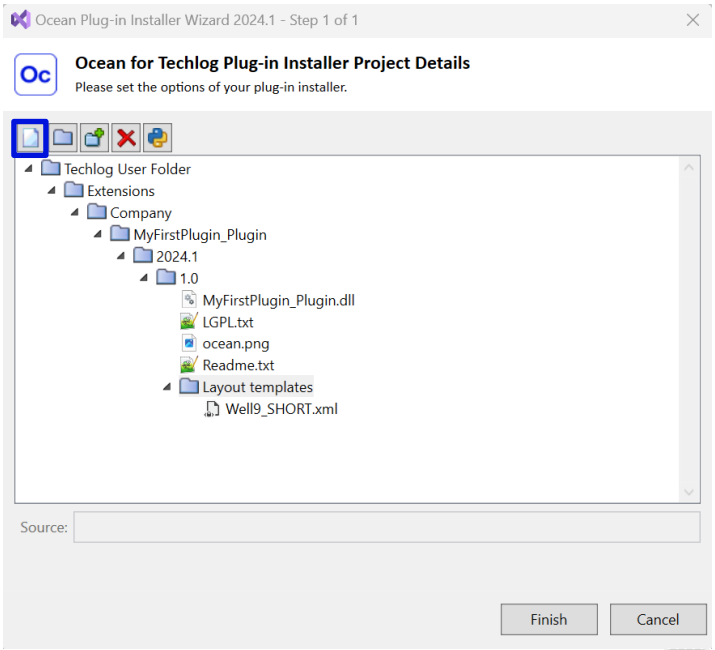
The Ocean plug-in installer wizard dialog allows you to do this.

Browse to a folder on the disk and add it or add a virtual folder that is created at plug-in installation time.

1. Add "LayoutTemplates" folder to plug-in folder



Click on the "Add files" button to browse on the disk and add the **Well9_short.xml** file to the new "LayoutTemplates" folder or drag and drop directly the file from a window explorer to the new "LayoutTemplates" folder.



You can also delete folders and files with the **Remove**  button. When you are happy with folder structure and files that will be deployed with the plug-in, click **Finish** in the dialog. Then the Ocean plug-in installer is added to the Visual Studio solution with folders and files added to **Product.wxs** file.

After the fact the plug-in installer content can be modified opening back the plug-in installer project wizard by right clicking on the plug-in installer project in the Solution Explorer and selecting **Ocean Plug-in Installer Wizard**.

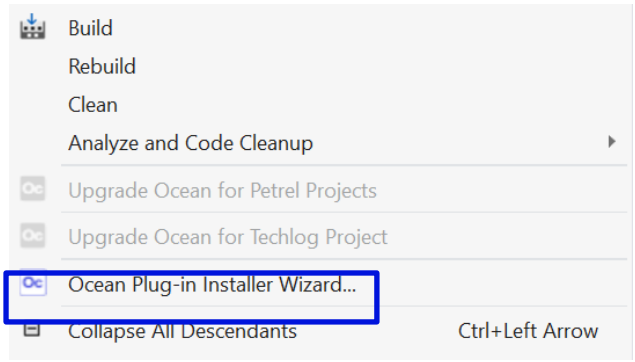


Figure 93: Update plug-in installer project

Plug-in WIX installer designed with Techlog deployment options

A plug-in end user can deploy a plug-in at the Techlog, Company or user level.

This deployment option is clearly exposed in the destination folder window of the WIX installer wizard as shown in the screenshot below:

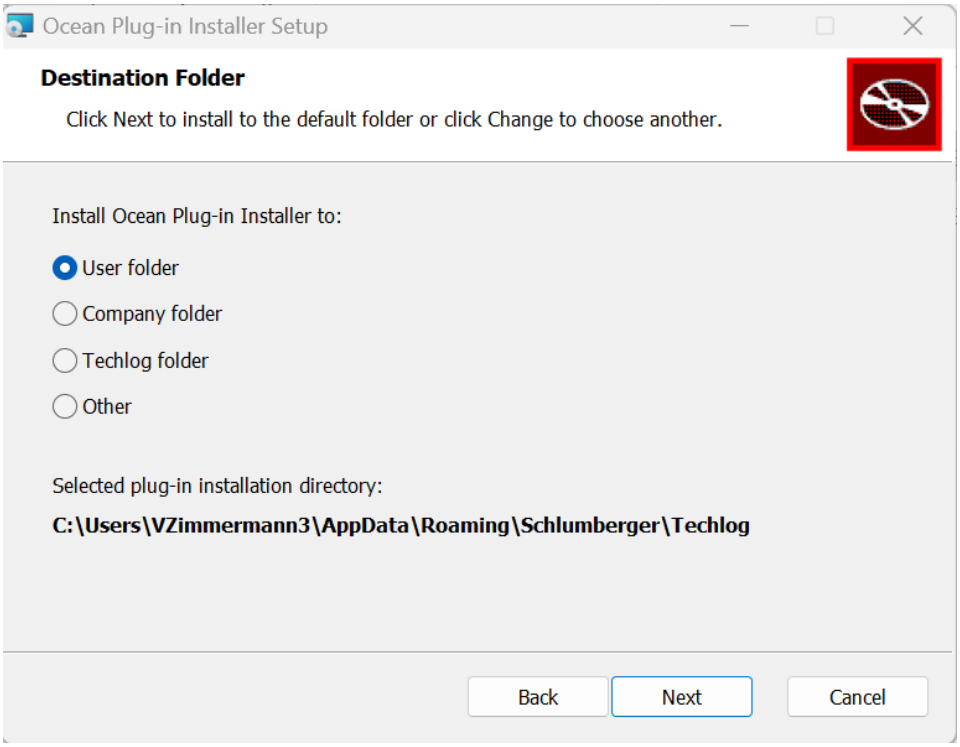


Figure 94: Techlog deployment options

Change the license agreement text of the plug-in installer

A plug-in installer is created with a default Lorem ipsum text that shows up as the plug-in license agreement text during the plug-in installation.

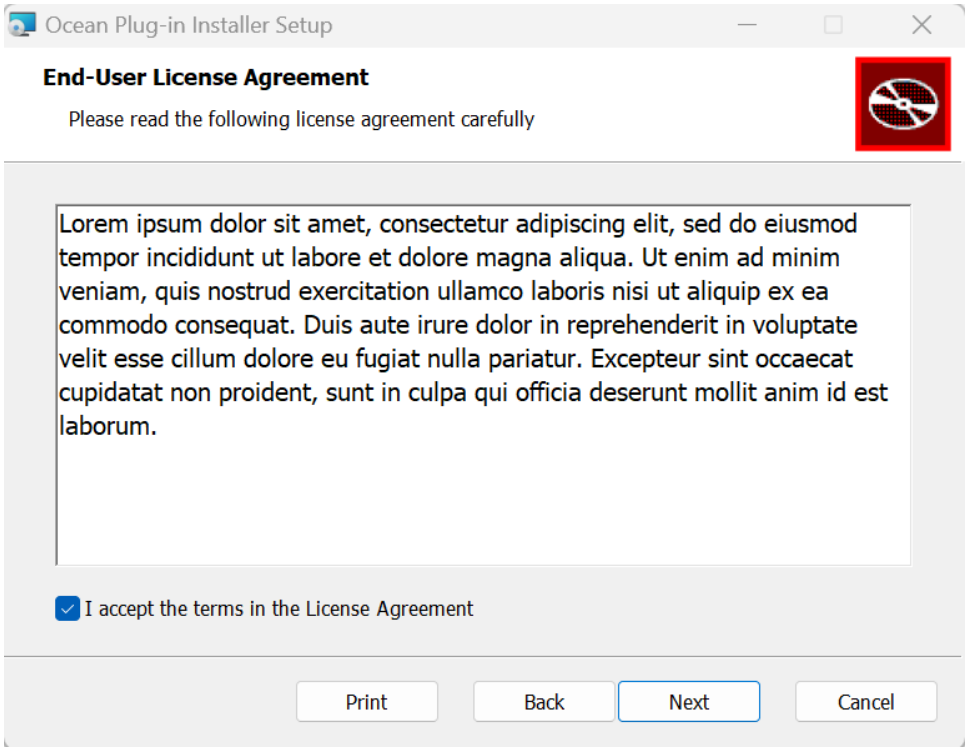


Figure 95: Default "Lorem ipsum" license agreement text

In order to replace the default Lorem ipsum text by your plug-in license agreement text, you have to:

1. add a **License.rtf** file that contains you license agreement text at the root folder of the WIX project
2. add the **License.rtf** file to the WIX project in the Visual Studio solution
3. build your solution

User folder versus company folder deployment

A plug-in is uniquely identified in the module manager by its key **VendorName/PluginName/TechlogVersion/PluginVersion.**

This information is set in the code of the plug-in (plug-in information of the main plug-in class) and the plug-in information has to match the plug-in folder structure:
Extensions/VendorName/PluginName/TechlogVersion/PluginVersion.

See "Writing the plug-in" section for details on how to declare plug-in information.

If two plug-ins with the same key are deployed in user and company folders, they show up both in the Techlog module manager.

You can see in the information pane of the Techlog module manager at which level the plug-in is deployed.

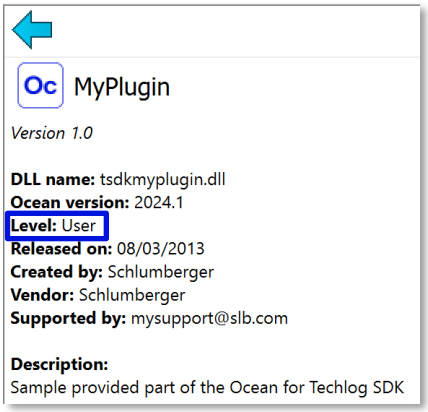


Figure 96: Level where the plug-in is deployed

Deploy a Python package with the plug-in

A plug-in can deploy to its plug-in folder a Python script that can be called from Ocean plug-in code. See the "Call Python script from an Ocean plug-in" section in *Ocean for Techlog Developer Guide - Basics* for more information on how to call a Python script with Ocean framework API.

In some cases the Python script deployed with the plug-in can import Python modules for which the corresponding Python packages aren't installed in Techlog. This makes that the Python script can't be run by the Ocean plug-in if the user doesn't install before the missing Python packages in the Techlog installation folder.

Since 2020.2 release the Ocean plug-in installer template and wizard deployed on top of Visual Studio by the Ocean framework allows you to add Python packages to the WIX installer that are deployed with your plug-in in the plug-in folder. Then any Python scripts in Techlog are able to import modules from the Python packages that are deployed at the plug-in folder level.
To add a new Python package that will be deployed with the plug-in clicks on the "Edit python packages for selected plug-in" button in the toolbar of the plug-in installer wizard. Then add to the list of Python package the name and version of the package that you want to deploy with the plug-in.

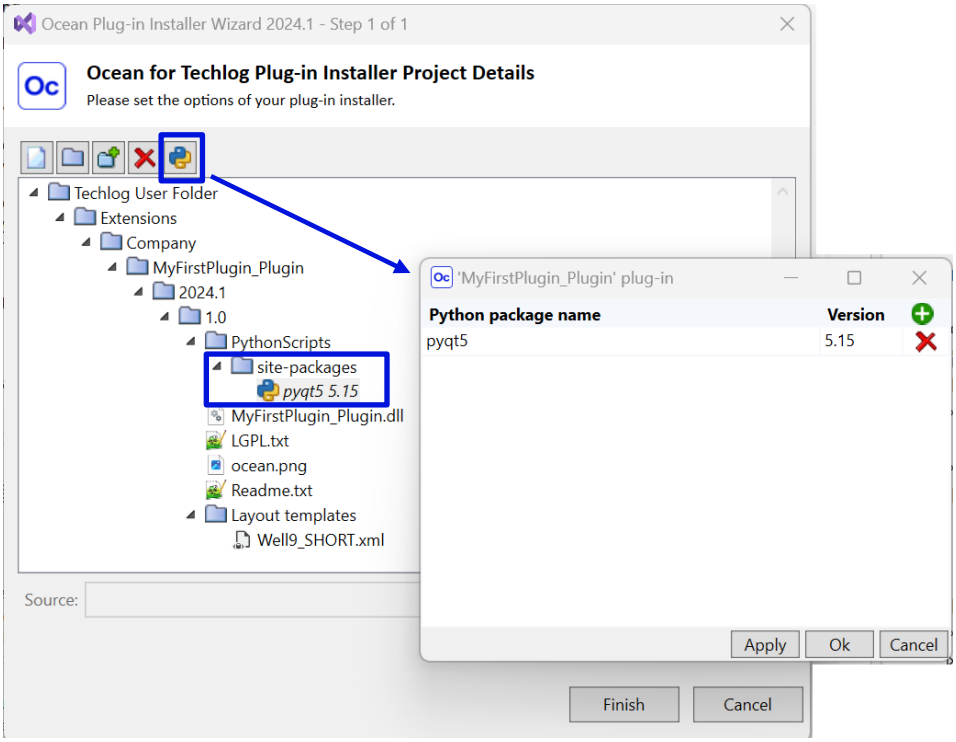


Figure 97: Add a Python package to the plug-in installer

As shown in the above screenshot the Python package is deployed under the **PythonScripts\site-packages** folder of the plug-in folder.

Note: The Python package won't be deployed at the plug-in level if one version of the package is already deployed at the Techlog level.
The version of the package to be deployed is optional. If there is no version specified the latest version of the Python package is deployed at the plug-in level.

6.Upgrade an existing Ocean plug-in to the current Ocean release

It isn't a prerequisite to uninstall previous Ocean framework versions before installing the 2024.4 release. You may have several Ocean framework versions installed on your machine.
If you open an Ocean plug-in project created with a previous Ocean template and wizard version, you can upgrade this project to the current Ocean framework version by right clicking on the project in the Visual Studio **Solution Explorer**. Then right click **Upgrade Ocean for Techlog Project** in the context menu.

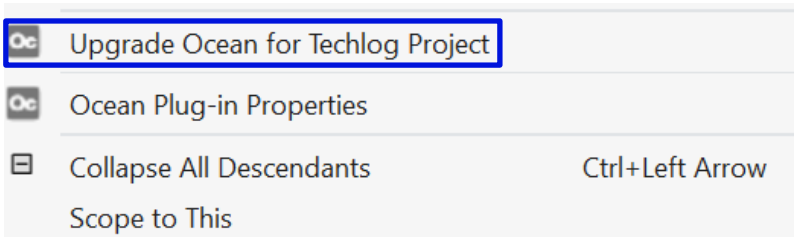


Figure 98: Upgrade Ocean for Techlog project

The upgrade window opens, asking you to confirm the project upgrade.

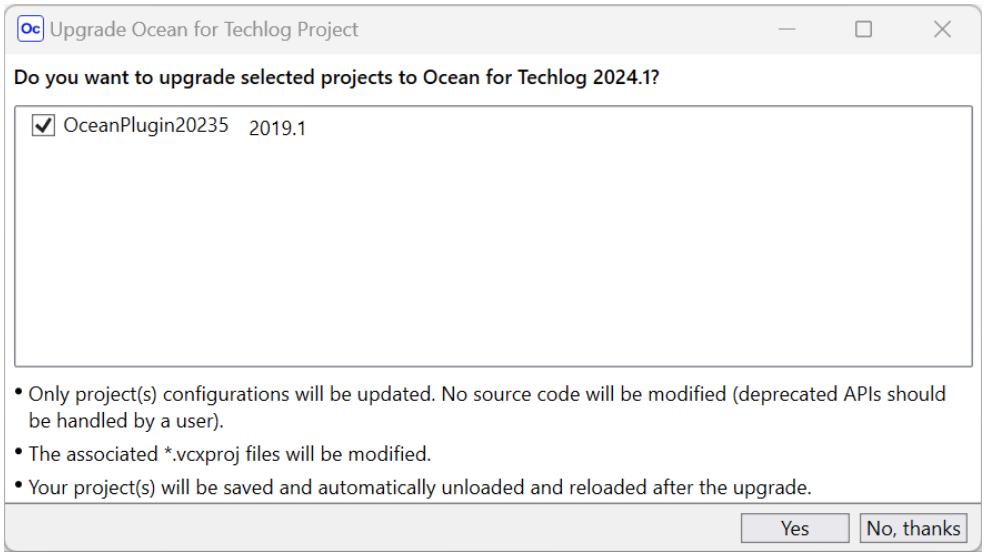


Figure 99: Upgrade window

Then an information message warns you about changes that have been applied to the Ocean for Techlog plug-in project.

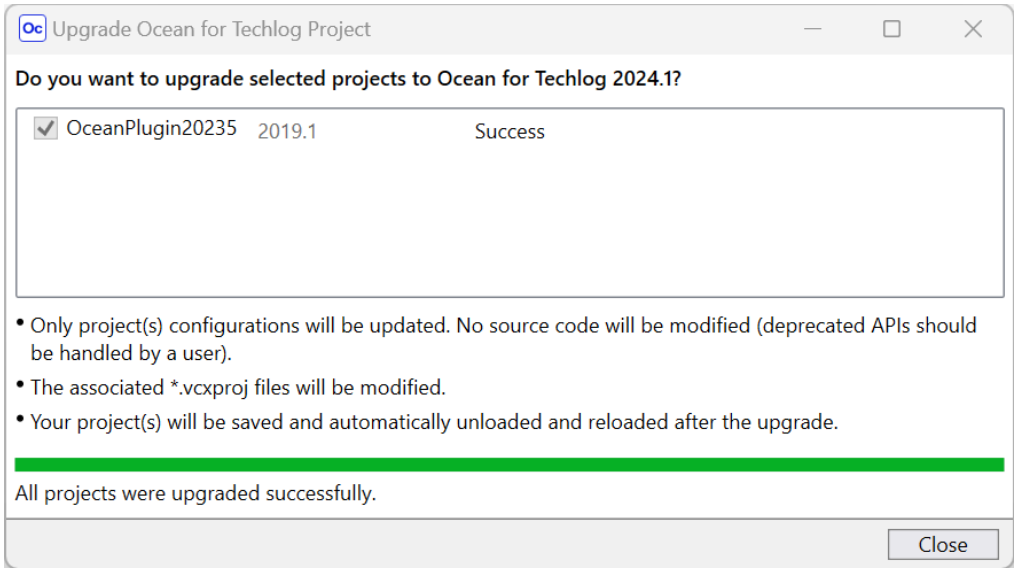


Figure 100: Upgrade information message

The Ocean for Techlog properties are automatically upgraded to the new release during the upgrade process. Those properties are displayed in the **Ocean plug-in properties** dialog window opened from the Visual Studio project contextual menu.

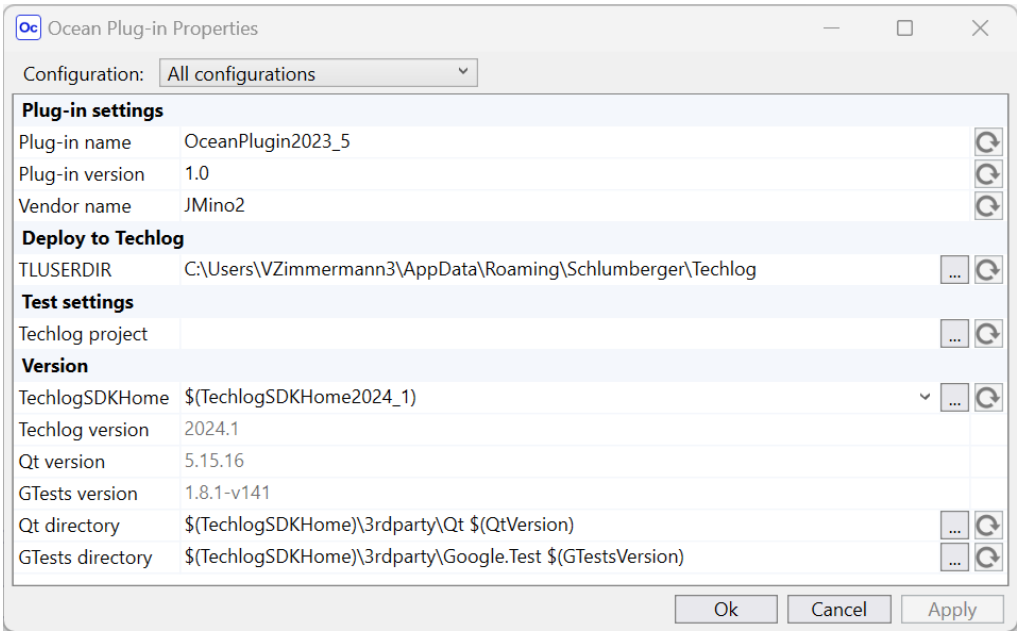


Figure 101: Ocean for Techlog project properties

In this dialog window you have access to four groups of properties:

1. **Plug-in settings:** allow you to give mandatory plug-in information values like plug-in name, version and vendor name. Changing those values will deploy the plug-in output dll with the corresponding plug-in structure folder at post-build time.

2. **Deploy to Techlog:** allow you to change the path to the Techlog user folder where the plug-in is deployed at the post build time
3. **Test settings:** allow you to specify the path to a Techlog project that will be used to run Ocean unit tests in Techlog through the Techlog test adapter.

4. **Version:** this group of properties allows you to handle the Ocean for Techlog binary versions (TechlogSDKHome), Qt binary versions (QtDir) and Google tests binary versions (GTests) with which you want to build your plug-in. If you create an Ocean project plug-in from a 2024.4 template or upgrade an Ocean plug-in project to the 2024.4 release, the default values for those properties are the TechlogSDKHome2024_4 environment variable value, QTDIR and GTests values stored in property sheets installed with the Ocean framework 2024.4.

If you change those values through the Ocean plug-in properties dialog, the new values are only set at the user level and the TechlogSDKHome environment variable value or other properties defined in property sheets at the project level remain unchanged.

Techlog version, Qt version and Gtests version are linked respectively to the Ocean framework version, Qt libraries version and Google tests libraries version both deployed with the Ocean package. This means that you can't change manually those properties through the Ocean plug-in properties window.

See the "Ocean for Techlog property sheets" section for more information on Ocean properties deployed with the Ocean framework package.