# Contents

- Welcome to Ocean for Techlog ............................................................................. 2
- Ocean for Techlog Advantage ........................................................................... 2
- Ocean for Techlog Architecture ....................................................................... 3
- Access to the Techlog data model ..................................................................... 3
- Ocean for Techlog UI Infrastructure ................................................................ 4
- Ocean for Techlog plug-in identity and activities ........................................... 5
- Ocean framework license .................................................................................. 6
- Qt LGPL notice ................................................................................................... 6

## Install and setup the Ocean for Techlog development environment ................ 8

- Ocean for Techlog installation ......................................................................... 8
- Ocean for Techlog package content .................................................................. 11
- Ocean for Techlog environment variables ....................................................... 12

## Test the Ocean for Techlog development environment .................................... 13

- Writing your first plug-in .................................................................................. 18

### Writing the plug-in ......................................................................................... 18

#### Creating the Plug-in and Activity with Visual Studio ................................. 18

#### Inspecting the files ....................................................................................... 21

#### Plugin ............................................................................................................. 21

#### Activity .......................................................................................................... 24

#### Writing the algorithm code ......................................................................... 25

## Running the plug-in ......................................................................................... 28

## Debug the plug-in ............................................................................................. 29

## Create unit tests for your plug-in ..................................................................... 31

### Creating a Test plug-in with Visual Studio .................................................. 31

### Inspecting the files ......................................................................................... 32

### Implement the tests ........................................................................................ 33

### Run the tests .................................................................................................... 37

## Create an installer for your plug-in ................................................................. 40

### Deploy folders and files with the plug-in dll .............................................. 42

### User folder versus company folder deployment .......................................... 44
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 2015.1.

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  Techlog is purple, Ocean for Techlog is blue and plugins are dark blue.

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.

![Techlog module manager](image)

**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.
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).

**Figure 4**

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 5.3.2 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](image)

The installer checks:
- corresponding Techlog version is installed
- Visual Studio 2012 or 2013 is installed

2. Click **Next** in the dialog window. (See Figure 7.)
If you have already a Techlog user folder defined on your system (TLUSERDIR environment variable), the installer asks you if you want to overwrite it with a new user folder deployed with the Ocean for Techlog package. (See Figure 8.)

**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 9.)
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:

- 3rdparty
  - Qt
    - Qt 5.3.2 (Digia) libraries and header files
  - buildtools
- examples
  - DotNetExample
  - HelloWorld
  - MyPlugin
  - include
  - lib
- techlog
  - Extensions
  - Visual Studio Extensions
- win32

Figure 10 Ocean for Techlog package content

⚠️ The Ocean framework revision has to match EXACTLY the Techlog revision.

Plug-ins are built on Qt. The Ocean for Techlog Framework installer comes with the commercial 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.

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

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 TLCOMPANYDIR or TLUSERDIR environment variables to point the Company or 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 end-user 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 installer 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 you have a user folder already set on your machine, the Ocean for Techlog installer asks you if you want to replace your current user folder by the Extensions folder of the package (`%TechlogSDKHome%\techlog`).

If there is no user folder sets on your machine, the installer sets the Extensions folder of the package as user folder.

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

---

Schlumberger Private - Customer Use

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.

If you do not want to change the user folder you can also copy the entire content of the `D:|OceanForTechlog|SDK|techlog|Extensions` folder into your current user folder.

💡 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:
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 Studio and change plug-in version to 2.0 in:
   a. `getInformation` method of HelloWorldPlugin.cpp
      `pluginInformation.setVersion( QLatin1String( "2.0" ) );`
   b. path of dll output directory of HelloWorld.pro
      DESTDIR = ../../../techlog/Extensions/Schlumberger/HelloWorld/2015.1/2.0/

4. Run qmakepluginhelloworld.bat to update the visual studio project file and build the project in debug x64 mode.
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.

5. 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 14.
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.


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 TLUSERDIR is set to a previous SDK version, the built plugins will not be loaded (revision check).
- 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:

2. Create a new project by selecting File > New Project.
3. In the Project types area, under Visual C++ project type, select Ocean > Techlog 2015.
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](image-url)
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”, “Author”, “Support e-mail”, “Plugin URL”, “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 2015.

12. Select the Ocean Activity template.

13. Provide the name “ReadDataActivity” for the activity.

14. Click Add in the dialog (See Figure 19.)
Note: From this window you have the ability to create an Ocean Workstep Activity. This will add to the project an activity class that instanciates 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.

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 19  New activity window

Figure 20  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 21).

![Solution Explorer screenshot](image)

**Figure 21** 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:

```cpp
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)

```cpp
#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.

```cpp
#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, Creator, Support email, Crash dump email, Help file, Plug-in license feature and Techlog license features dependency. The contents of **getInformation** should look something like:

```cpp
void MyFirstPlugin::getInformation(PluginInformation& pluginInformation) const
{
```
pluginInformation.setVendorName("Schlumberger");
pluginInformation.setName("MyFirstPlugin");
pluginInformation.setVersion("1.0");
pluginInformation.setDescription("This is my first plug-in");
pluginInformation.setReleaseDate("03/12/2015");
pluginInformation.setIcon(QIcon("icon.png"));
pluginInformation.setCreator("Schlumberger");
pluginInformation.setSupportEmail("jsmith@slb.com");
pluginInformation.setCrashDumpEmail("jsmith@slb.com");
pluginInformation.setHelp("http://ocean.slb.com");
}

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*****/
}

Implement 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 22 Plug-in menu classes](image)

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

```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.

```cpp
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.

```cpp
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.

```cpp
#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 `logSeverity` 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 22) or stops Techlog.

![Techlog workspace manager](image)

**Figure 23** Techlog workspace manager

The following shows the complete activity class:

```cpp
#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, QStringLiteral("<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, QStringLiteral("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 24).

![Figure 24 Enable MyFirstPlugin in the module manager](image)

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 25).

**Note:** Opening a Techlog project with an Ocean framework license will taint the project.
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 and plug-in version to 2.0. 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 deployment folder. Then go back to Techlog and close and re-open the module manager. The new plug-in for debugging is displayed in the module manager below Ocean plug-ins category. Disable the release version and enable the debug one.

Press the Ctrl 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 26.
Figure 26  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 27  Visual Studio debugger engines
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 2015**. 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 28).

![New project window](image)

**Figure 28** New project window

Test plug-in wizard shows up (see Figure 29).

The user is requested to set the following inputs:
- **Class**: Test plug-in class name
- **Google test home folder**: path to the folder that contains Google test libraries (debug and release folders) and header files (include folder). Google test home directory must contain: include, debug and release sub-folders. Google test libraries can be downloaded from the [link](#).
- **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.

![Test Plug-in wizard](image)

**Figure 29** Test Plug-in 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 30.)
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/](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.

```cpp
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();

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.

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 31).
Then in `ReadDataActivity` header file, the code below must be added:

```c++
#ifdef 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:
    void 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.

```c++
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();
}

Run the tests

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

![Figure 32 Enable test plug-in](image)

**Figure 32** 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 33).

![Figure 33 Gtest plug-in runs in Techlog context](image)

**Figure 33** 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:

![Open Test Explorer window](image)

**Figure 34** 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.

![Enable TestRunnerPlugin](image)

**Figure 35** Enable TestRunnerPlugin
3. 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 36 Run all tests](image)

**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://techloggtestexecutor/': An attempt was made to load a program with an incorrect format.

---

**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 righ 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 2015**. 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 37.)

![Figure 37 New project window](image)

Plug-in installer wizard shows up (See Figure 38). 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.
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 39 Plug-in installer project](image)

**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.

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

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

```xml
<Feature Id="ProductFeature" ConfigurableDirectory="EXTENSIONS" Description="$(var.description)" Title="$(var.mainpluginname)" Level="1">
    <ComponentRef Id="Component" Primary="yes" />
    <ComponentRef Id="IniFile" Primary="yes"/>
    <ComponentRef Id="MyLayoutTemplate" Primary="yes"/>
</Feature>
```

Then declare inside the `<Directory></Directory>` block tags of the plug-in dll a `<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.

```xml
<Directory Id="PluginVersion" Name="$(var.mainpluginverson)"
    <Component Id="Component" Guid="80dd22d7-5e83-4967-88f3-9fec434a6b83">
        <Condition>TECHLOGPATH</Condition>
        <File Id="fil1649aa340c434607ae9771ceeebeb051" Source="..\MyFirstPlugin\x64\$(var.Configuration)\MyFirstPlugin.dll" />
    </Component>
    <Directory Id="LayoutTemplates" Name="LayoutTemplates">
        <Component Id="MyLayoutTemplate" Guid="80dd22d7-5e83-4967-88f3-9fec434a6b84">
            <Condition>TECHLOGPATH</Condition>
            <File Id="fil1649aa340c434607ae9771ceeebeb052" />
        </Component>
    </Directory>
```

Schlumberger Private - Customer Use
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 40](image.png)  

**Figure 40**  
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.