

Getting Started



Ocean Software Development Framework for Techlog
Version 2016

Copyright © 2006-2016 Schlumberger. All rights reserved.

This work contains the confidential and proprietary trade secrets of Schlumberger and may not be copied or stored in an information retrieval system, transferred, used, distributed, translated or retransmitted in any form or by any means, electronic or mechanical, in whole or in part, without the express written permission of the copyright owner.

Trademarks & Service Marks

Schlumberger, the Schlumberger logotype, and other words or symbols used to identify the products and services described herein are either trademarks, trade names or service marks of Schlumberger and its licensors, or are the property of their respective owners. These marks may not be copied, imitated or used, in whole or in part, without the express prior written permission of Schlumberger. In addition, covers, page headers, custom graphics, icons, and other design elements may be service marks, trademarks, and/or trade dress of Schlumberger, and may not be copied, imitated, or used, in whole or in part, without the express prior written permission of Schlumberger. Other company, product, and service names are the properties of their respective owners.

An asterisk (*) is used throughout this document to designate a mark of Schlumberger.



Contents

Welcome to Ocean for Techlog	3
Ocean for Techlog Advantage	3
Ocean for Techlog Architecture	4
Access to the Techlog data model	4
Ocean for Techlog UI Infrastructure	5
Ocean for Techlog plug-in identity and activities	6
Ocean framework license	7
Qt LGPL notice	7
Install and setup the Ocean for Techlog development environment	9
Ocean for Techlog installation	9
Ocean for Techlog package content	12
Ocean for Techlog environment variables	13
Test the Ocean for Techlog development environment	14
Writing your first plug-in	19
Writing the plug-in	19
Creating the Plug-in and Activity with Visual Studio	19
Inspecting the files	22
Plugin	23
Activity	27
Writing the algorithm code	27
Running the plug-in	30
Debug the plug-in	32
Techlog Viewer plug-in development (Schlumberger Internal only)	34
Techlog Viewer specific	34
Signed plug-ins	34
Create unit tests for your plug-in	34
Creating a Test plug-in with Visual Studio	35
Inspecting the files	36
Implement the tests	37
Run the tests	42

Create an installer for your plug-in	46
Deploy folders and files with the plug-in dll	48
User folder versus company folder deployment.....	51
Upgrade existing Ocean plug-in to 2016.1	52

Welcome to Ocean for Techlog

Ocean for Techlog is an application development framework, part of the Ocean suite of Schlumberger software platform SDKs, 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 the 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 as worksteps which run within a Techlog workflow.



All the code snippets in this document have been built with Ocean for Techlog 2016.1.

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 plugins to interoperate with Techlog functionalities. Figure 1 shows how Ocean for Techlog provides the glue between Techlog and the plugins.

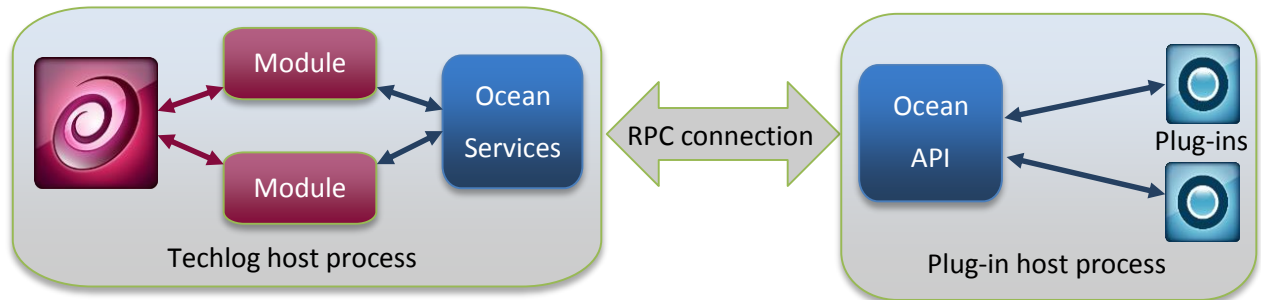


Figure 1 Ocean for Techlog architecture. Plug-in isolation.

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, recompile and rerun of 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 can access the following data types and properties of the Techlog data model:

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

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.

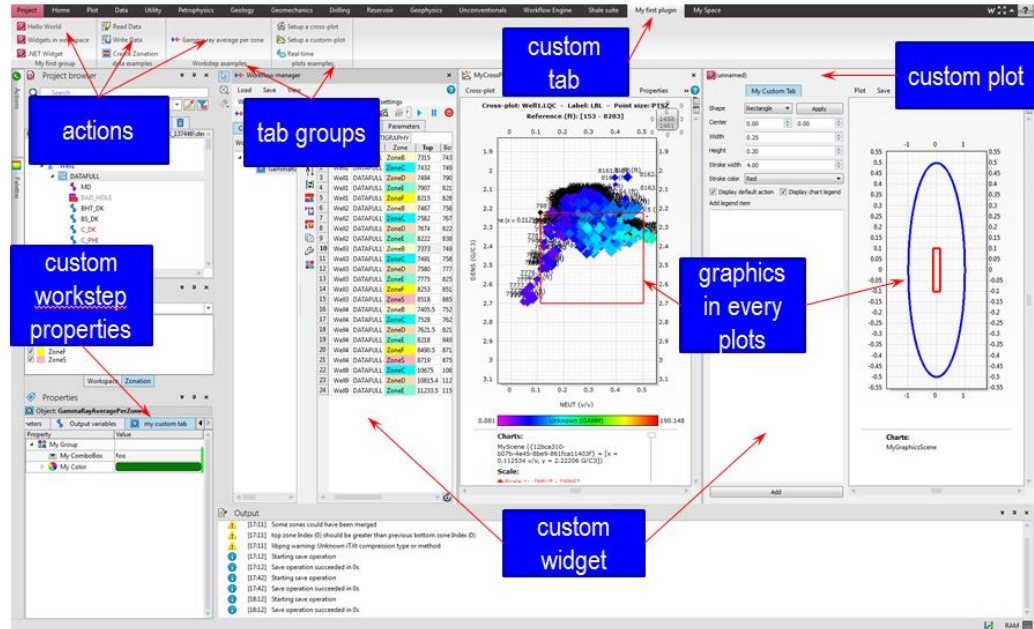


Figure 2 Techlog UI extensibility

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

- Menu bar extensions:
 - Adding new tabs, groups and menus to the tbar (Techlog ribbon) or extending native Techlog menus
- Windows:
 - Adding custom windows (Qt widgets) in Techlog workspace
- Plots:
 - Adding custom plots
 - Customize native and custom plots adding graphic items
 - Add user interactions through graphic items
 - Extend menu bar, tool bar and context menu of native and custom plots with custom tools
- Workflow manager
 - Add custom user interface to an Ocean workstep
 - Extend workstep properties (Techlog properties editor) with custom properties tab

Ocean for Techlog plug-in identity and activities

The `PluginIdentity` is an interface that the developer has to implement 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 (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 **Project > Licensing > Module Manager** menu.

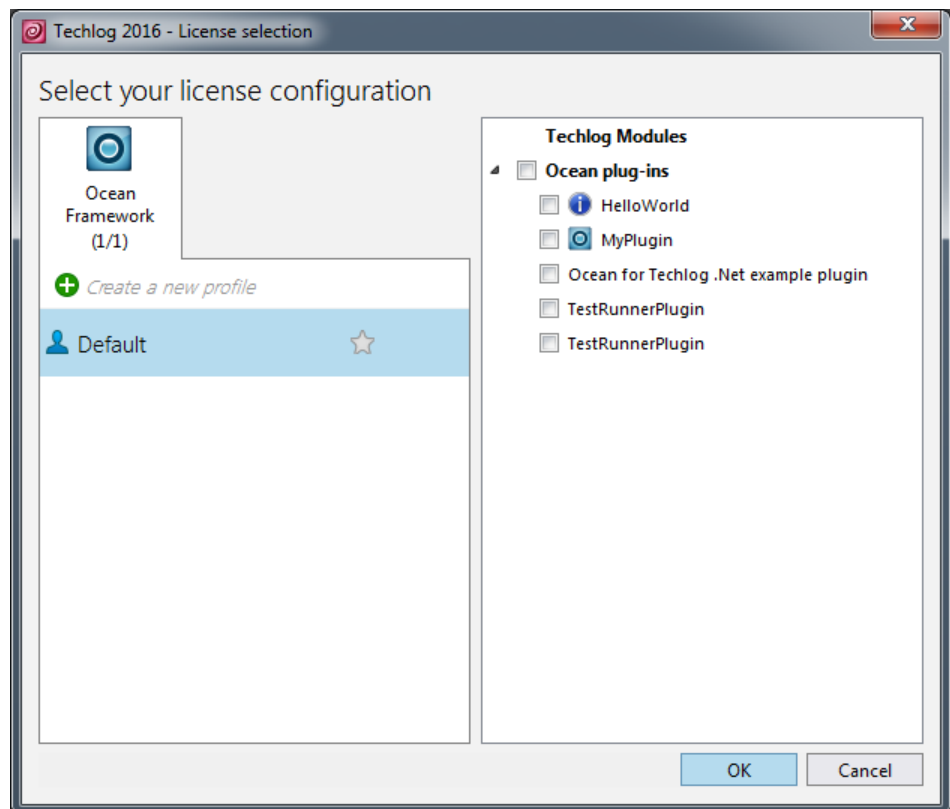


Figure 3 Techlog module manager

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

In Techlog, a module is a set of functionalities associated to a license feature. A plugin can contribute its Activities into some Techlog Modules. For instance, a Plugin can contribute an environmental correction workstep (associated to the environmental correction license), and can also add some geology-related processing to the WBI (wellbore imaging) module, that is to say 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.

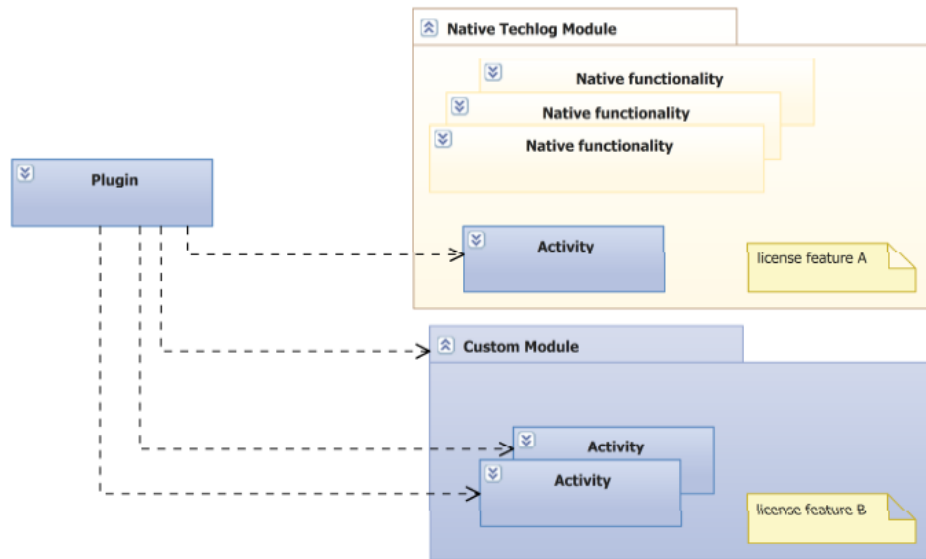


Figure 4 Plugins contribute activities to modules (native or custom). Modules can be licensed.

All the activities of a given plugin run in a single process, and multiple instances of a given activity can run in that same process. This way, activities within a given plugin 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 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

Ocean framework is distributed with Qt LGPL 5.4.2 libraries. Per requirement of LGPL components used, you must provide with your plug-in a notice that LGPL code is being used. This can be done by deploying with your plug-in dll (plug-in folder) **README.txt** and **LGPL.txt** files shipped with the Ocean framework.

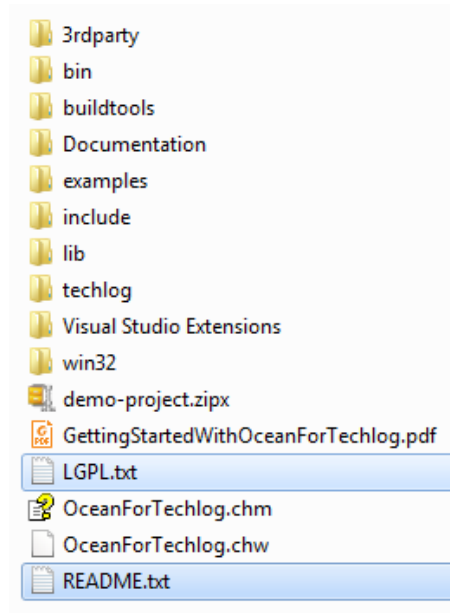


Figure 5 LGPL notice files

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.

Install and setup the Ocean for Techlog development environment

Ocean for Techlog installation

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 can be located anywhere on the disk.

1. Browse the installation folder and click **Next** in the dialog window. (See Figure 6.)

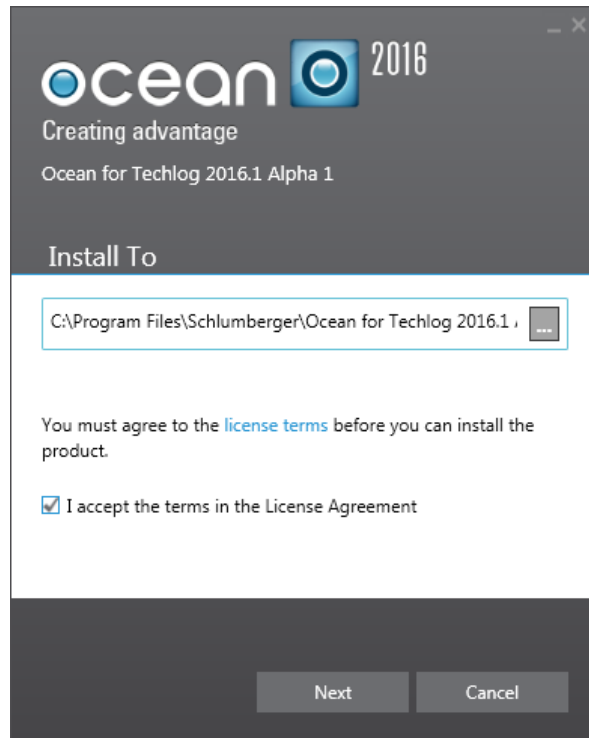


Figure 6 Ocean for Techlog install location

The installer checks:

- corresponding Techlog version is installed
 - Visual Studio 2012, 2013 or 2015 is installed
2. Click **Next** in the dialog window. (See Figure 7.)

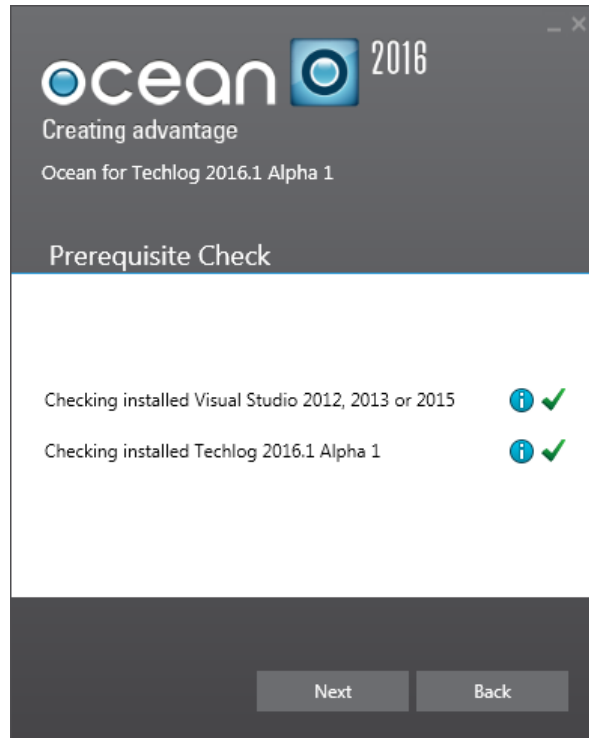


Figure 7 Techlog and VS 2012, 2013 or 2015 installed

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

Note: If you have already a **QTDIR** environment variable defined on your system and pointing on Qt version installed on your machine, the value of this environment variable is replaced by the path to Qt folder deployed with Ocean for Techlog package.

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.

3. Select all components and click **Install** in the dialog window. (See Figure 8.)

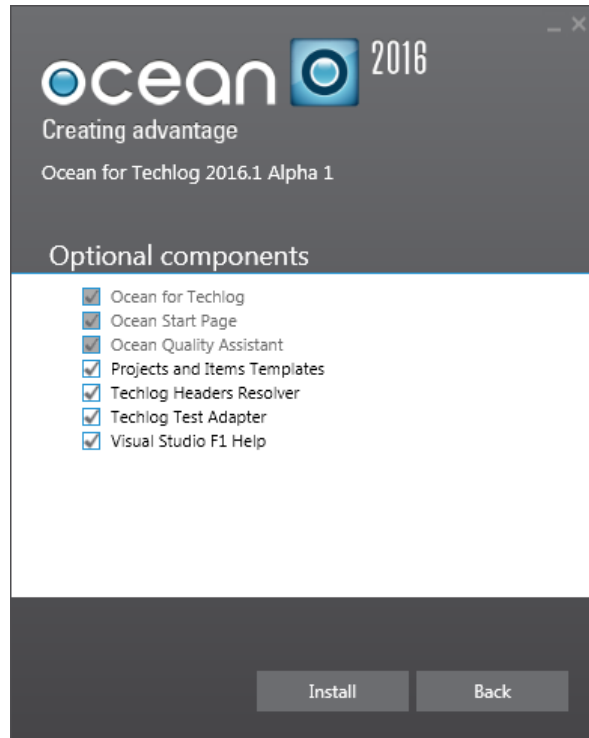


Figure 8 Visual Studio components

4. Reboot is required to get all Ocean for Techlog Visual Studio extensions properly installed. (See Figure 9.)

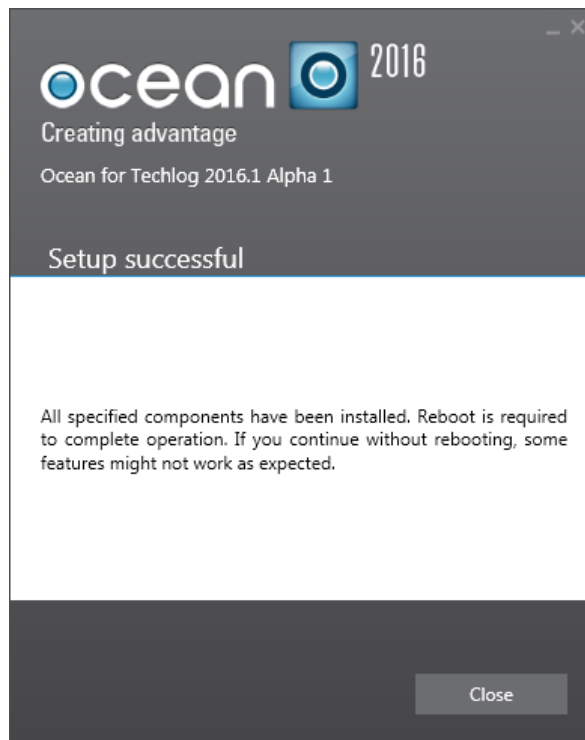


Figure 9 Reboot required

Ocean for Techlog package content

Ocean for Techlog Framework is deployed by the installer. The Ocean for Techlog package will have the following folders tree installed on your disk when installed:

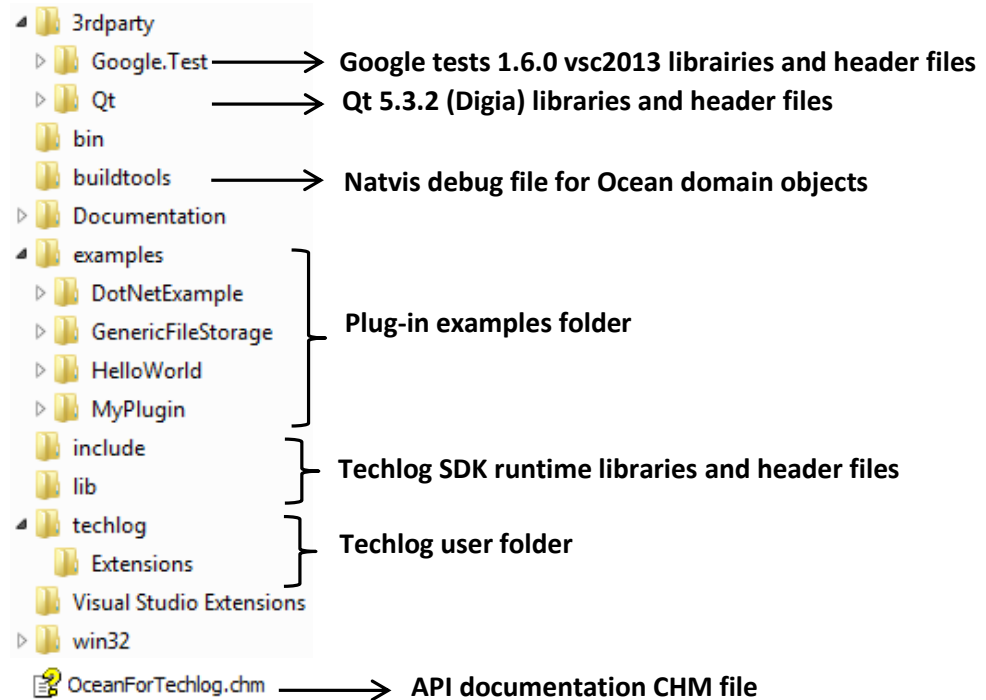


Figure 10 Ocean for Techlog package content

Plug-ins are built on Qt. The Ocean for Techlog Framework installer comes with the LGPL 5.4.2 version of Qt and QtCore and QtGui libraries (the 2 most basic Qt libraries).

The Ocean for Techlog API exposes the following 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 tests 1.6.0 mvsc2013 x64 librairies are provided with Ocean framework in order to create 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.

All libraries needed to develop plug-ins with the Ocean for Techlog framework are shipped with the installer and are installed under the **3rdparty** folder.


The **examples** folder includes the following plug-ins:

- **HelloWorld**: a simple plug-in useful to test your Ocean for Techlog development environment.
- **DotNetExample**: showing how to integrate a .NET library in Techlog using Qt and Ocean framework.

- **MyPlugin:** some code examples of each API exposed in Ocean for Techlog
 - Read and write data access
 - Creating workstep, add it and run it in a Techlog workflow
 - Plot examples as Logview, cross-plots, custom plots
 - Custom UI examples
- **GenericFileStorage:** some plug-in domain objects (custom domain objects) code samples

The **Extensions** folder contains the compiled plug-in examples listed previously and can be used as deployment folder during the development phase of your plug-ins. For that you need to create or modify the **TLUSERDIR** environment variable to point the **User** folder to the parent of the **Extensions** folder of the package. This is described in the next section.

The same known **Extensions** location can be added within Techlog's multi-level folder organization: **Techlog, Company** and **User**. This allows the plugin to be deployed along with the Techlog installation, or on the Company's shared drive to reach many users, or just by an individual.

 It is not recommended for a plug-in developer to deploy a plug-in directly at the company or Techlog level for the following 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 at least two environment variables which are:

- **TechlogSDKHome** is the root folder path where the Ocean for Techlog framework is installed on your disk (e.g. *D:\OceanForTechlog\SDK*).
- **QTDIR** used to build plug-in with Qt libraries. If you use the Qt libraries shipped with the package in the third-party folder, the path can be set as follows
%TechlogSDKHome%\3rdParty\Qt

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

If there is no user folder sets on your machine, the installer sets the **%AppData%\Schlumberger\Ocean for Techlog 2016.1\techlog** folder as user folder and deploy sample plug-ins in this folder. Sample plug-ins code is also deployed in the **%AppData%\Schlumberger\Ocean for Techlog 2016.1\examples**

You can change it anytime through Techlog Options window (**Project > Options**).

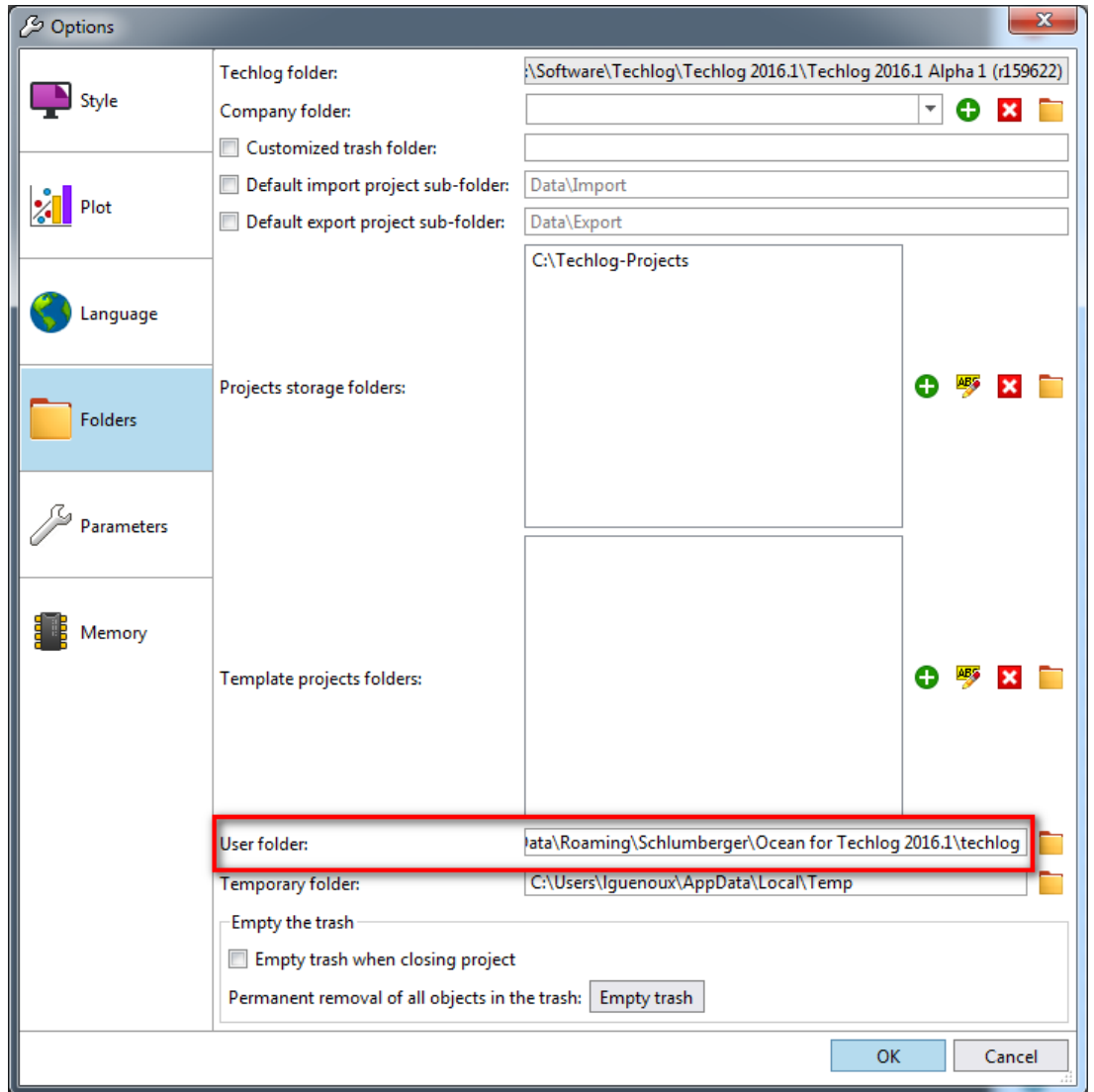



Figure 11 Techlog user folder

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

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

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

Test the Ocean for Techlog development environment

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

1. Run Techlog and open the module manager from the **Project > Licensing** menu:

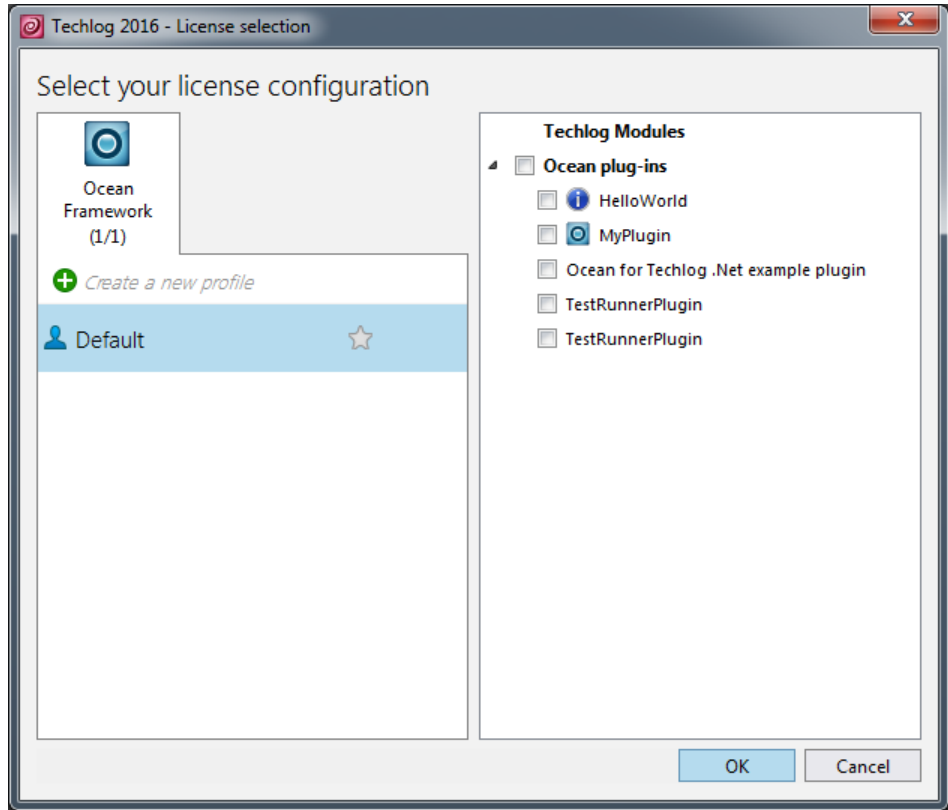


Figure 12 Release plug-ins deployed with Ocean Framework

The module manager scans the **Extensions** folder of the Ocean for Techlog package and the three example plug-ins built in release mode and shipped with the Ocean framework displayed as in Figure 11.

2. Go to **%TechlogSDKHome%\examples\HelloWorld** folder and run **qmakepluginhelloworld.bat** to create the visual studio project file.
3. Open **HelloWorld.vcxproj** with Visual and build the project in **debug x64** mode.

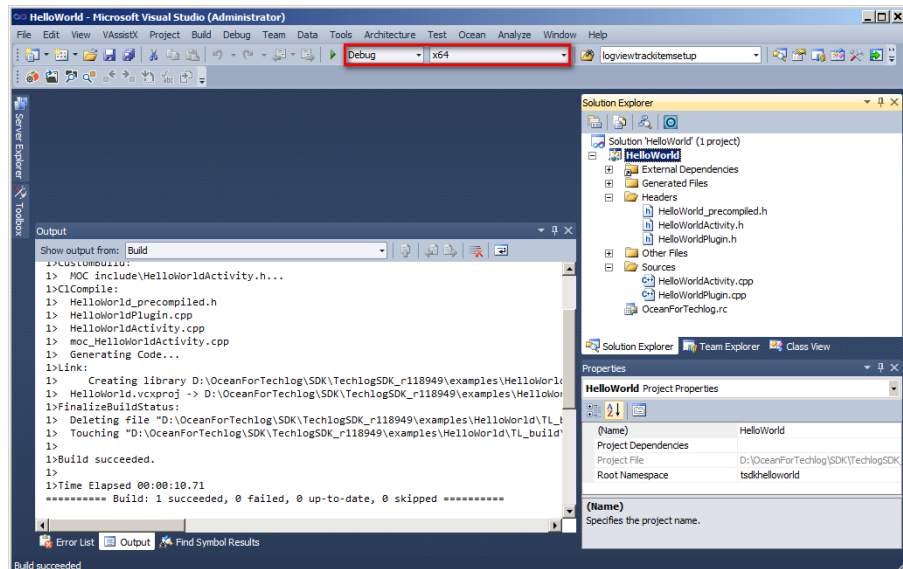


Figure 13 HelloWorld build in debug x64

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

Note: In the following 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.

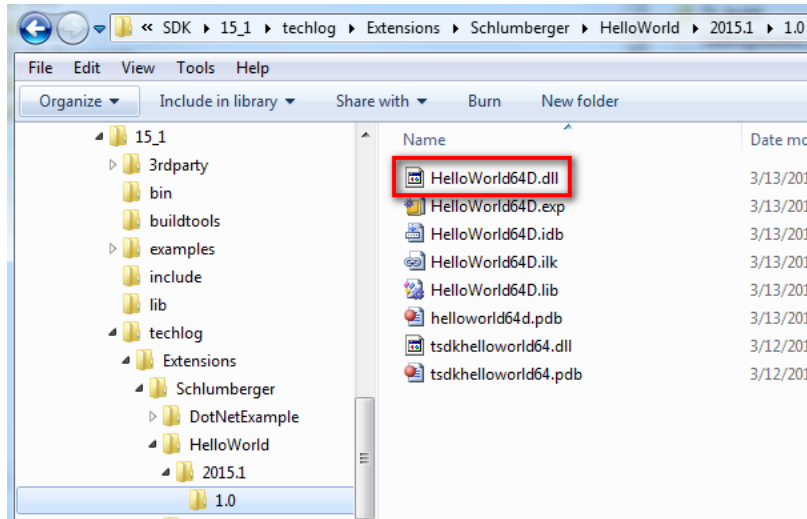


Figure 14 HelloWorld debug x64 library

4. In Techlog open the module manager, right-click on Ocean plug-ins node and click on **Refresh plug-ins** item in the context menu. The new **HelloWorld debug x64** plug-in appears in ocean plug-ins group as in Figure 15.

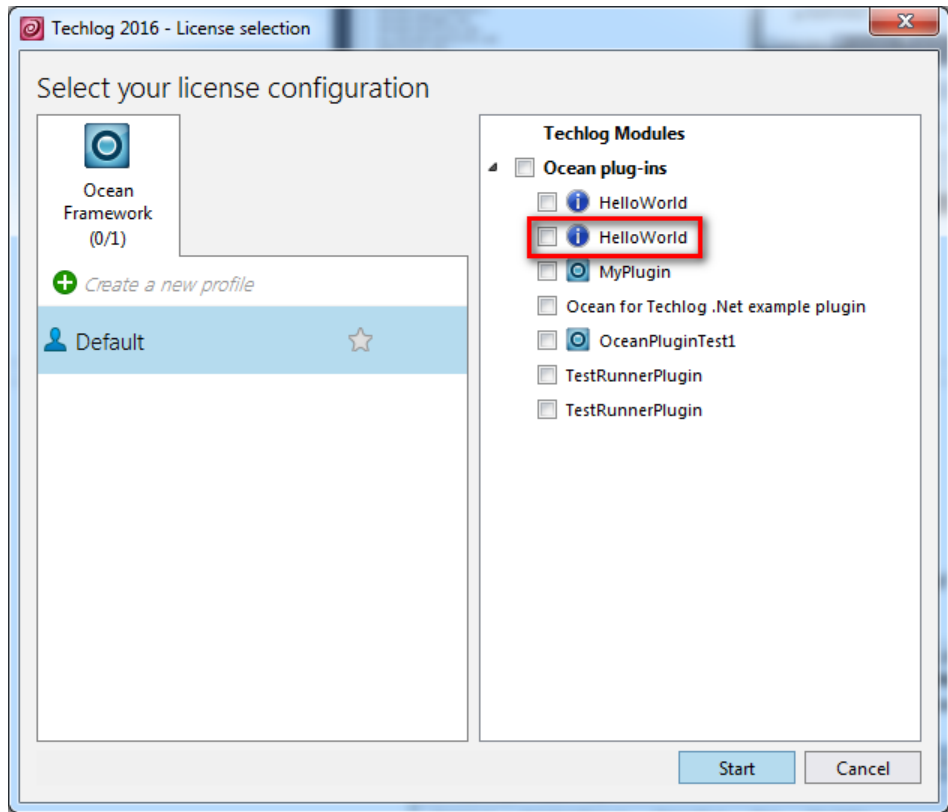


Figure 15 HelloWorld64D plug-in



*If you get the error messages below for some plug-ins built in debug mode when you refresh the list of plug-ins in the module manager it means that 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 Ocean package.*

Error: Plugin 'myplugin64D.dll': can't find corresponding plugin host file.

Error: Can't launch plugin host for plugin 'myplugin64D.dll': host process not running.

5. Enable the plug-in and click on the Hello World action menu in the new HelloWorld plug-in added in Techlog. Hello Plugin World message displays in the Techlog output console.

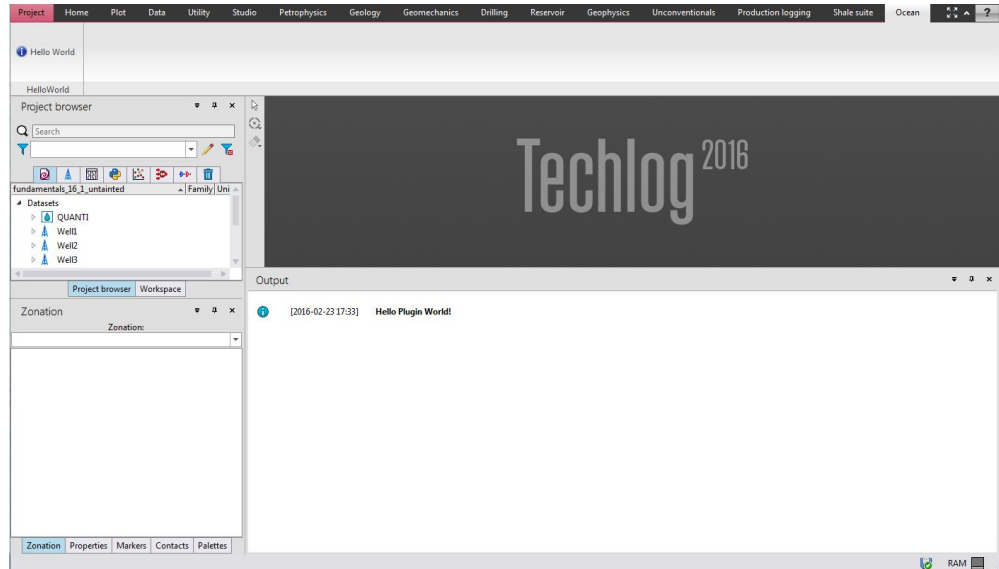


Figure 16 HelloWorld64D activity running

Please review the user folder path in Techlog (or **TLUSERDIR**), **TechlogSDKHome**, and **QTDIR** environment variables if one of these steps does not work properly.



One of the reasons listed below can be the root of your issue:

- *If **QTDIR** is not set correctly, qmake will not create the solution.*
- *If the plug-in dll isn't generated in **TLUSERDIR**, the built plugin will not be loaded.*
- *If the plug-in folder structure **Vendor-Name/PluginName/TechlogVersion/PluginVersion/** isn't respected the plug-in will not be loaded.*
- *If a Debug version plug-in is not loaded, the Debug version of the pluginhost is not present in **bin64/pluginhost** folder of Techlog installation folder.*

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, plugin, 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 2016.1**.

Note: Since Ocean 2016.1, you can have two different versions installed on the user machine, a 2015.1 and a 2016.1.

4. Select the **Ocean Plug-in** template.
5. Provide the name "MyFirstPlugin" for the project.
6. Click **OK** to start the Wizard.

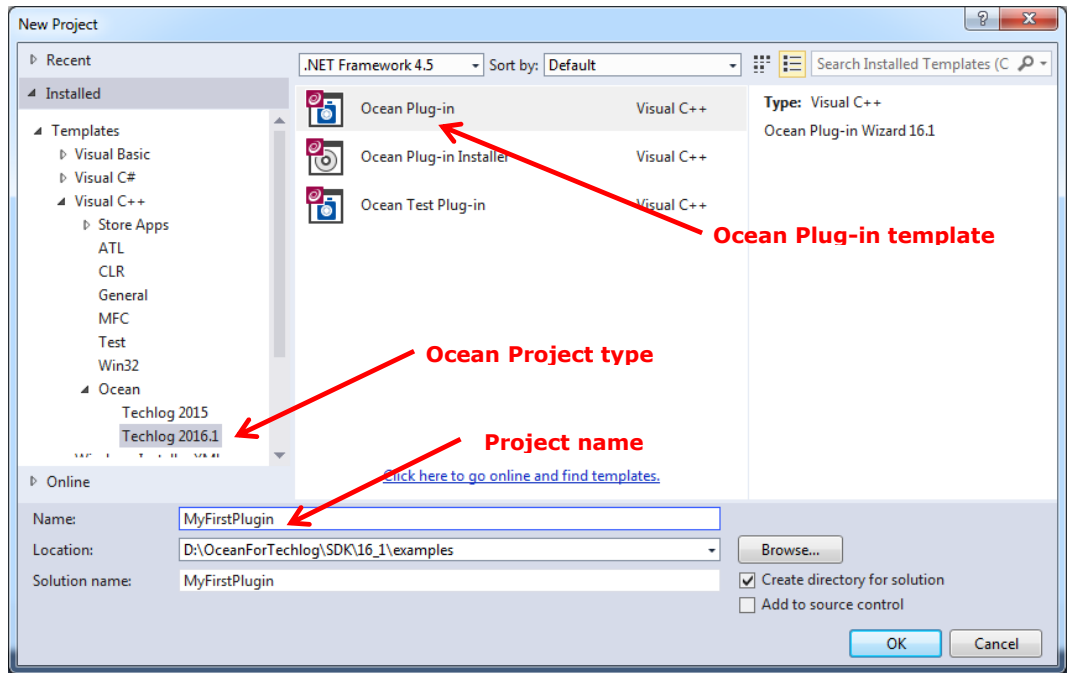


Figure 17 New project window

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 18.).
Note that "Vendor name", "Plug-in name" and "Plug-in version" are mandatory plug-in information.
9. Click **Finish**.

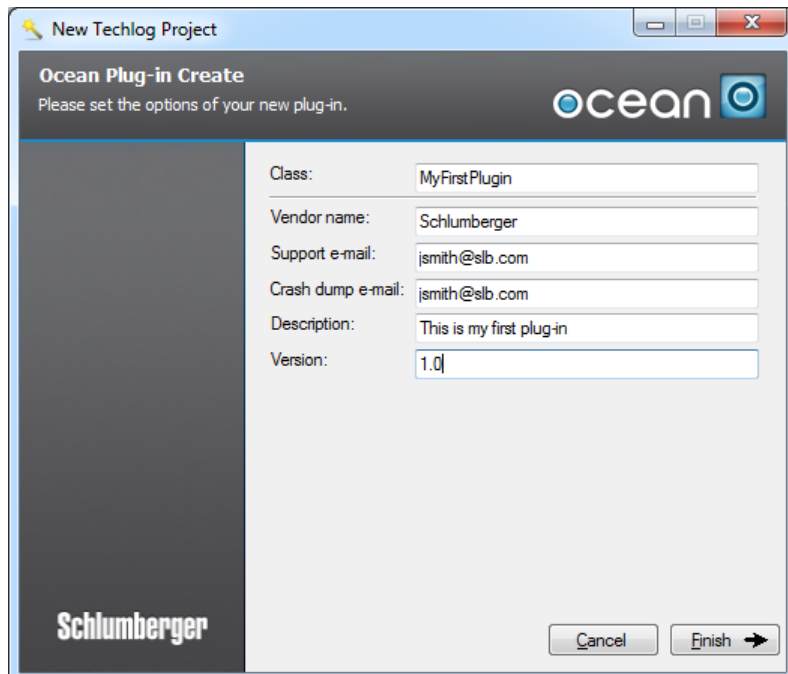


Figure 18 Plug-in wizard

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

10. Add a new plug-in activity by right-clicking on the project in the Solution Explorer and selecting **Add > New Item** in the contextual menu.
11. In the Item types area, under **Visual C++** item type, select **Ocean > Techlog 2016.1**.
12. Select the **Ocean Activity** template.
13. Provide the name "ReadDataActivity" for the activity.
14. Click **Add** in the dialog (See Figure 19.)

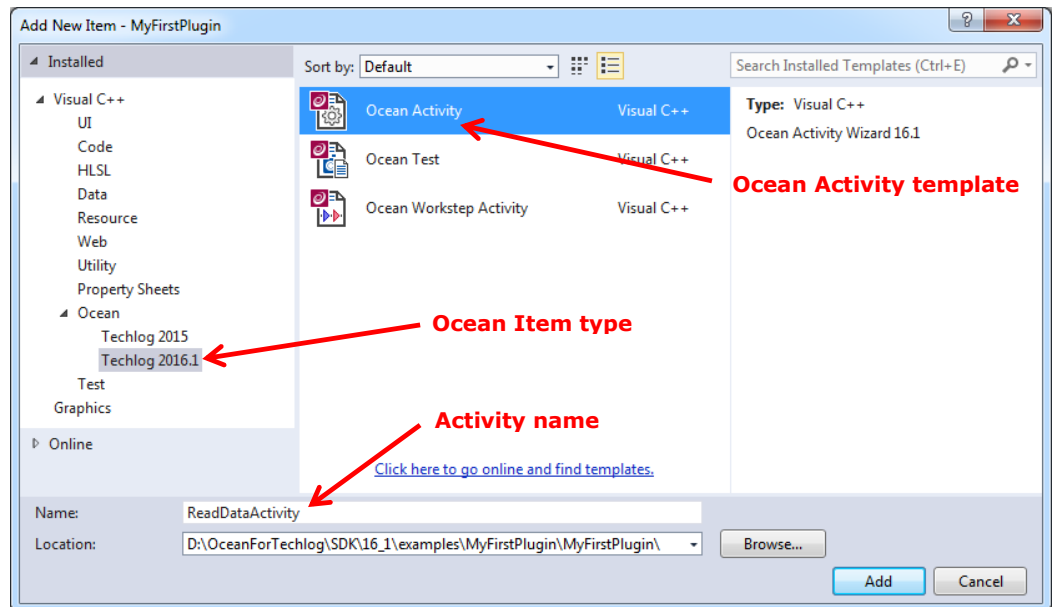


Figure 19 New activity window

Note: From this window you have the ability to create an **Ocean Workstep Activity**. This will add 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 Basics developer guide for more information on how to implement an Ocean workstep.

15. Change the "Tab title", "Group title", "Action menu text" and "Action menu tooltip" fields as appropriate (See Figure 20.).
Note that those fields are used to create the plug-in menu in Techlog toolbar that triggers the Ocean activity.
16. Click **Finish**.

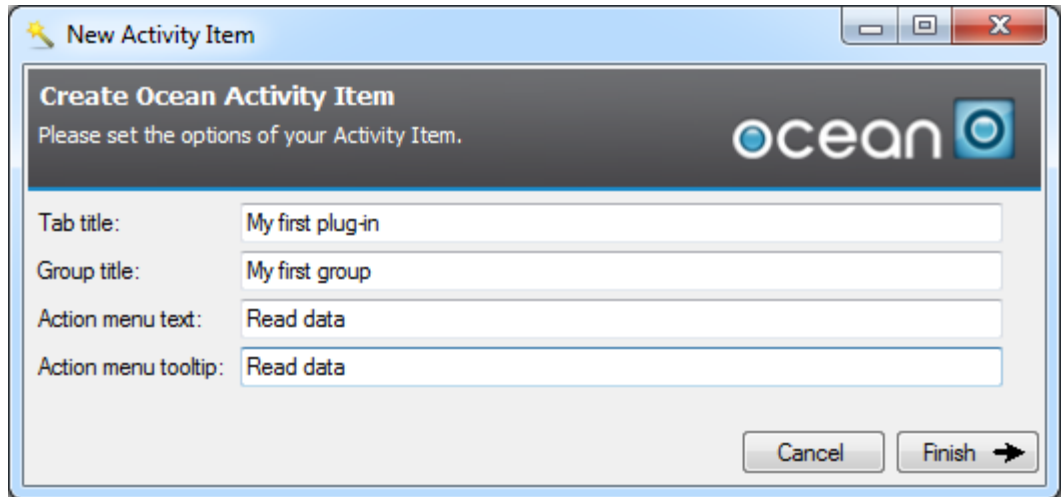



Figure 20 New activity wizard

The wizard adds the activity class to the project.

 *If Intellisense is disabled in Visual Studio 2013, Ocean template items are not accessible and an error message is raised. In **Tools > Options** menu of Visual Studio 2013, **Disable database** has to be turned off.*

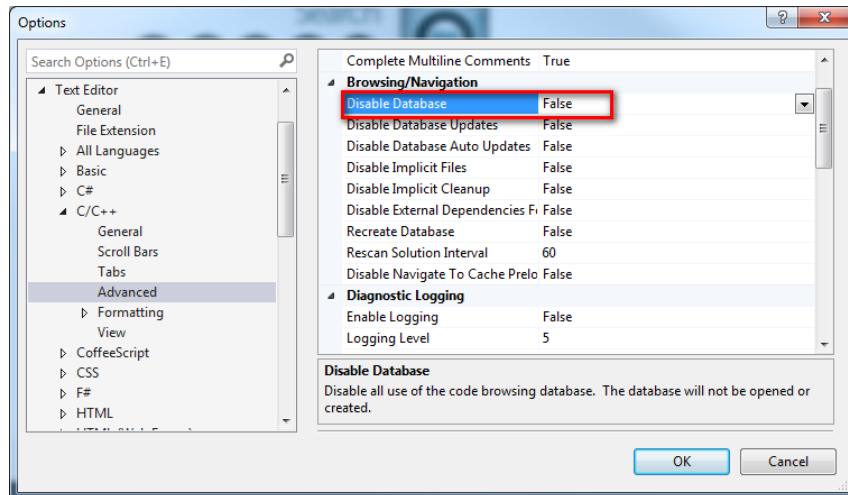


Figure 21 Disable database

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 22).

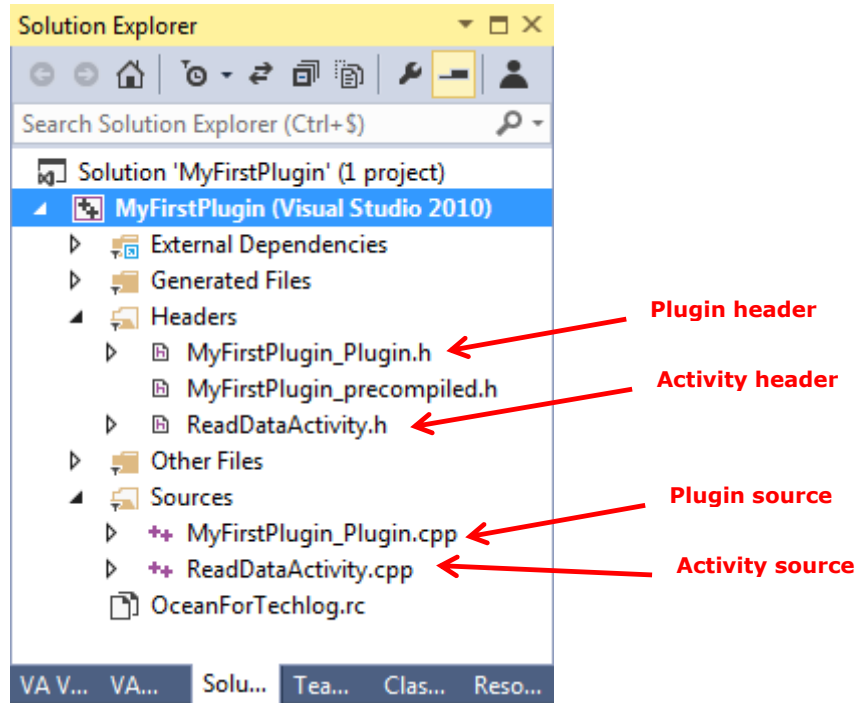


Figure 22 Example project header and source files in Solution Explorer

Plugin

The main plug-in class derives from `PluginIdentity` interface class.

`PluginIdentity` class is derived from `IPlugin` class (plug-in interface) that exposes the following 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"

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

```

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

```

These three methods must be implemented in the source file that first includes the plugin 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;

```

The `getInformation` method contains properties which provide information to the plugin. These include **Vendor name, Plug-in name, Plugin version, Description, Release date, Plug-in icon, Creator, Support email, Crash dump email, Plug-in license feature** and **Techlog license features dependency**. 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("03/12/2015");
    pluginInformation.setIcon(QIcon("ocean.png"));
}

```

```

pluginInformation.setCreator(PLUGIN_VENDOR_NAME);
pluginInformation.setSupportEmail("jsmith@slb.com");
pluginInformation.setCrashDumpEmail("jsmith@slb.com");
}

```

PLUGIN_VENDOR_NAME, PLUGIN_NAME and PLUGIN_VERSION Visual Studio properties can be changed through "Ocean for Techlog" tab in project properties.

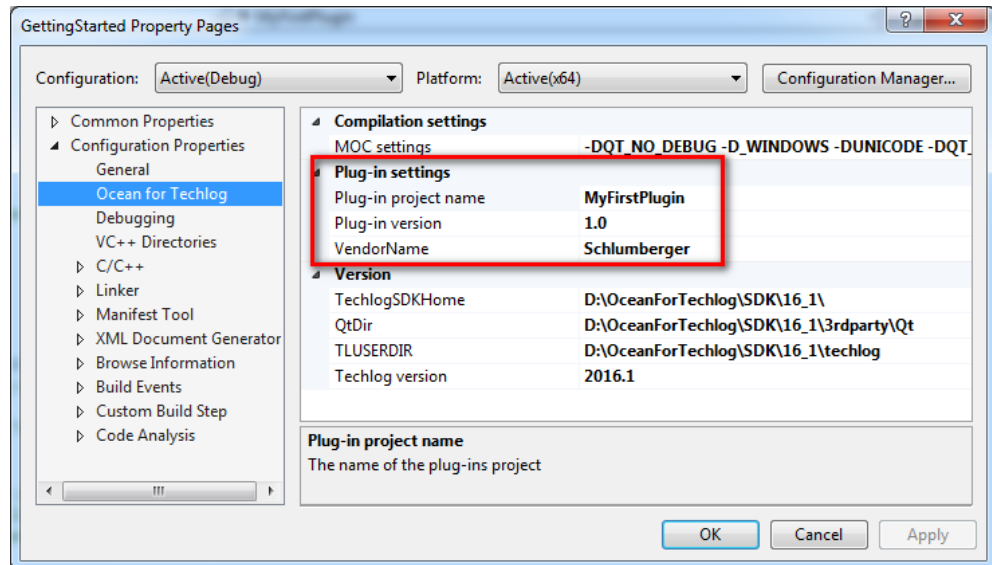


Figure 23 Plug-in settings

Note: vendorName, name and version property values of `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.

In `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("f1007f1e-1ce3-477e-a47f-d91f4e7e1b7b
"));

```

```

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 used to trigger activities.

The sequence to customize the TBar (Ribbon) can be summarized as follows 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.

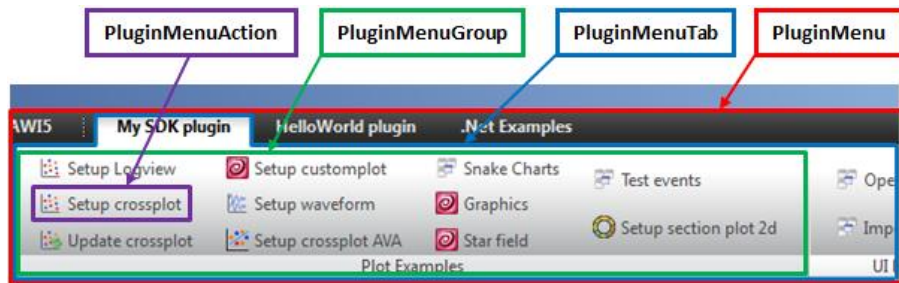


Figure 24 Plug-in menu classes

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

```
void MyFirstPlugin::getMenu( PluginMenu& menu ) const
{
    PluginMenuTab tab ("PluginArea");
    tab.setTitle("My first plug-in");

    PluginMenuGroup group ("PluginGroup");
    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.

The `dispose` method can be overridden in case 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 as follows:

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 exposed with `rowCount` public method and 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 exposes 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 `AbstractActivity` 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 25) or stops Techlog.

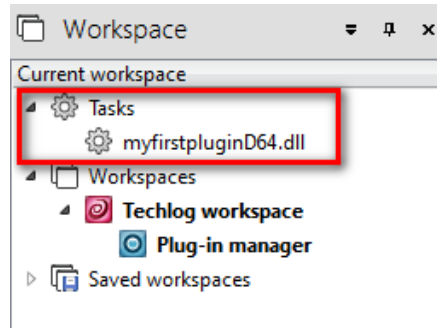


Figure 25 Techlog workspace manager

The following shows the complete activity class:

```
#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 running 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 separated 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 only one action menu **Read Data** (Figure 26).

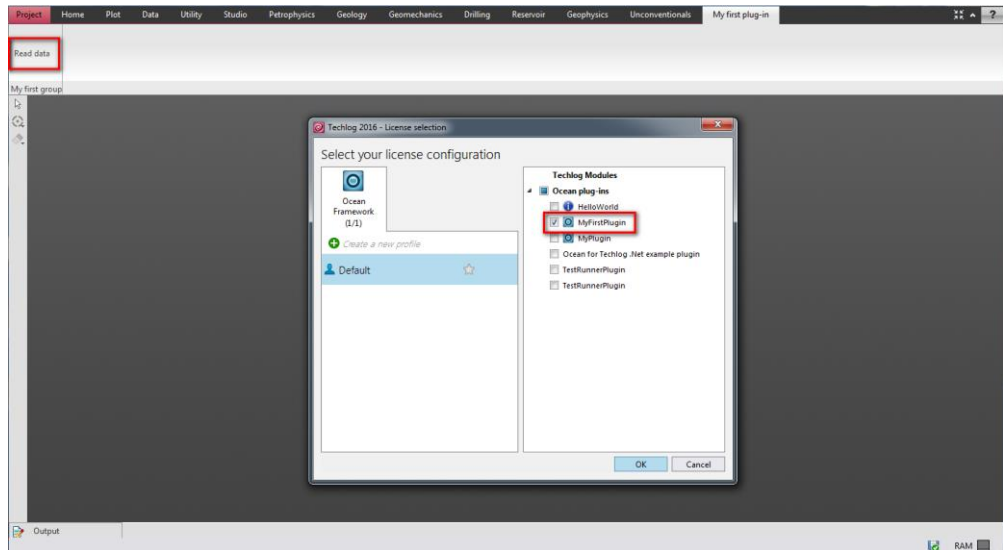


Figure 26 Enable MyFirstPlugin in the module manager

Import 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 27).

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

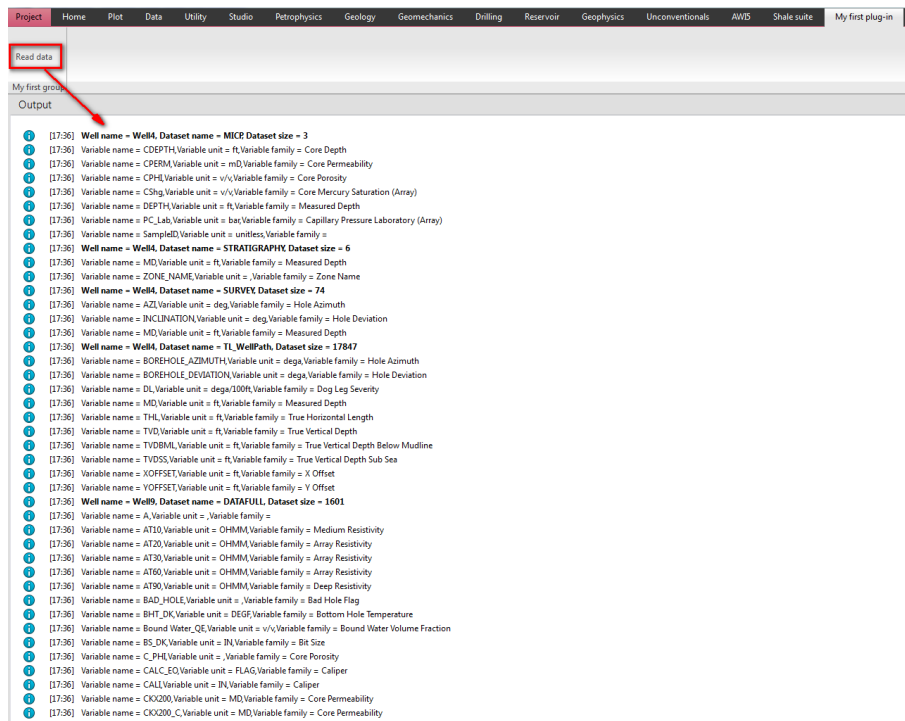


Figure 27 Read Data activity output messages

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 **MyFirstPlugin64D.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 node). The new plug-in for debugging appears in the module manager below Ocean plug-ins category. Disable the release version and enable the debug one.

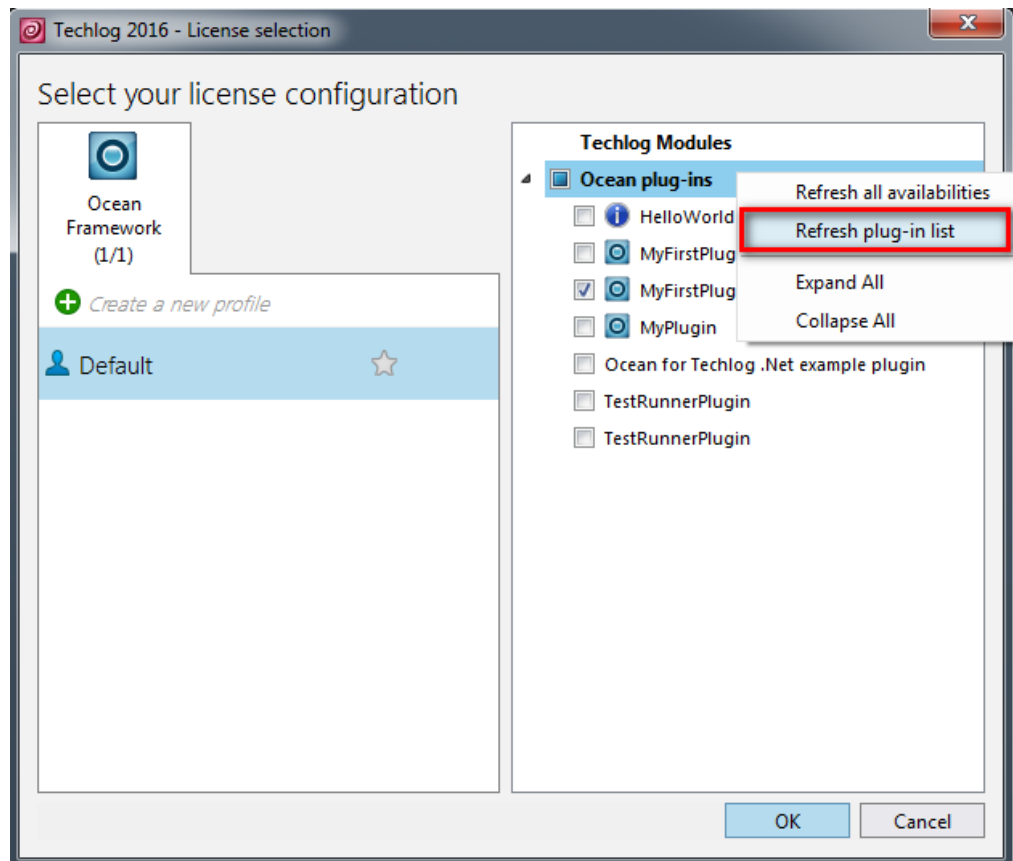


Figure 28 Refresh plug-in list

Press the **Ctrl+Alt** key of your keyboard 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 is for a plug-in built in 64 bit the **techlogpluginhost64D.exe**. Select **MyFirstPlugin** in the list and click **Yes** as shown in Figure 29.

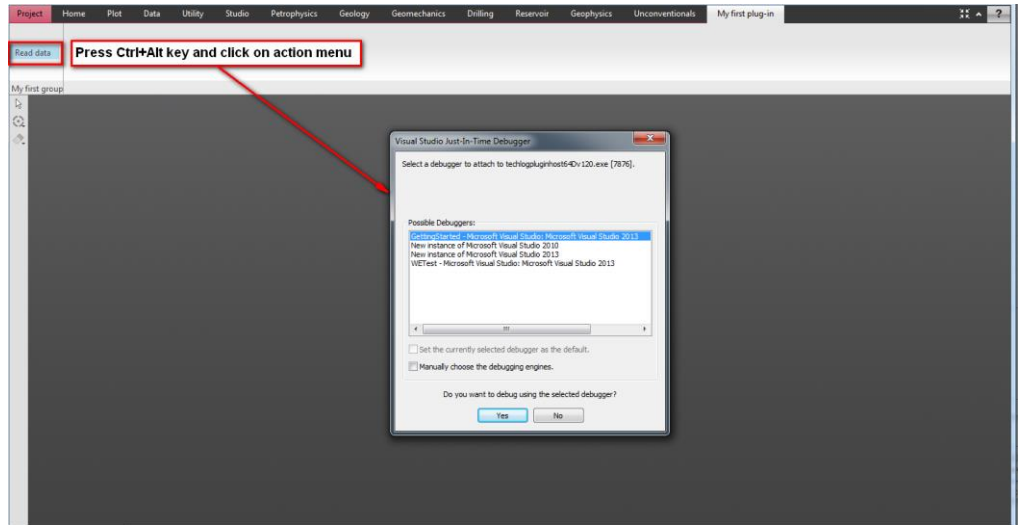


Figure 29 Debug the plug-in

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 plugin, you can simply not attached the .NET/Managed debugger at all.

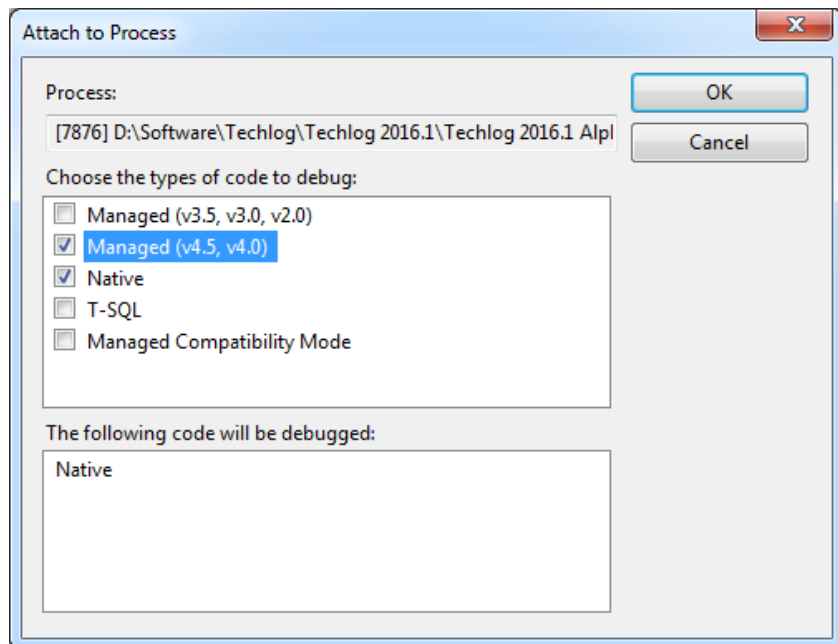


Figure 30 Visual Studio debugger engines

Techlog Viewer plug-in development (Schlumberger Internal only)

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

Techlog Viewer specific

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

```
class PluginInformation
{
public:
    ...
    void setTechlogViewerActivity(const QString
    &techlogViewerActivity)
};
```

Below 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 mono-well by design, developing a plug-in that will use several wells will result in having an assert 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 will be mandatory if any external deployment is planned.

To generate the signature file, please contact the Techlog Platform Product Champion
- 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 the tests can be integrated 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 clicking right on the solution in the Solution Explorer. Then select in the contextual menu **Add > New Project**. In the Project types area, under **Visual C++** project type, select **Ocean > Techlog 2016.1**. Then select the **Ocean Test Plug-in** template.

Note: A test project cannot be created into an empty Visual Studio solution. Test project wizard is looking at a main plug-in project in the solution.

Provide the name "TestMyFirstPlugin" for the project. Click **OK** to start the Wizard (see Figure 31).

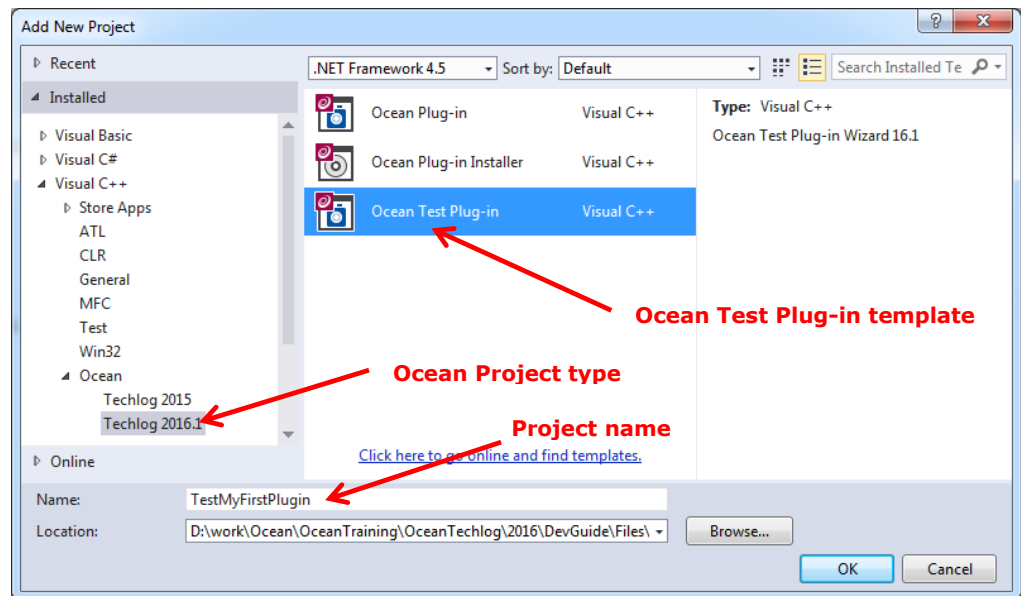


Figure 31 New project window

Test plug-in wizard shows up (see Figure 32).

The user is requested to set the following inputs:

- **Class:** Test plug-in class name
- **Main project:** select the Ocean plug-in presents in the solution that you want to test. Ocean plug-in will be a dependent library of the Test plug-in.

Click **Finish** in the dialog.

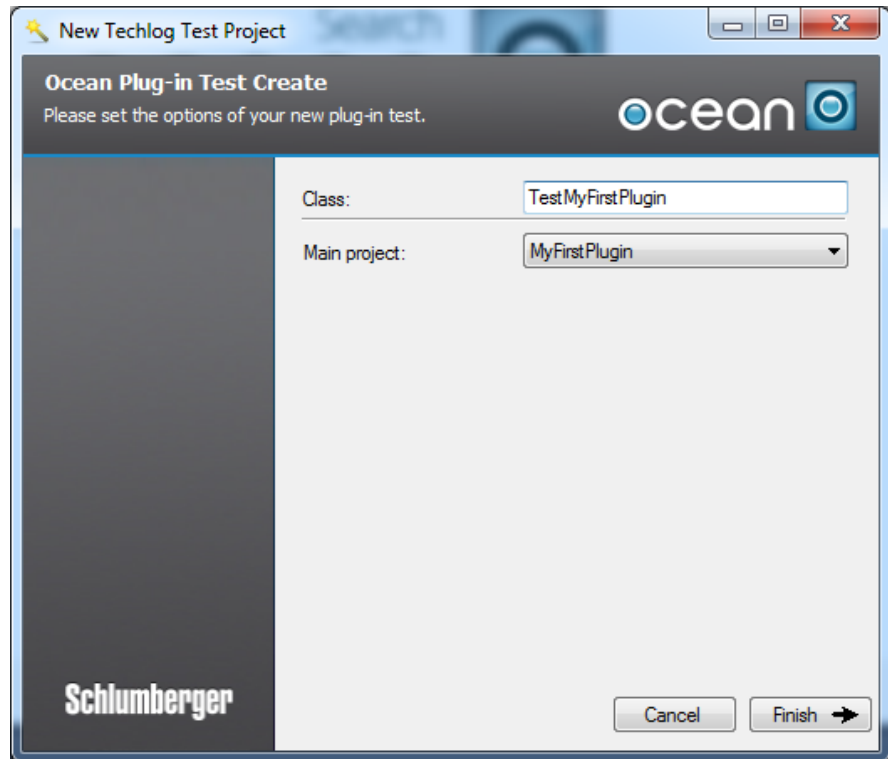


Figure 32 Test Plug-in wizard

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.

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 Test Activity and runner classes (see Figure 33.)

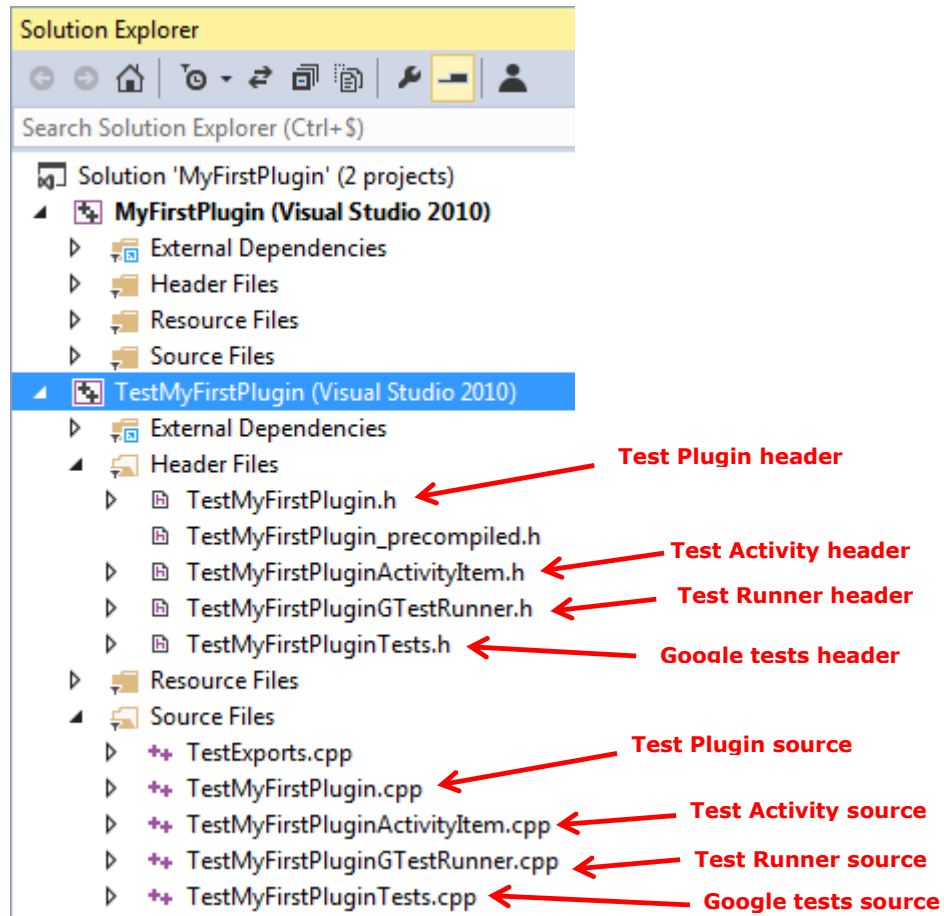


Figure 33 Test plug-in header and source files in Solution Explorer

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

Implement the tests

Google test API provides a number of options out there you could/should consider depending on your requirements. You can refer to the official Google test online documentation <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, you should define your test logic between a pair of braces. You can use a bunch of macros to indicate the success or failure of a test.

In the following 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 sub-routines that your tests need to invoke a lot.

In the following 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();
}

```

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 some calls to Ocean plug-in methods in Google tests.

if we have in **ReadDataActivity** of **MyFirstPlugin** a public method that allows us to remove from a Techlog **Variable** all the missing values and that we want to test this plug-in functionality calling it from a Google test of 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 that the plug-in developer needs to do is to export the **ReadDataActivity** class when **MyFirstPlugin** is built and import **ReadDataActivity** class when **TestMyFirstPlugin** is built. This can be done by adding to **MyFirstPlugin** project settings a conditional compilation tag in **C/C++ > Preprocessor > Preprocessor Definitions** (see figure 34).

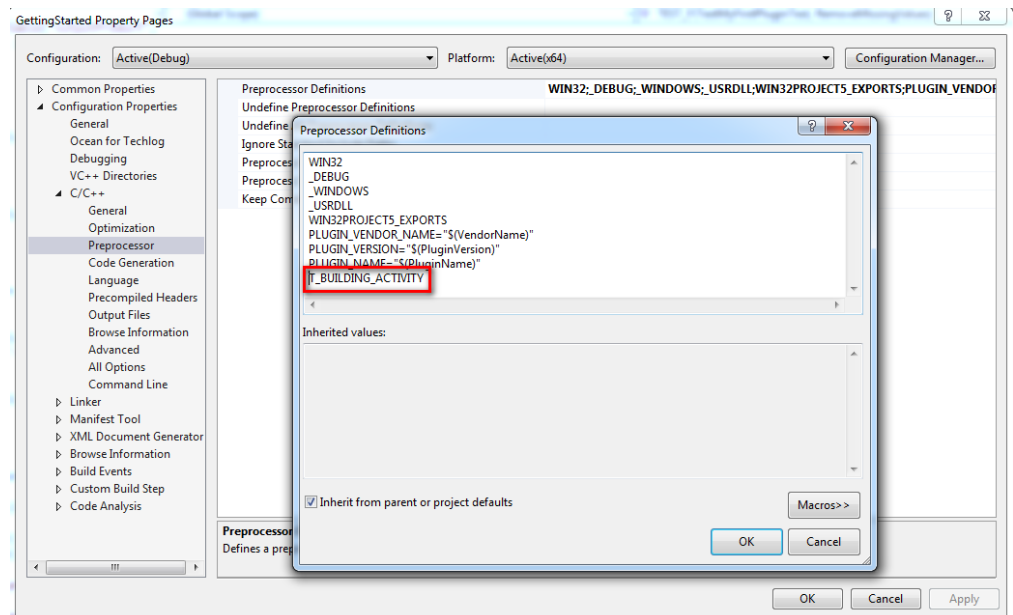


Figure 34 Add conditional compilation tag in Ocean plug-in settings

Then in **ReadDataActivity** header file, the code below must be added:

```
#ifndef T_BUILDING_ACTIVITY
#  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 and can now be called in a Google test as follows.

```
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();
}

```

In **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** can be listed by the Techlog module manager.

Open the module manager, refresh the list of plug-ins and enable **TestMyFirstPlugin**.

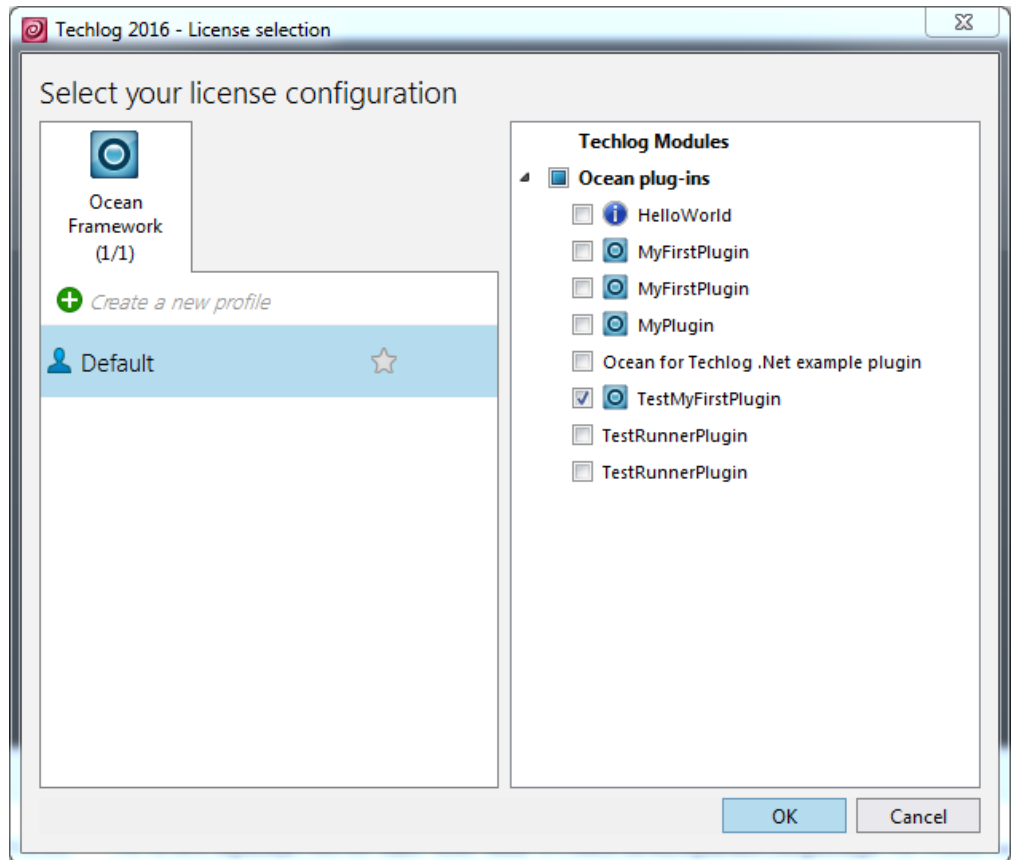


Figure 35 Enable test plug-in

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 to run, the user can open the result tests log file directly from the Techlog output console (see figure 36).

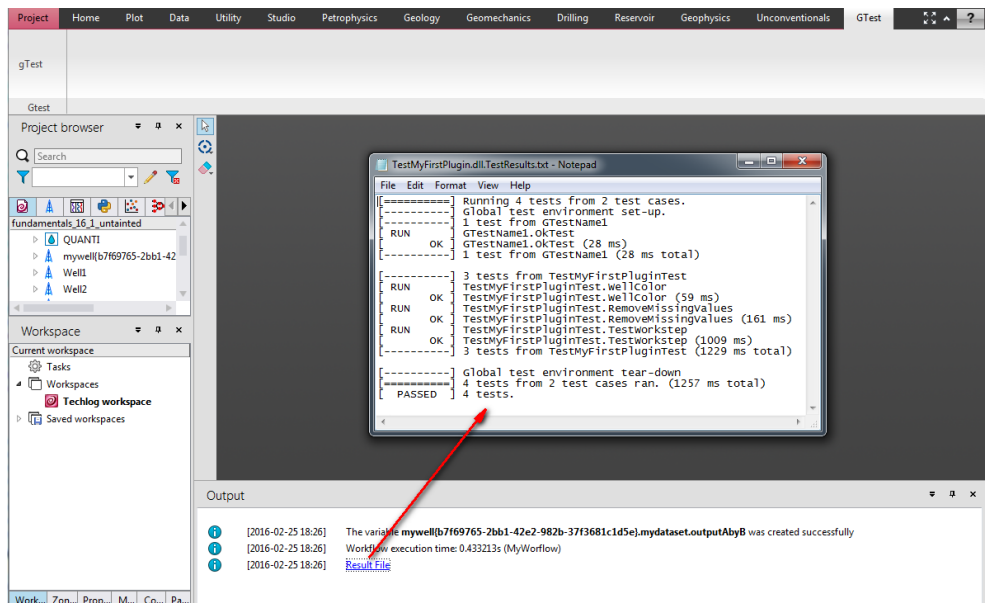


Figure 36 Gtest plug-in runs in Techlog context

You can also run the tests directly from Visual Studio through the Test Adapter.

1. In Visual Studio click on **TEST** menu and select **Windows > Test Explorer**. Text Explorer window opens in Visual Studio and when Test Plug-in is built all the tests are displayed in this window:

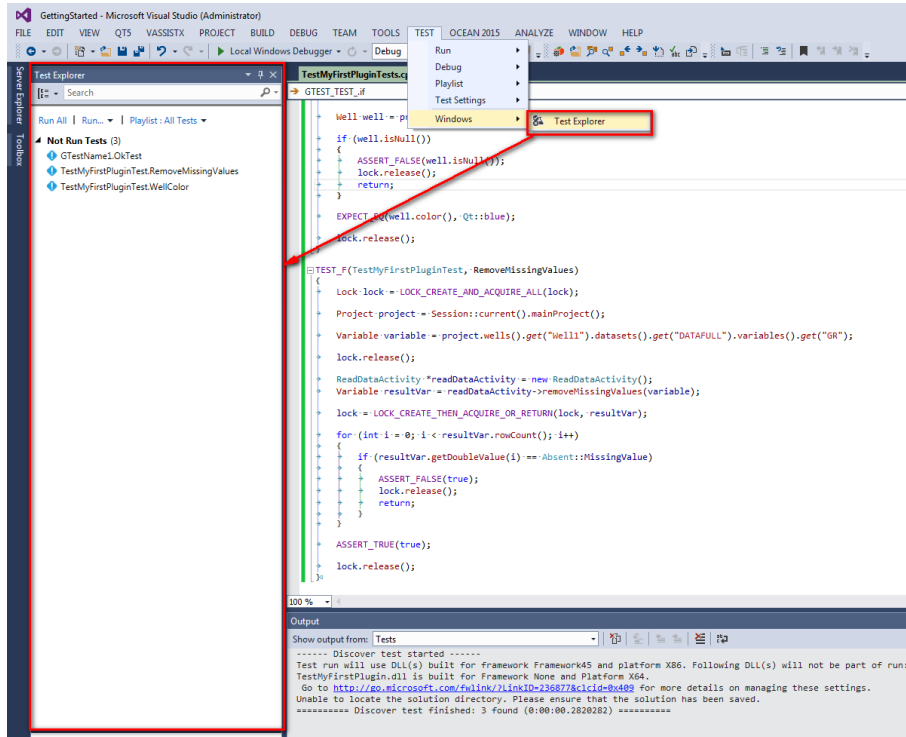


Figure 37 Open Test Explorer window

2. Open Techlog and enable **TestRunnerPlugin** in Techlog module manager. There are two TestRunnerPlugins available in the list: one to run the tests built in debug mode and the other one to run the tests built in release mode.

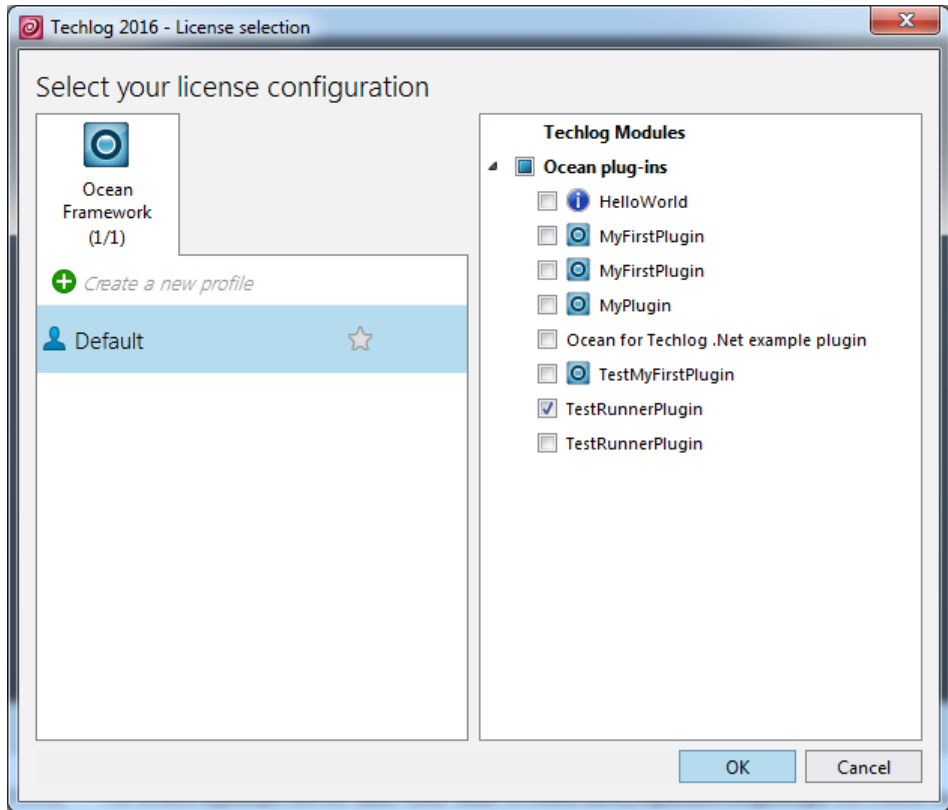


Figure 38 Enable TestRunnerPlugin

- Once Techlog is up and running with TestRunnerPlugin enabled, go back to Visual Studio and click the **Run all** link in the Test Explorer. The tests run in Techlog through the TestRunnerPlugin and results of the tests are displayed directly in Test Explorer window.

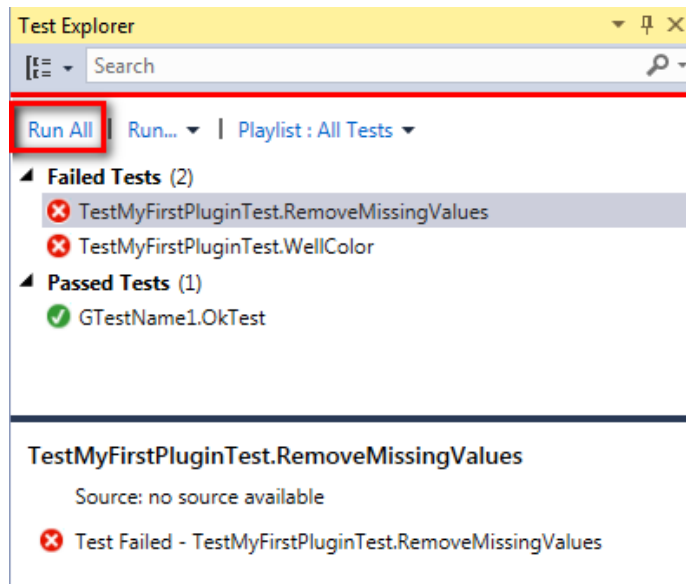


Figure 39 Run all tests

Note: Tests have to be run with "Default Processor Architecture" option set to **x64** in **TEST > Test Settings > Default Processor Architecture** menu. If x64 processor is not selected error message below is raised when tests are run.

Can't run the tests in Techlog. Please make sure that you use x64 version of VS Test Explorer. An exception occurred while invoking executor 'executor://techlogtestexecutor/': An attempt was made to load a program with an incorrect format.

Note: If several versions of the Ocean framework are installed make sure that the right **Test Adapter** version is selected in **Ocean > Techlog Test Adapter** menu in order to allow the **Test Explorer** to discover tests properly.

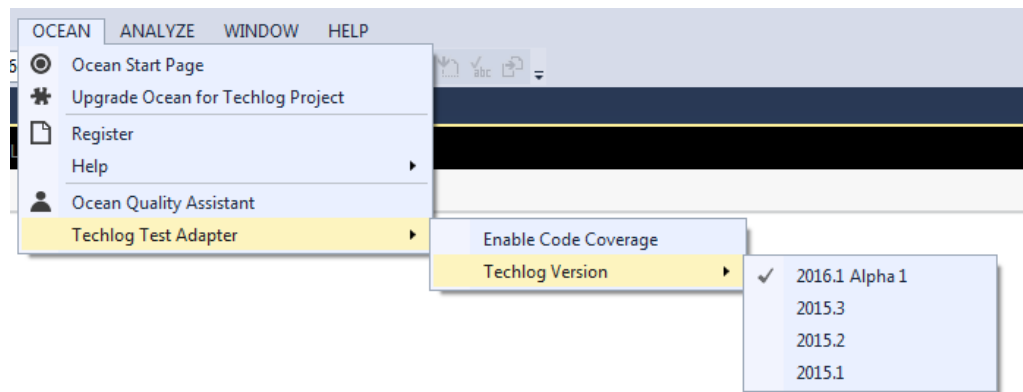


Figure 40 Techlog Test Adapter version

Create an installer for your plug-in

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. The prerequisite to use the Ocean Plug-in installer template is to have WIX 3.8 or an earlier version installed on his machine.

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 clicking right on the solution in the Solution Explorer. Then select in the contextual menu **Add > New Project**. In the Project types area, under **Visual C++** project type, select **Ocean > Techlog 2016.1**. Then select the **Ocean Plug-in Installer** template.

Note: An installer project cannot be created into an empty Visual Studio solution. Installer project wizard is looking at a main plug-in project in the solution.

Provide the name "MyFirstPluginInstaller" for the project. Click the **OK** button to start the Wizard. (See Figure 41.)

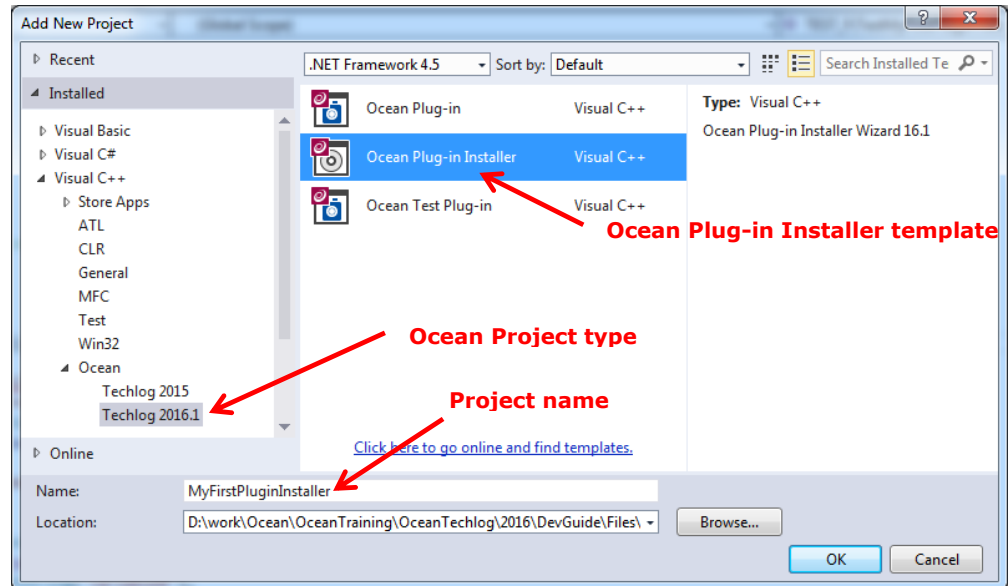


Figure 41 New project window

Plug-in installer wizard shows up (See Figure 41).

The user is requested to set the following inputs:

- **Title:** title of the Ocean plug-in that is showing up during plug-in installation
- **Company:** company name that owns the Ocean plug-in. This information is showing up during plug-in installation.
- **Description:** description of the Ocean plug-in that is showing up during plug-in installation
- **Projects:** select the Ocean plug-ins present in the solution that you want to package in the installer.

Click **Finish** in the dialog.

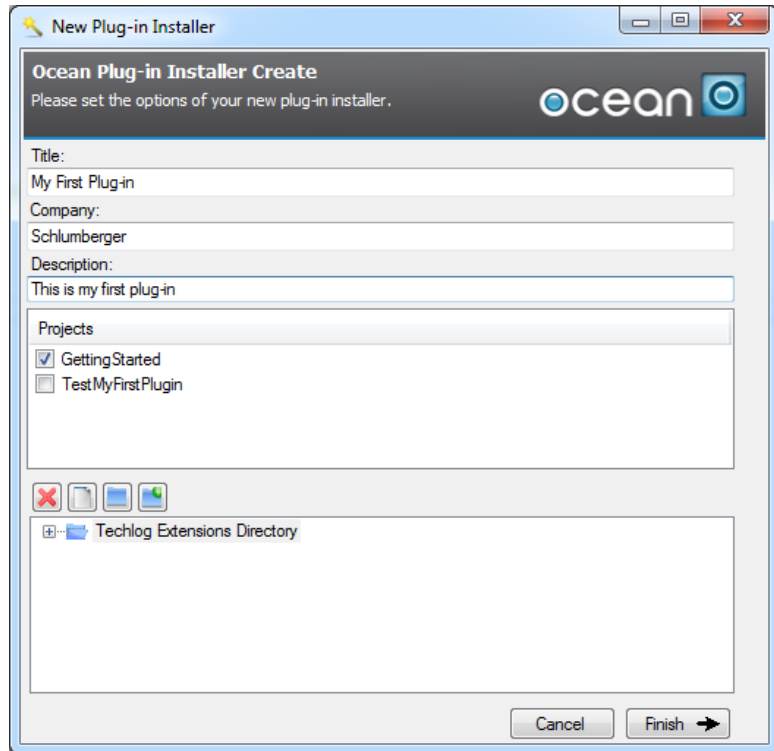


Figure 42 Plug-in installer wizard

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 and can be used to deploy the plug-in in Techlog for the plug-in users.

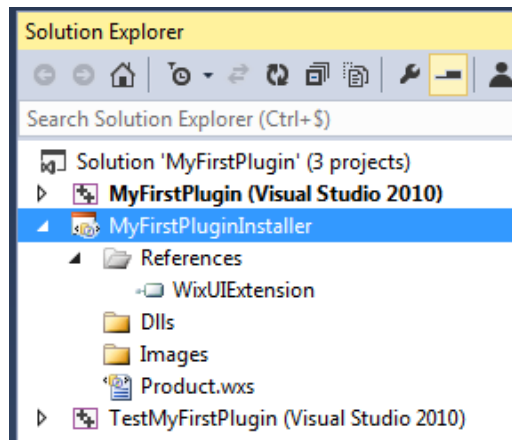


Figure 43 Plug-in installer project

Deploy folders and files with the plug-in dll

You may have to deploy additional files following a particular folders structure with your plug-in dll. WIX installer created through the Ocean plug-in installer template allows you to add those files editing the **Product.wxs** file.

Let's 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");

    Logview logview =
    Logview::create(workspace, logviewTemplate, wells);

    lock.release();

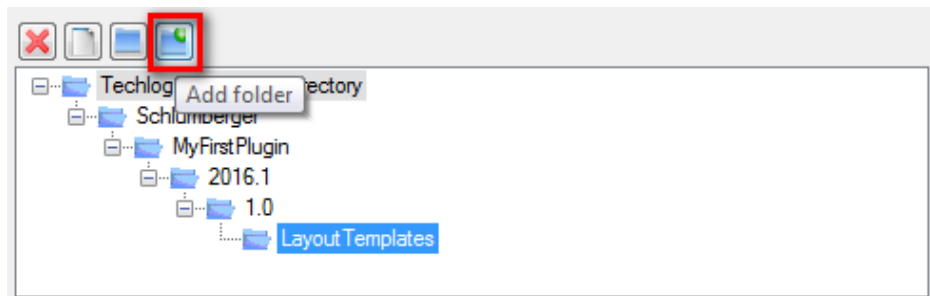
    stop();
}

```

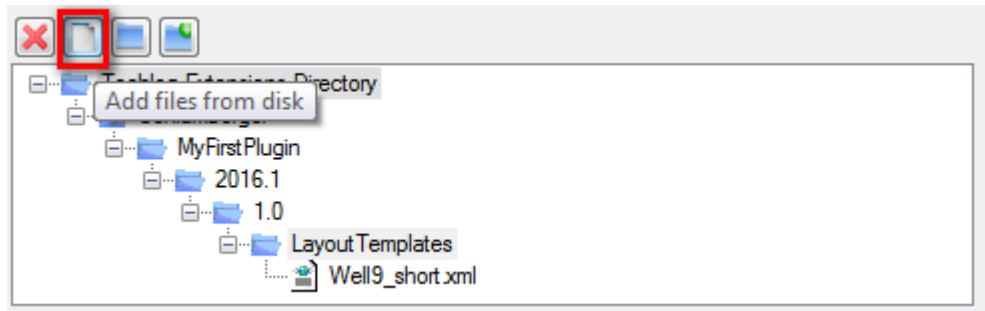
The layout template Well9_short.xml has to be deployed with the plug-in dll in a folder named LayoutTemplates.


In the lower section of the Ocean plug-in installer wizard dialog allows you to do this. You can browse a folder on the disk and add it or you can add a virtual one that will be created at the plug-in installation time.

1. Add LayoutTemplates folder to plug-in folder



2. Add Well9_short.xml file to 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 you can 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 some additional folders and files can be added modifying manually the **Product.wxs** file as follow:

1. The file must be added to the <Feature></Feature> block tags in the **Product.wxs** file.

```
<Fea-
tu-
re Id="ProductFeature" ConfigurableDirectory="EXTENSIONS" Descriptio
n="$(var.description)" Title="$(var.mainpluginname)" Level="1">
  <ComponentRef Id="Component" Primary="yes" />
  <ComponentRef Id="IniFile" Primary="yes"/>
  <ComponentRef Id="MyLayoutTemplate" Primary="yes"/>
</Feature>
```

2. Then declare inside the <Directory></Directory> block tags of the plug-in dll a <Directory></Directory> block tags with name attribute value equals to the name of the folder that you want to deploy with the plug-in dll (LayoutTemplates). The file, in our case Well9_short.xml, is added inside the new <Directory></Directory> block tags with component id previously declared.

```
<Directory Id="PluginVersion" Name="$(var.mainpluginversion)">
  <Compo-
nent Id="Component" Guid="80dd22d7-5e83-4967-88f3-9fec434a6b83">
  <Condition>TECHLOGPATH</Condition>
  <File Id="fil1649aa340c434607ae9771ceeebeb051" Source="..\MyFirs
tPlugin/x64/$(var.Configuration)/MyFirstPlugin.dll" />
  </Component>
  <Directory Id="LayoutTemplates" Name="LayoutTemplates">
  <Compo-
nent Id="MyLayoutTemplate" Guid="80dd22d7-5e83-4967-88f3-9fec434a6b8
4">
  <Condition>TECHLOGPATH</Condition>
  <File Id="fil1649aa340c434607ae9771ceeebeb052"
Source="..\MyFirstPlugin/x64/$(var.Configuration)/Well9_short.xml" /
>
  </Component>
  </Directory>
</Directory>
```

- The WIX installer searches for the file in the output directory of Visual Studio plug-in project.

Well9_short.xml file must be:

- copied to the Visual Studio plug-in project directory
- added to the Visual Studio project (**Add > Existing item** in contextual menu of the project)
- copied from the project directory to the output directory adding the following command line in Post-Build Event of project properties:

```
copy "Well9_short.xml" "$(OutDir)Well9_short.xml" /Y
```

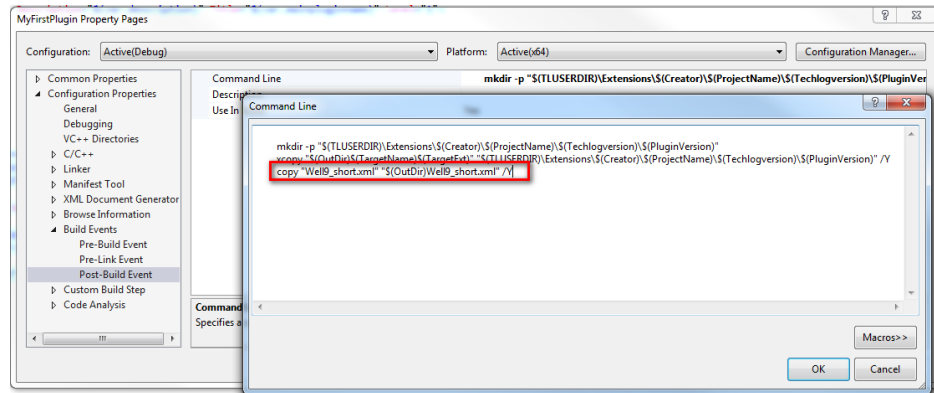


Figure 44 Post-Build Event in Ocean plug-in properties

User folder versus company folder deployment

A plug-in is unique 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 the "Writing the plug-in" for details on how to declare plug-in information.

If two plug-ins with the same key are deployed in the user folder, the Techlog module manager only shows up one. The behavior is exactly the same if one of the two plug-ins with same key is in the company folder and in this case the priority is given to user folder according to Techlog priority levels.

If you want to see the plug-in at company and user levels in the Techlog module manager, you have to rebuild the plug-in with a different **PluginInformation::version** and deploy the output dll under the corresponding version folder.

Upgrade existing Ocean plug-in to 2016.1

It isn't a prerequisite to uninstall Ocean framework 2015.1 before to install 2016.1 version. You can have several Ocean framework versions installed on your machine.

If you open an Ocean plug-in project created with Ocean template and wizard 2015.1, you can upgrade this project to 2016.1 clicking right on the project in the Solution Explorer. Then select in the contextual menu **Upgrade Ocean for Techlog project**.

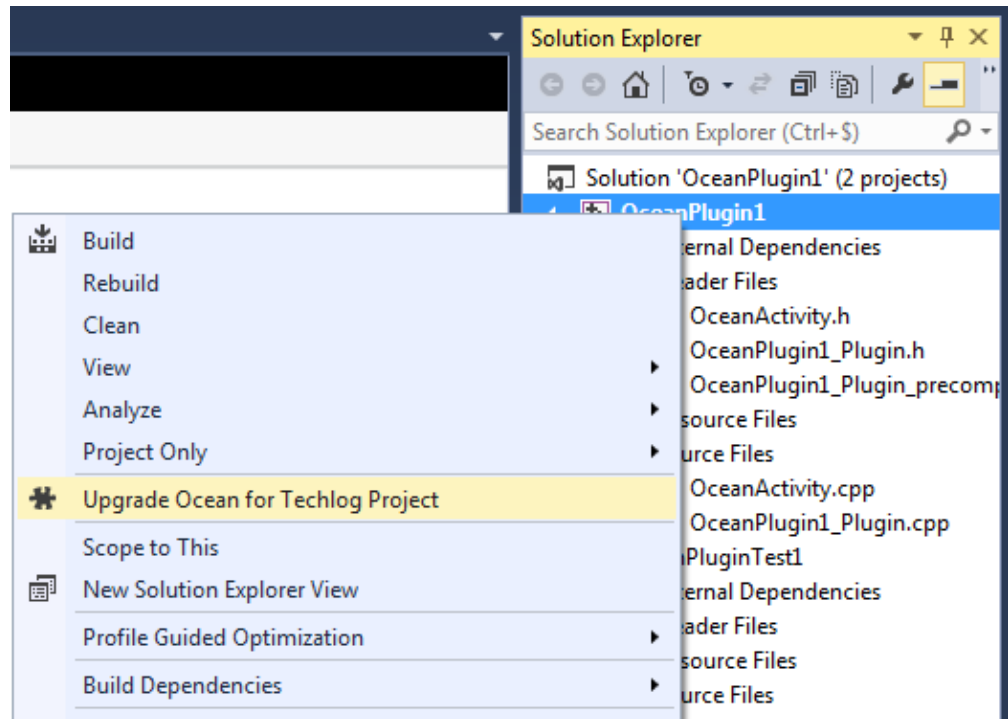


Figure 45 Upgrade Ocean for Techlog project

Upgrade window shows up asking you to confirm the project upgrade.

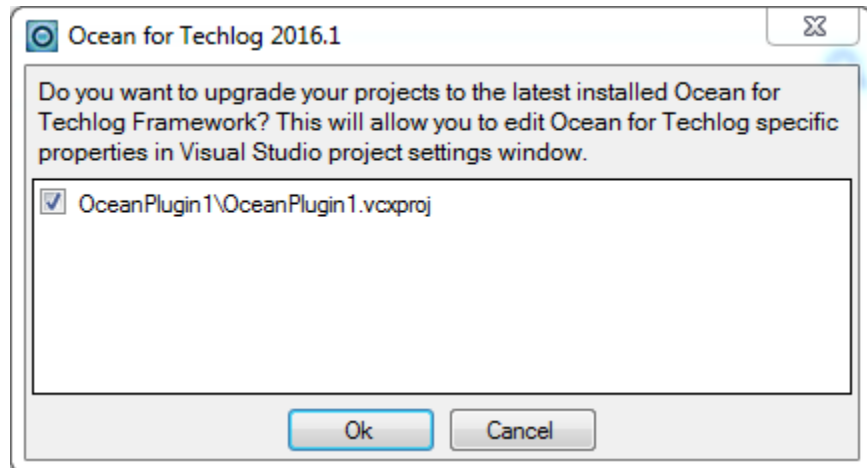


Figure 46 Upgrade window

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

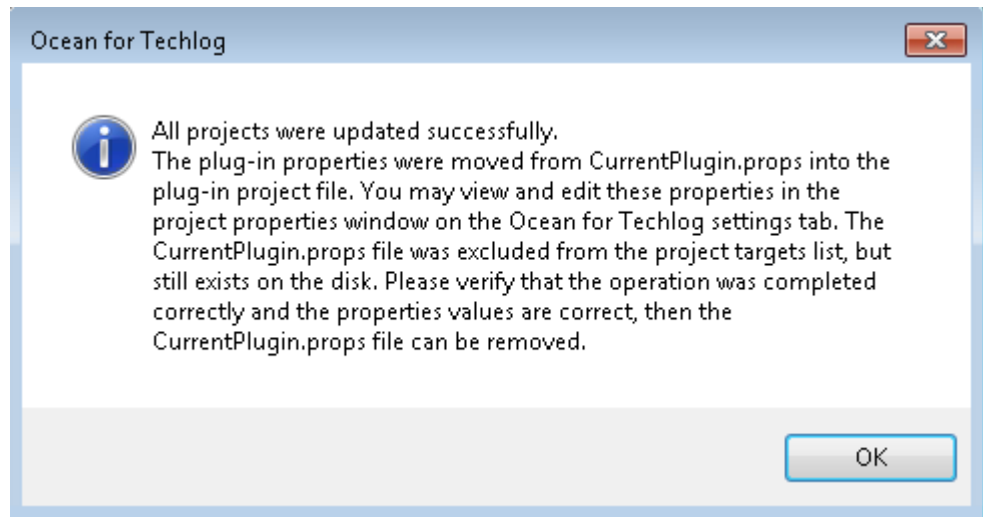


Figure 47 Upgrade information message

As it is mentioned in the dialog window, upgrading your project to Ocean framework 2016.1 allows you to take advantage of Ocean for Techlog properties added to project properties.

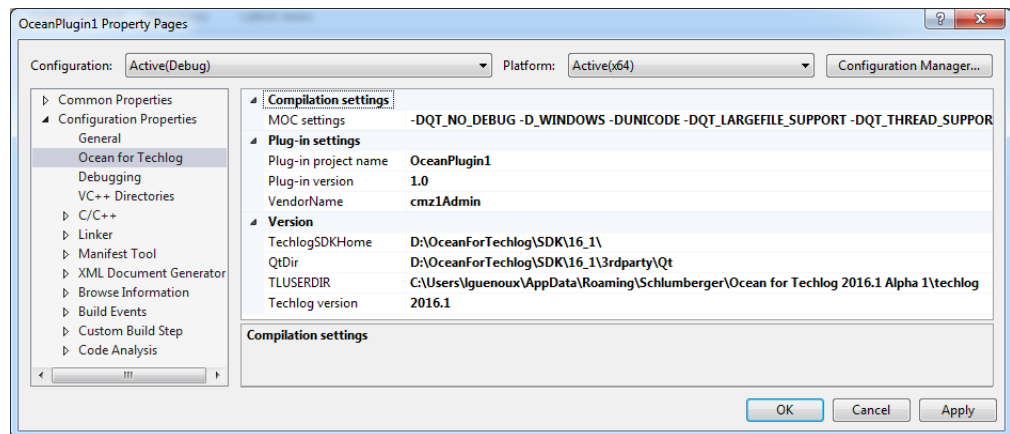


Figure 48 Ocean for Techlog project properties

In this tab you have access to three groups of properties:

- 1) **Compilation settings:** MocUserSettings is a set of definitions which are passed to MOC.exe at build-time.
- 2) **Plug-in settings:** allow you to mandatory plug-in information values as plug-in name, version and vendor name. Changing those values will make that the plug-in output dll is deployed with the corresponding plug-in structure folder at the post-build time.
- 3) **Version:** this group of properties allows you to handle the Ocean for Techlog binaries version (TechlogSDKHome) and Qt binaries version (QtDir) with which you want to build your plug-in. If you create an Ocean project plug-in from 2016.1 template or upgrade an Ocean plug-in project to 2016.1, the default values set to those properties are the TechlogSDKHome and QTDIR environment variable

values. But if you change those values through the Ocean for Techlog properties editor, the new values are only set at the project level and environment variable values remain unchanged. Through the "Techlog version" you can also control the Techlog version folder name of the plug-in structure folder. You can also change at the project setting level the path to the Techlog user folder (TLUSERDIR) where plug-in structure folders and the plug-in dll are generated at the post-build time.