ocean

Creating advantage

OCEAN FOR STUDIO

Version 2020.1

DEVELOPER'S GUIDE

Schlumberger

Contents

1 Schlumberger product documentation

In this chapter

About Schlumberger

Schlumberger is the leading oilfield services provider, trusted to deliver superior results and improved E&P performance for oil and gas companies around the world. Through our well site operations and in our research and engineering facilities, we develop products, services, and solutions that optimize customer performance in a safe and environmentally sound manner.

Online documentation

Documentation is provided in these electronic formats:

- Compiled HTML online help (**.chm**) files
- Online Adobe® Acrobat® PDF files
- Sample code project with comments

Note: You can use Adobe® Reader to read the PDF file. Adobe Reader installation programs for common operating systems are available for a free download from the Adobe Web site at www.adobe.com.

Typestyle conventions

These conventions are observed throughout this guide:

- **Bold** text designates a file and folder names, dialog titles, names of buttons, icons, and menus, and terms that are objects of a user selection.
- *Italic* text emphasizes words, defined terms, and manual titles.
- Monospaced text (`Courier`) shows literal text as you would enter it or as it would appear in code or on screen.

Alert statements

The alerting statements are Notes, Cautions, and Warnings. These statements are formatted in these styles:

Note: Information that is incidental to the main text flow, or to an important point or tip provided in addition to the previous statement or instruction.

Caution: Advises of computer or data error that could occur should the user fail to take or avoid a specified action.

Warning: Requires immediate action by the user to prevent actual loss of data or where an action is irreversible, or when physical damage to the computer or devices is possible.

Third-party products

This product uses the following third-party products:

- **IQToolkit**: This product is distributed under the Microsoft® Public License (MsPL) (*http://iqtoolkit.codeplex.com/license*) and is available at *http://iqtoolkit.codeplex.com/*.

Contacting Schlumberger

Schlumberger has sales and support offices around the world. For information on contacting Schlumberger, please refer to the information below.

For Technical Support for SIS software:

- Schlumberger Support Portal: *http://support.software.slb.com*
- Customer Care Center e-mail: *customercarecenter@slb.com*
- Support Contact Details - Select your country to view support phone numbers. If your country is not listed, select the nearest location.

2 Overview

In this chapter

Overview

Ocean for Studio* is an extensible and integrated framework for Studio that shares the objectives of the Ocean for Petrel* development environment by providing data access, indexing tools, and plug-in convenience functions for Studio rather than Petrel.

With Ocean for Studio, developers can create custom plug-ins for Studio Manager that incorporate proprietary ideas or software while still participating fully in the Studio platform with respect to data access and interoperability with other aspects of Studio. If a plug-in solution does not meet a client's needs, standalone applications can also be developed with Ocean for Studio to take advantage of data access and indexing functionality.

Ocean for Studio Software Development Framework

The Ocean for Studio Software Development Framework contains code libraries, .dlls, and small sample programs to help you start developing applications for Studio. Ocean for Studio is composed of three APIs:

- The Studio for Petrel Database SDK* API
- The Studio Manager* API
- The Studio Find* API

This developer's guide provides information about the Studio for Petrel Database SDK API (a.k.a. The Studio SDK API, the shortened name will be used throughout this document) and the Studio Manager API.

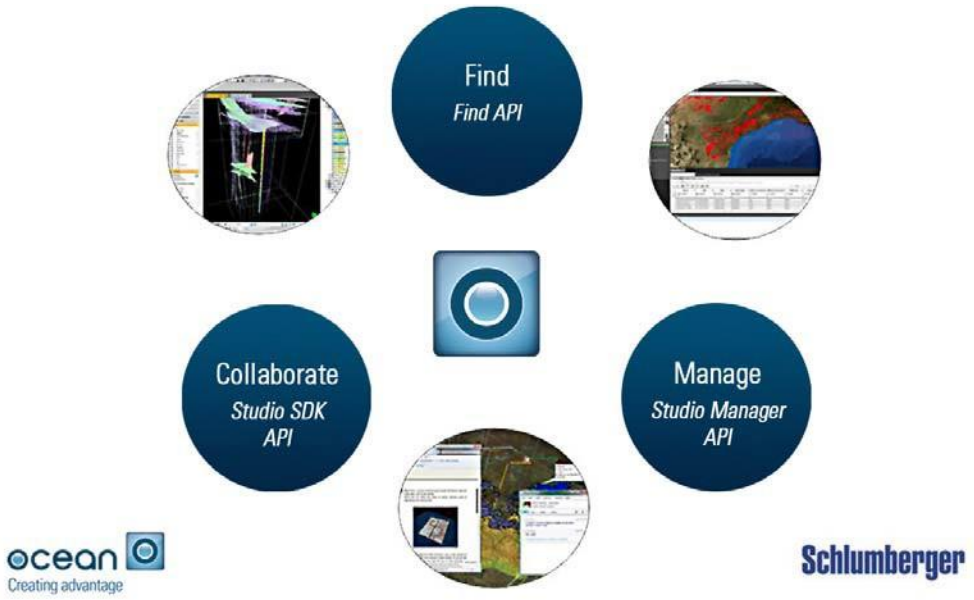


Figure 2-1 Ocean for Studio APIs and functionality

Note: This document does not include information about the Studio Find API, which has separate documentation and training materials.

Ocean for Studio package content

Ocean for Studio is delivered as part of the Studio installation package. For installation instructions, see the *Studio Installation and Configuration Guide*.

The Ocean for Studio package contains four folders.

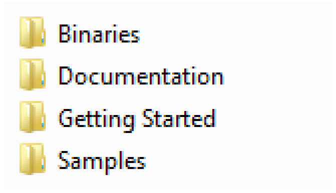
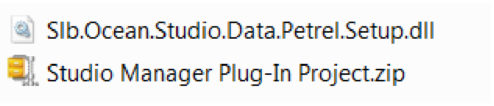

Binaries
Documentation
Getting Started
Samples

Figure 2-2 Contents of Ocean for Studio distribution package

Folder content

Binaries folder

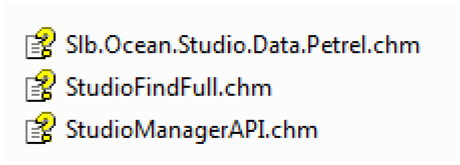
This folder contains one **.dll** file and one **.zip** file:


Slb.Ocean.Studio.Data.Petrel.Setup.dll
Studio Manager Plug-In Project.zip

- **Slb.Ocean.Studio.Data.Petrel.Setup.dll** is the assembly reference that can be used to create your own standalone applications, it enables applications to find Studio Runtime installation, load binaries and initialize Ocean services at run time. Refer to Chapter 3 of the Ocean for Studio Developer's Guide for more details.
- **Studio Manager Plug-in Project.zip** contains the Visual Studio templates that can be used to create a skeleton Visual Studio project of a Studio Manager plug-in.

Documentation folder

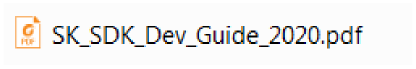
This folder contains three **.chm** files:


Slb.Ocean.Studio.Data.Petrel.chm
StudioFindFull.chm
StudioManagerAPI.chm

- **Slb.Ocean.Studio.Data.Petrel.chm** contains the logical description of the Ocean for Studio class model, especially pertaining to the Studio SDK API. This .chm file should be considered a primary reference for SDK developers.
- **StudioManagerAPI.chm** contains the description of the class model for the Studio Manager API.
- **StudioFindFull.chm** contains the description of the namespaces, classes, and services for the Studio Find API.

Getting Started folder

This folder contains .pdf versions of Ocean for Studio documents:

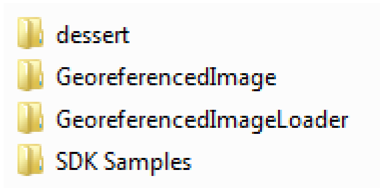

SK_SDK_Dev_Guide_2020.pdf

- The Ocean for Studio Developer's Guide is the most recent version of this document.

Note: The Ocean for Studio release notes, which contains information about changes to the product from release to release and documents any known issues you should be aware of, is now merged into the Studio Manager Release Notes.

Samples folder

This folder contains several subfolders:


dessert
GeoreferencedImage
GeoreferencedImageLoader
SDK Samples

Each subfolder contains example code that you can study and adapt to create your own plug-in or application. Highlights include:

- The **dessert** example indexes custom domain objects in Petrel so they can be viewed in Find Index results.
- The **GeoreferencedImage** example extends Find Indexing to external files (in this case, EXIF-tagged image files).
- The **GeoreferencedImageLoader** example extends Find Index loading into a Petrel application.
- The **SDK Samples** subfolder contains two examples:
  – **DomainObjectSample** contains sample code for a standalone application using the Studio SDK API that performs basic data loading into an empty Studio repository by using the sample data provided.
  – **SDK Plugin Sample** provides a similar example program for data loading, but it is in the form of a Studio Manager plug-in instead of a standalone application.

System requirements

Ocean for Studio has separate system requirements for development and distribution.

Development

When you develop applications and plug-ins, your development system must meet these minimum requirements:

- Microsoft Visual Studio® 2015
- Microsoft .NET Framework 4.7.2 on Windows 10 Pro or Enterprise (64-bit)
- A license for Ocean Development Framework

Distribution

When you distribute an application or plug-in, you must ensure that the target computer has the Microsoft Visual C++ Redistributable for Visual Studio 2010, 2012, 2013, 2019 installed. They are installed along with Runtime for Studio.

Licensing

Commercial implications

In general, the license model is the same as Ocean for Petrel. A fee is charged for the Ocean Development Framework which includes Ocean framework for Studio, Ocean framework for Petrel and Ocean framework for Techlog. This fee includes a run-time license for Studio Manager and a run-time license for third-party application accessing the Studio database, which allows development work to proceed without purchasing additional Studio commercial licenses for the applications.

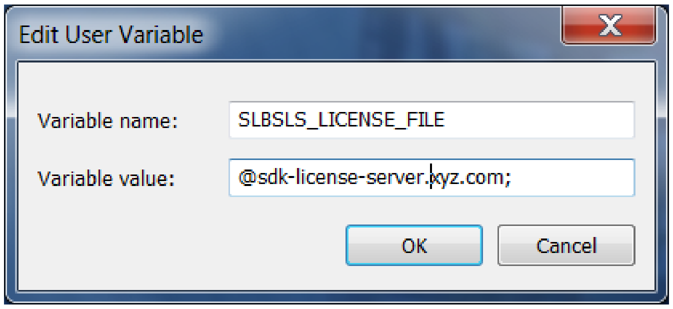
The restriction is that Studio repositories created with an Ocean development license cannot be used in commercial deployments, and they are internally stamped as *Development*. Essentially, these restrictions require the commercial license features for Studio to deploy a commercial solution written using Ocean for Studio.

For additional details about the commercial terms of Ocean for Studio, contact the SIS Ocean team or your SIS account manager.

Software requirements

Ensure that your system meets these software requirements:

- Run the latest commercial version of the Schlumberger Licensing Manager.
- Define the **SLBSLS_LICENSE_FILE** environment variable so that it points to the Ocean for Studio license server as shown in the following figure.



Variable name: SLBSLS_LICENSE_FILE
Variable value: @sdk-license-server.xyz.com;

Figure 2-3 Edit or define the SLBSLS_LICENSE_FILE environment variable

To access your system environment variables:

1. Open the **Microsoft Control Panel**, click **System**, and then click **Advanced system settings**.
2. On the **Advanced** tab, click **Environment Variables**. In the **User variable** pane, locate **SLBSLS_LICENSE_FILE**. If it isn't listed, click **New** to define it.

License types

There are two types of licenses for applications developed using Ocean for Studio:

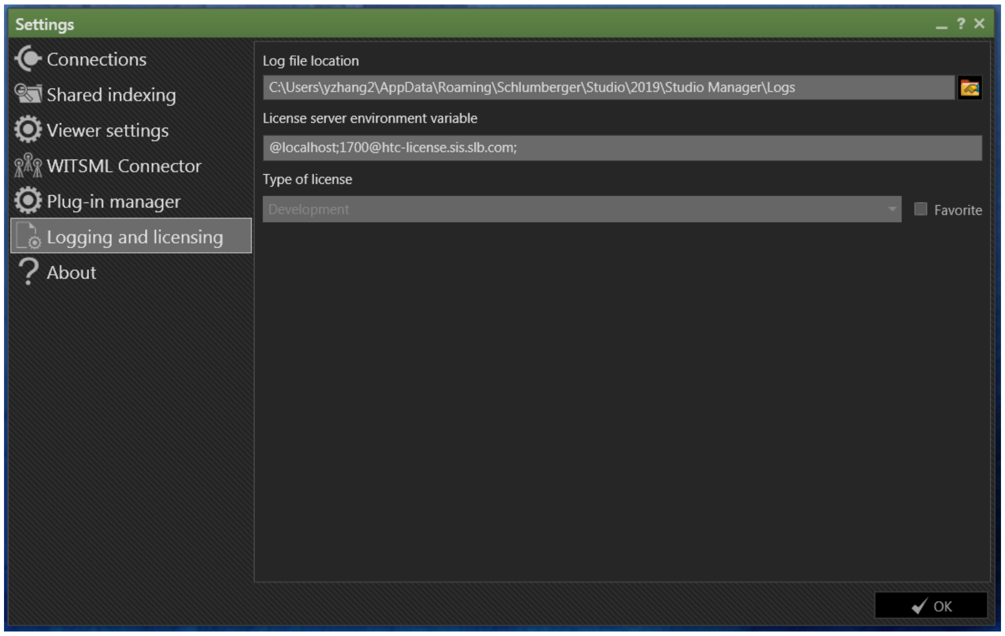
- *Commercial* (**Studio Third-Party Access** license with feature name **`Studio_Runtime`**, feature name **`Studio_Knowledge_Runtime`** has been Obsolete since 2015.1)

- *Developer* (**Ocean Development Framework** license with feature name `Petrel_FAPI` and `Studio_API`, feature name `Studio_Knowledge_SDK` has been Obsolete since 2015.1)

Additionally, you can create a repository at either of the two access levels:

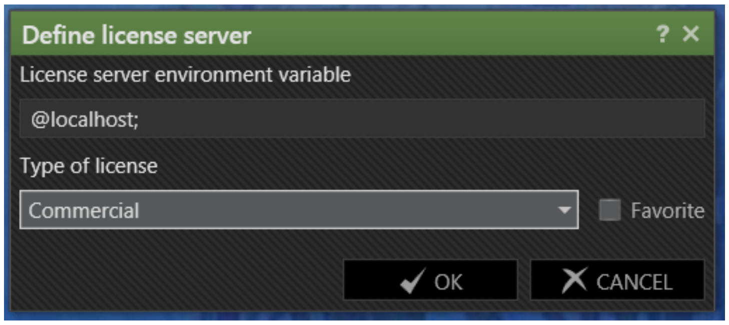
- `Commercial` access level, when Studio Manager is launched with Commercial selected as type of license
- `Development` access level, when Studio Manager is launched with Development selected as type of license

To launch Studio Manager with a specific license type, you may first need to uncheck **Favorite** option from **Logging and licensing** page under **Settings** (Figure 2-4), close Studio Manager; then re-launch Studio Manager, **Define license server** dialog (Figure 2-5) is now opened to select type of license.



Log file location
C:\Users\yzhang2\AppData\Roaming\Schlumberger\Studio\2019\Studio Manager\Logs
License server environment variable
@localhost;1700@htc-license.sis.slb.com;
Type of license
Development
Favorite

Figure 2-4 Uncheck Favorite for Type of license – Commercial or Development



License server environment variable
@localhost;
Type of license
Commercial
Favorite

Figure 2-5 Define type of license – Commercial or Development

For a standalone application developed using the Studio SDK API, a license is checked out when a repository is opened. As illustrated in the matrix below, the license type, combined with the repository access level, determines your ability to access repository data.

Repository Access Level	License Type	
	Commercial	Developer
Commercial	Read/write	Read-only
Development	Access denied	Read/write

Here are example scenarios of different combinations of license types and repository access levels.

Scenario 1: Developer license

You have an Ocean development framework license that allows you to launch a standalone application with Runtime for Studio. Your application opens a repository that has a Commercial access level. The result is that your application can open the repository as read-only. If it tries to save changes, you will be notified that the repository is read-only.

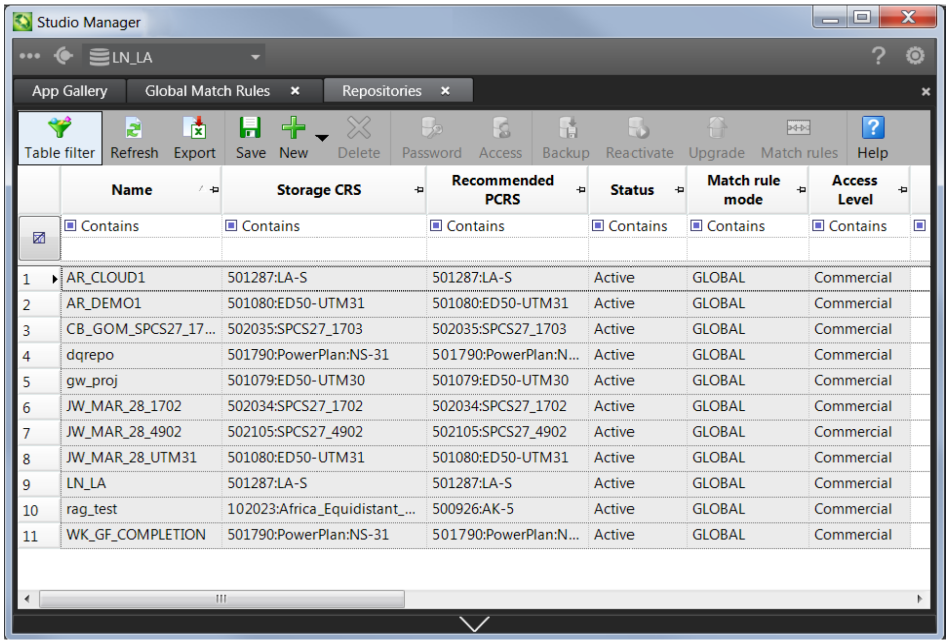
Scenario 2: Commercial license

You have a Studio third-party access license that allows you to launch a standalone application with Runtime for Studio. Your application opens a repository that has a Development access level. As a result, the request to open the repository is denied.

Scenario 3: Both types of licenses

You have both types of licenses and you open a repository that has a Commercial access level.

- Ocean for Studio first tries to open the repository by checking out a Studio third-party access license. If this is successful, the result is that you have read/write permissions to the repository.
- Otherwise Ocean for Studio then tries to open the repository by checking out a Ocean development framework license. The result is that you have read-only access to the repository.



	Name	Storage CRS	Recommended PCRS	Status	Match rule mode	Access Level
	Contains	Contains	Contains	Contains	Contains	Contains
1	AR_CLOUD1	501287:LA-S	501287:LA-S	Active	GLOBAL	Commercial
2	AR_DEMO1	501080:ED50-UTM31	501080:ED50-UTM31	Active	GLOBAL	Commercial
3	CB_GOM_SPCS27_17...	502035:SPCS27_1703	502035:SPCS27_1703	Active	GLOBAL	Commercial
4	dqrepo	501790:PowerPlan:NS-31	501790:PowerPlan:N...	Active	GLOBAL	Commercial
5	gw_proj	501079:ED50-UTM30	501079:ED50-UTM30	Active	GLOBAL	Commercial
6	JW_MAR_28_1702	502034:SPCS27_1702	502034:SPCS27_1702	Active	GLOBAL	Commercial
7	JW_MAR_28_4902	502105:SPCS27_4902	502105:SPCS27_4902	Active	GLOBAL	Commercial
8	JW_MAR_28_UTM31	501080:ED50-UTM31	501080:ED50-UTM31	Active	GLOBAL	Commercial
9	LN_LA	501287:LA-S	501287:LA-S	Active	GLOBAL	Commercial
10	rag_test	102023:Africa_Equidistant_...	500926:AK-5	Active	GLOBAL	Commercial
11	WK_GF_COMPLETION	501790:PowerPlan:NS-31	501790:PowerPlan:N...	Active	GLOBAL	Commercial

Figure 2-6 Studio Manager access levels - Commercial

Ocean for Studio* and Ocean for Petrel*

The Studio SDK API portion of Ocean for Studio is conceptually based on the Ocean for Petrel API.

About Ocean for Petrel

Ocean for Petrel is an application development framework that allows you to develop plug-ins that are tightly integrated with the Petrel* product.

Ocean for Petrel enables plug-ins to extend the Petrel functionality with new custom workflows, while still benefiting from existing functionality in Petrel.

The coexistence of plug-ins with Petrel on top of the same framework allows plug-in developers to better integrate workflows that cross domain boundaries, from seismic to simulation.

Ocean for Petrel offers a public API that is open to clients and third parties.

Differences

Although the Studio SDK API is based on the Ocean for Petrel API, there are some important differences:

- Ocean for Petrel API: Designed to extend Petrel by applications that interact with the users through the Petrel interface.
- Studio SDK API: Flows data in and out of the Studio Database without interacting with the Petrel interface or the Studio Manager interface.

This table summarizes the differences between the Studio SDK API and the Ocean for Petrel API.

Feature	Ocean for Petrel	Ocean for Studio (Studio SDK API)
Repository	One per-process: **`PetrelProject.PrimaryProject`**	Many per-process: **`DatabaseSystem.RepositoryService.OpenRepository`**()
Transaction	**`ITransaction`**.{Commit, Abandon}	**`RepositoryService`**.{SaveRepository, CloseRepository, DiscardChanges}
Locking	**`ITransaction.Lock`**	NA
Events	Supported	NA
Delete	Supported	**`DomainObjectBase.Delete`**()
Auto Recomputation	Supported	NA

Query	Hierarchical + LINQ to object	Hierarchical + LINQ to object Custom LINQ to SQL (limited operator/object)
Factory method	Cannot specify **`Droid`**	Can specify **`UniqueID`**
Validation	At the method/property level	At the method/property level. Also, supported at the repository level during save and commit, and supported at the object level.

Ocean for Studio documentation

The Ocean for Studio documentation is available in these formats:

- Online help
- PDF files
- Sample code

Online help

The Ocean for Studio domain objects is documented in these files:

- **Slb.Ocean.Studio.Data.Petrel.chm**
- **StudioFindFull.chm**
- **StudioManagerAPI.chm**

These files contain information about all the Ocean for Studio classes, objects, and functions.

PDF files

These manuals, provided in PDF format, contain valuable Ocean for Studio information:

- *Ocean for Studio Developer's Guide* provides overview and getting started information related to the SDK distribution. The developer's guide is included with the Ocean for Studio distribution.
- *Ocean for Petrel Developer's Guide* provides conceptual and development information related to the Ocean for Petrel API, therefore helps to understand the Studio SDK API. This 11-volume set is not included with the Ocean for Studio distribution.

Ocean information is available on the Developer section of the Ocean store at *https://www.ocean.slb.com/en/developer*.

Sample code

Ocean for Studio includes sample code that you can use to test your configuration. The sample code includes comments that document how you should use the sample code. Additional sample code is available on the Developer section of the Ocean store.

Ocean for Petrel developer resources

Because there are many similarities between the Studio SDK API and the Ocean for Petrel API, the Ocean for Petrel API information can be a valuable resource. You can view this information on the Developer area of the Ocean Store:

- Bulletin Board
- Code Samples
- Developer's Guide
- Best Practices & Papers
- Open Inventor Resources
- GeoToolkit Resources

You can view the Developer area of the Ocean Store at this URL:

https://www.ocean.slb.com/en/developer

Note: The *Ocean for Petrel Developer's Guide* is the documentation that you may find most helpful.

These sections of the Ocean for Petrel Developer's Guide apply to using the Studio SDK API. Although the content is written with a different purpose in mind, the information can be very valuable.

- Ocean Core and Services
- Basic Concepts of Ocean Plug-ins
- Shapes Objects
- Seismic Data and Seismic Interpretation Data
- Wells, Well Logs, Markers and Geologic Data

Ocean for Studio API updates

For a complete list of API changes, see the Studio Manager Release Notes.

Note: See the **Slb.Ocean.Studio.Data.Petrel.chm** file for more information about the Studio SDK API

3 Develop applications with the Studio SDK API

In this chapter

Test the Studio SDK API environment

In this task, you will run a sample program to test your environment. You can find the files for this sample program in the folder named **/Samples/SDK Samples/DomainObjectSample**.

The executable that invokes the sample program is named **Slb.Ocean.Studio.Data.Petrel.Sample.exe**.

Before you compile the sample program or create your own programs, ensure that your Petrel and Studio environments are configured correctly.

Note: Ensure that licenses are set up and that you have a valid license server, and Runtime for Studio is installed.

About the configuration file

Use the **Slb.Ocean.Studio.Data.Petrel.Sample.exe.config** file to define repository connection information to a database server and to define connection information for Studio Manager.

You must change two sections in the file so that you can connect to your Studio database.

- **ConnectionSpecificationConfiguration**: This section provides the connection information for the Oracle or SQL server database. You must supply values for these parameters:
  - **Name**: The name of the data source. This can be any name you choose. This name is also used in the appSettings section to point to the Oracle or SQL Server information specified here.
  - **Provider**: Oracle or SQL server.
  - **Server**: Server name for the Studio database.
  - **Port**: Port number for the Studio database.
  - **Service**: Service name for the Studio database.
- **appSettings**: This section provides the connection information for Studio Manager. You must supply values for these parameters:

- **DataSource**: The data source **Name** you defined in the `ConnectionSpecificationConfiguration` section. This information can either be specified in that section or in Studio Manager. The data source **Name** is used to complete the database connection with the rest of the information supplied in the `appSettings` section.
- **UseIntegratedSecurity**: Set to True to enable Windows authentication.
- **UserID**: The user ID used to access Studio database.
- **Password**: The password used to access Studio database.
- **RepositoryName**: The name of the repository to use in the Studio database.
- **UserWellSymbolsFile**: optional, allows user to specify the directory where user defined well symbol file is located.
- **ActiveGroup**: If `UserWellSymbolsFile` is used, this parameter must be set to specify which symbol group to be used as active group.

Note: To run the sample program you must be connected to the Studio database as a Petrel super user or a repository administrator. If you use Windows authentication to connect, the user ID and the password are ignored.

Preliminary steps

Before you can run the sample program to test your environment, create an empty repository in Studio. You will use the sample program to populate the repository with data. Refer to the Studio documentation for information about how to create a repository.

Note: The suggested storage CRS for the empty repository can be set to WGS84. The recommended project CRS can be set to ED50-UTM31.

Note: If you use Ocean development framework license to run the program, make sure the Studio repository is created at **Development** access level. If the Studio repository is created at **Commercial** access level, you will need Runtime for Studio license to run the program.

Run the sample program

To test the Studio SDK API environment.

1. Edit the configuration file named:

   Slb.Ocean.Studio.Data.Petrel.Sample.exe.config

   to point to your Studio database and repository.
2. Update the **ConnectionSpecificationConfiguration** and **appSetting** sections to substitute actual values for the parameters listed in "About the configuration file" on page 3-2.

   The following figures show these sections for an Oracle database configuration and a SQL Server database configuration. The Oracle database information is commented out. Be sure to edit and comment out or uncomment the information that applies to your database setup.

3. Execute the sample program:

   Slb.Ocean.Studio.Data.Petrel.Sample.exe

4. While the sample program runs, a command prompt displays messages that the Studio database is populated with test data.

5. When the application finishes, use Petrel or Studio Manager to see if your test repository was populated.

If the application runs successfully, you know that your environment is set up correctly for you to begin work with the Studio SDK API.

```
  <!-- Configuration for database connections, the following is 
the example for oracle data source setting -->
  <ConnectionSpecificationConfiguration>
    <ConnectionSpecification Name="BaseHost" Provider="Oracle" 
UseIntegratedSecurity="false">
        <add Id="1" HostName="basehost" Port="1521" 
ServiceName="sks"/>
    </ConnectionSpecification>
  </ConnectionSpecificationConfiguration>
  <appSettings>
    <add key="dataSource" value="BaseHost"/>
    <add key="userId" value="BaseUser"/>
    <add key="password" value="BaseUser"/>
    <add key="repositoryName" value="BaseProject"/>
    <add key="UserWellSymbolsFile" 
value="Data\UserSymbolFile.xml" />
    <add key ="ActiveGroup" value ="UserSymbols"/>
    <add key="ClientSettingsProvider.ServiceUri" value=""/>
  </appSettings>
```

Figure 3-1 Slb.Ocean.Studio.Data.Petrel.Sample.exe.config for Oracle

```
  <!-- Configuration for database connections, the following is 
the example for sql server data source setting -->
  <!-- When using SQL Server, it is required to use windows 
authentication, keeping the following argument userId and password 
as default value provided below. -->
  <ConnectionSpecificationConfiguration>
    <ConnectionSpecification Name="BaseHost" Provider="SQL 
Server" UseIntegratedSecurity="true">
      <add Id="1" HostName="basehost" Port="1433" 
ServiceName="sks" />
    </ConnectionSpecification>
  </ConnectionSpecificationConfiguration>
  <appSettings>
    <add key="dataSource" value="BaseHost" />
    <add key="userId" value="/" />
    <add key="password" value="" />
    <add key="UseIntegratedSecurity" value="true"/>
    <add key="repositoryName" value="BaseProject" />
    <add key="UserWellSymbolsFile"
value="Data\UserSymbolFile.xml" />
    <add key ="ActiveGroup" value ="UserSymbols"/>
    <add key="ClientSettingsProvider.ServiceUri" value="" />
  </appSettings>
```

Figure 3-2 **Slb.Ocean.Studio.Data.Petrel.Sample.exe.config** for SQL server

Build the sample application

In this task, you will test the Visual Studio development environment's ability to build and run the test sample application. You can find the files for this sample program in the directory named:

/Samples/SDK Samples/DomainObjectSample

Note: Ensure that the licenses are set up and that you have a valid license server, and Runtime for Studio is installed.

Preliminary steps

Before you can build and run the sample program to test your development environment, you must create an empty repository in Studio. You will use the sample program to populate the repository with data.

Refer to the Studio documentation for information about how to create a repository.

Note: The suggested storage CRS for the empty repository can be set to **WGS84**. The project CRS can be set to **ED50-UTM31**.

Note: If you use Ocean development framework license to run the program, make sure the Studio repository is created at **Development** access level. If the Studio repository is created at **Commercial** access level, you will need Runtime for Studio license to run the program.

Build and run the sample application

1. In Microsoft Visual Studio, open the sample project file named:

   Slb.Ocean.Studio.Data.Petrel.Sample.sln
2. Set your platform target to 64 bit.
3. Edit the App.config file as needed and described in "About the configuration file" on page 3-2.
4. Build and run the sample application as you would any other Visual Studio program.

Note: You can use the same configuration file that you used to run the prebuilt sample application.

5. If you can build and run the sample application, you are ready to create your own applications.

Create your own application

Use this procedure to create your own application.

To create your own application.

1. Create your own application based on the sample code.
2. Add a reference to the **Slb.Ocean.Studio.Data.Petrel.Setup.dll**. (This file is in the **Binaries** folder of the Ocean for Studio distribution package). The assembly reference must have the **Copy Local** attribute set to True.

Most applications also need these Runtime for Studio and Ocean Core assemblies, which reside in the Runtime for Studio installation folder.

- **Slb.Ocean.Studio.Data.Petrel.dll**
- **Slb.Ocean.Core.dll**
- **Slb.Ocean.Coordinates.dll**
- **Slb.Ocean.Units.dll**

Any assemblies referenced from the Runtime for Studio installation folder must have the **Copy Local** attribute set to False.

3. At the beginning of your application, add a call to **DomainObjectSetup.Initialize**. This method locates and loads the required assemblies from the Runtime for Studio installation directory. It also performs most other runtime initialization.
4. Distribute **Slb.Ocean.Studio.Data.Petrel.Setup.dll** with your application.

Note: See the Ocean and Studio documentation for more information about other assemblies that your application may require.

The **Slb.Ocean.Studio.Data.Petrel.Setup.dll** assembly is included to enable the application to find the Runtime for Studio installation, load the binaries, and initialize the Ocean for Studio services at run time. This is the only assembly that is required to be repackaged in the installer.

Note: The **DomainObjectSetup.Initialize** method must be called before any other Studio SDK API methods or properties are accessed. Other Studio SDK API data types must not be accessed from within the same method as the **DomainObjectSetup.Initialize** or the object references will not be properly resolved, and an exception will be thrown.

Enable SQL logging

This task enables SQL logging.

To turn on SQL logging.

1. Create the **Slb.Ocean.Studio.Data.Petrel.Setup.dll.config** configuration file or modify the existing **Slb.Ocean.Studio.Data.Petrel.Setup.dll.config** file.

The **Slb.Ocean.Studio.Data.Petrel.Setup.dll.config** file contains the parameter configuration for **`PdbSdkEnableSqlTracing`** and **`PdbSdkEnableSqlTracingAllParameters`**, which enable SQL logging:

PdbSdkEnableSqlTracing	Set to True to enable SQL tracing
PdbSdkEnableSqlTracingAllParameters	Set to True to enable SQL tracing for all the parameters

This code sample shows how to configure the DomainObjectSample application.

```
<configuration>
  <appSettings>
    <!-- Set PdbSdkEnabledSqlTracing to true to enable SQL tracing
-->
    <add key="PdbSdkEnableSqlTracing" value="true"/>
    <!-- Set PdbSdkEnableSqlTracingAllParameters to true to
enable SQL for all the parameters -->
    <add key="PdbSdkEnableSqlTracingAllParameters"
value="false"/>
  </appSettings>
</configuration>
```

2. Create an **App.config** file or modify the existing **App.config** file with the logging configuration section.

   This section contains the configuration for logging, in addition to the other application configurations.

   The logging configuration section contains the configuration information for the log file, formatter, trace source, and trace listener.

<listeners>	Contains the configuration of the Trace listener and output log file location
<formatters>	Contains the configuration for the formatter of the output log file
<categorySources>	Contains the configuration of the trace source
<specialSources>	Contains the configuration of special trace source

As this example shows, the DomainObjectSample application creates a log file named PdbSdkSql.log in the DomainObjectSample executable location.

```
  <configSections>
    <section name="loggingConfiguration"
type="Microsoft.Practices.EnterpriseLibrary.Logging.Configurat
ion.LoggingSettings,
Microsoft.Practices.EnterpriseLibrary.Logging"/>
  </configSections>
  <!-- Configuration for logging -->
  <loggingConfiguration name="Logging Application Block"
tracingEnabled="true" defaultCategory="General"
logWarningsWhenNoCategoriesMatch="true">
    <listeners>
```

```
      <add source="Enterprise Library Logging" formatter="Text
Formatter" log="Application" machineName=""
listenerDataType="Microsoft.Practices.EnterpriseLibrary.Loggin
g.Configuration.FormattedEventLogTraceListenerData,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
traceOutputOptions="None"
type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListe
ners.FormattedEventLogTraceListener,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
name="Formatted EventLog TraceListener"/>
      <!-- Log the PdkSdkSql trace logs in the PdbSdkSql.log -->
      <add fileName="PdbSdkSql.log" header="" footer=""
formatter="Text Formatter"
listenerDataType="Microsoft.Practices.EnterpriseLibrary.Loggin
g.Configuration.FlatFileTraceListenerData,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
traceOutputOptions="None"
type="Microsoft.Practices.EnterpriseLibrary.Logging.TraceListe
ners.FlatFileTraceListener,
Microsoft.Practices.EnterpriseLibrary.Logging, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35"
name="FlatFile TraceListener for PdbSdkSql"/>
    </listeners>
    <formatters>
      <!-- Formatter for the log file -->
      <add template="{timestamp} GMT {category} [{machine}
{appDomain}]  {message}"
type="Microsoft.Practices.EnterpriseLibrary.Logging.Formatters
.TextFormatter, Microsoft.Practices.EnterpriseLibrary.Logging,
Version=6.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" name="Text Formatter"/>
    </formatters>
    <categorySources>
      <!-- Configures the trace source(pdbsdksql) and trace
listener(Flat file TraceListener) -->
      <add switchValue="All" name="PdbSdkSql">
        <listeners>
          <add name="FlatFile TraceListener for PdbSdkSql"/>
        </listeners>
      </add>
    </categorySources>
    <specialSources>
      <allEvents switchValue="All" name="All Events"/>
      <notProcessed switchValue="All" name="Unprocessed
Category">
        <listeners>
          <add name="Formatted EventLog TraceListener"/>
        </listeners>
      </notProcessed>
      <errors switchValue="All" name="Logging Errors &amp;
Warnings">
        <listeners>
          <add name="Formatted EventLog TraceListener"/>
        </listeners>
      </errors>
    </specialSources>
  </loggingConfiguration>
```

Note: Alternatively, you can redirect the log file to your user profile:

%*USERPROFILE*%\AppData\Roaming\Schlumberger\Studio\...\PdbSdkSql.log

Distribute your application

After you create an application and are ready to distribute it, you need to prepare a group of files to distribute with the application. This group of files includes:

- Your program files
- Runtime for Studio
- Microsoft files

Include your program files

Include any files, such as images or data, that your program needs to execute. The **Slb.Ocean.Studio.Data.Petrel.Setup.dll** must also be included here.

Include Runtime for Studio

Runtime for Studio must be installed on the target computer.

Note: Because you are developing the application with Ocean for Studio 2020, Runtime for Studio 2020 must be installed on the target computer.

Include Microsoft files

Ensure that your users' target computer has installed Microsoft Visual C++ Redistributable for Visual Studio 2010, 2012, 2013, 2019 installed. They are installed along with Runtime for Studio.

Follow best practices

Follow these best practices when you use Studio SDK API to develop applications.

- Polylines are stored as defined by the storage CRS in the Studio repository. If the storage CRS differs from the original CRS, CRS conversion is performed on the original data before it is saved to the repository. Because of the CRS conversion, when retrieving points for PolylineSet objects that use the original CRS, the retrieved data may differ slightly when compared to the original dataset. This is expected behavior and should be taken into consideration.
- For Surfaces, the **`Domain`** property must have a valid value of **`ELEVATION_DEPTH`** or **`ELEVATION_TIME`** before you set the **`HorizonType`** property.
- Ocean for Studio uses Canonical/SI units unless the API explicitly has an option for unit's preference and another unit type has been selected. For DateTime related properties, the value is given as the local time zone.
- When a deletion operation occurs, the ClearCache operation is performed immediately after a successful repository save (implemented in API **`RepositoryService.SaveRepository`**). The save and ClearCache operation address an issue with deletions and cache inconsistencies. In consequence, all previous references obtained from the repository must be re-initialized for the Studio SDK before they are reused.
- When reading large amount of data using the Studio SDK, memory may increase while looping through data objects and not be released until the database connection is closed. Periodically calling the ClearCache operation (implemented in API **`RepositoryService.ClearCache`**) followed by the force garbage collection (System API **`GC.Collect`**) will release the memory cached by the Studio SDK.
- Single sign on may be implemented for Studio with Oracle database. For an application developed with the Studio SDK API, to connect to a Studio Oracle database via Single sign on, Windows authentication mode should be used. Sometimes the application may run into connection problem and throw Oracle exceptions such as "*TNS: could not resolve service name*" or "*invalid username/password; logon denied*". This may be due to the Oracle environment variables (**TNS_ADMIN** and **ORACLE_HOME**) are not set on the client machine running the application. In case when the Oracle client is not configured on local disk, setting the environment variables to the network paths (using System API **`Environment.SetEnvironmentVariable`**) at beginning of the application code may solve the connection problem.

For more information about data validation, see the Ocean for Studio data validation appendix.

4 LINQ and extension framework

In this chapter

LINQ

This section provides a high-level description of Language-Integrated Query (LINQ) in the context of the Studio SDK API. For more detailed information about LINQ, refer to the Microsoft Developer Network (http://msdn.microsoft.com).

What is LINQ?

LINQ is a Microsoft .NET component that extends .NET languages to include native data-query capabilities. LINQ enables you to use the same data-query language regardless of the database type (XML, SQL, etc.).

This example shows how you can use LINQ with the Studio SDK API:

```
public static void Main(string[] args)
{
  String repositoryName;
  DataConnectionContext dataConnectionContext;
  GetConnectionInfo(args, out dataConnectionContext, out
repositoryName);

  RepositoryService ps = DatabaseSystem.RepositoryService;
  using (Repository project =
      ps.OpenRepository(dataConnectionContext, repositoryName))
  {
    LinqExample(project);
  }
}

private class Projection
{
  public string Name { get; set; }
  public string UWI { get; set; }
}

private static void LinqExample(Repository project)
{
  string[] uwis = {"a", "b", "c"};
  IEnumerable<Projection> resultsUsingLinqToObject = null;
  {
    IQueryable<Projection> queryUsingLinqToObject =
        from b in GetBoreholes(project, false)
        where b.Cost > 100.0
        &&
        b.BoreholeCollection.ParentBoreholeCollection.Name
        .StartsWith("a")
        ||
        b.Logs.WellLogs.Any(wl => wl.Comments.Contains("sink"))
        ||
        !uwis.Contains(b.UWI)
        orderby b.Cost descending
        orderby b.Name ascending
        select new Projection() {Name = b.Name, UWI = b.UWI};

        resultsUsingLinqToObject = queryUsingLinqToObject
                                   .ToList();
  }

  IEnumerable<Projection> resultsUsingLinqToSql = null;
  {
    IQueryable<Projection> queryUsingLinqToSql =
        from b in GetBoreholes(project, true)
        where b.Cost > 100.0
        &&
        b.BoreholeCollection.ParentBoreholeCollection.Name
        .StartsWith("a")
        ||
        b.Logs.WellLogs.Any(wl => wl.Comments.Contains("sink"))
        ||
        !uwis.Contains(b.UWI)
        orderby b.Cost descending
        orderby b.Name ascending
        select new Projection() {Name = b.Name, UWI = b.UWI};

        resultsUsingLinqToSql = queryUsingLinqToSql.ToList();
  }
}

private static IQueryable<Borehole>
        GetBoreholes(Repository project, bool useLinqToSql)
{
  if (useLinqToSql)
    return project.Queryables.Well.AllBoreholes;

  List<Borehole> boreholes = new List<Borehole>();
  BoreholeCollection topBC = WellRoot.Get(project)
                             .BoreholeCollection;
  if (topBC != null)
  {
    Queue<BoreholeCollection> boreholeCollections =
                              new Queue<BoreholeCollection>();
    boreholeCollections.Enqueue(topBC);
    while (boreholeCollections.Count > 0)
    {
      BoreholeCollection bc = boreholeCollections.Dequeue();
      foreach (BoreholeCollection childBC in
                 bc.BoreholeCollections)
        boreholeCollections.Enqueue(childBC);
      boreholes.AddRange(bc.Boreholes);
    }
  }
  return boreholes.AsQueryable();
}
```

Note: The sample application includes a more elaborate LINQ example (**LINQExamples.cs**) with comments in the code.

LINQ techniques

To use LINQ successfully, you should understand these concepts and techniques:

- Deferred execution
- **`IQueryable`** and **`IEnumerable`**
- Lazy loading
- Compiled queries

Deferred execution

During deferred execution, a database query is parsed and the parameters are bound and set before the query is actually executed. Deferred execution can greatly improve performance when you work with large amounts of data. Instead of multiple passes through the data to query information, deferred execution can allow a single iteration through the data to query the same information.

Ensure you understand the concept of deferred execution and how queries are actually generated and executed before you use LINQ. Composable queries and deferred execution work together to make LINQ a rich query language. If you understand these features of LINQ you will be able to write fewer lines of code that are more efficient.

IQueryable and IEnumerable

Two extension methods enable LINQ to access data in a data table. One method exposes the data as **`IQueryable`** and the other exposes data as **`IEnumerable`**.

When it comes to composing LINQ queries, understanding how **`IQueryable`** works is essential. Discussions about LINQ include many references to **`IEnumerable`** because it is the base from which everything else draws. As long as you have something that implements **`IEnumerable`**, you can use LINQ.

`IEnumerable` doesn't have the concept of moving between items; it is a forward only collection. For example, if you want to use LINQ to find duplicates in a large collection and perform a join on this same collection, using the forward-only model of **`IEnumerable`** will become expensive, especially if you start to combine operations. However, **`IQueryable`** adds two methods that make this much easier:

- **`CreateQuery`**

- **`Execute`**

These methods accept an expression, which is basically a new set of classes that allow you to express things like method calls, operations, lambda functions, etc. as a tree of expressions. This is not just for simple expressions but for the entire LINQ query. You can look inside the LINQ query and complete tasks with the expression tree.

.NET guidelines

Observe these guidelines when you write queries:

- .NET processes **`IEnumerable<T>`** and **`IQueryable<T>`** differently. The runtime processes an expression that contains **`IEnumerable`** first, thus turning it into an in-memory **`IEnumerable<T>`**, and then processes the rest of the expression using LINQ to object. You should modify the code to tell the .NET runtime how to execute the query more efficiently.
- **First()** queries do not have a **where** clause, which causes the query to scan the entire table. Insert a **where** clause to improve query efficiency.

Lazy loading

By default, LINQ to SQL loads related entities and entity sets that use lazy loading. This means that if you retrieve a list of entities in a query that references other entities, these other entities are not actually loaded unless they are accessed.

Compiled queries

Compiled queries can improve performance if a specific query is executed repeatedly. Compiled queries in LINQ provide these benefits:

- The query must be compiled each time so execution of the query is fast.
- The query is compiled once and can be used any number of times.
- The query must be recompiled even if the parameter of the query is changed.

Repository.Queryables

LINQ queries that use **`Repository.Queryables`** as a starting point are executed on the server side. Therefore, be careful that you do not perform a **`Repository.Queryables`** query while the cache contains unsaved changes that have not been committed to the database. Unsaved cache changes will not be reflected in the results of the **`Repository.Queryables`** query.

Database columns

The database contains columns. Some columns are indexed and some are not.

Table 4-1 Database column status

Name	Indexed
`UniqueID`	No

Supported LINQ queries

The following table shows the LINQ query operators that Ocean for Studio supports.

Table 4-2 LINQ query operators

Operator	Order	Mandatory	Syntax	Example
Type selection	1	Yes	`from item in <queryables>`	`from b in project.Queryables.Well.AllBoreholes`
Conditions	2	No	`where <predicate for item>`	`where b.BoreholeCollection.Name.StartsWith("A 10")`
Sorting	3	No	`orderby <property selector for item> <asc/desc>`	`orderby b.UWI asc`
Projection	4	No	`select <projection for item>`	`select new { UID = b.UniqueId, UWI = b.UWI }`
Set(Distinct)	5	No	`.Distinct()` with no predicates	`(from b in project.Queryables.Well.AllBoreholes Select b.Name).Distinct()`
Windowing	6	No	`.Skip(<number of item>).Take(<number of item>)`	`.Skip(5).Take(100)`
Counting	7	No	`.Count()` with no predicates	`.Count()`
Existence	7	No	`.Any()` with no predicates	`.Any()`
First	7	No	`.First()` or `.FirstOrDefault()` with no predicates	`.First()` or `.FirstOrDefault()`

Best practices for executing queries

Remember that links are not pre-fetched for LINQ query executions. For example, accessing the borehole property for a checkshot involves fetching the specific borehole object. If this operation takes place in a loop, then the boreholes are fetched one at a time, which could lead to slower performance.

Follow these recommended best practices when executing LINQ queries:

- Fetch items in batches.
- Properties of domain objects used in LINQ queries must be LINQ enabled. (Check for [**`LinqEnabledAttribute`**] on properties.)

- Use Projection for needed properties where possible. Properties that are links to other domain objects cannot be projected, e.g. the borehole property for a checkshot. (These properties can be used in constraints.)
- Remember that the default behavior is a case-insensitive comparison for all string data types.

The extension framework

This section provides a high-level description and examples of the extension framework of Studio SDK API.

The extension API is a new property access pattern introduced in Ocean for Studio 2014.1, complementary to the Ocean attribute pattern (or referred as the old property access API). It is used to access custom attributes and properties, as well as well-known attributes and properties of domain objects. Use the new extension API for any new functionality.

The old property access API is based on Ocean for Petrel. Starting from 2014.1, the old property access APIs for **`Borehole`** and **`Marker`** become Obsolete.

Properties can be either continuous or dictionary, custom or well-known, single-valued or multi-valued. For example, **`UWI`** is a dictionary, well-known, single-valued property of **`Borehole`**; and **`AverageVelocity`** is a continuous, custom, multi-valued property of **`Checkshot`**. To learn more about the old property access API, please refer to chapter **Domain object attributes** of Volume 3 **Basic Concepts of Ocean Plug-ins** of Ocean for Petrel Developer's Guide.

IExtension

Every object that supports custom attributes or properties has an **`Extension`** property of type **`IExtension`** (for example, **`Borehole.Extension`**).

The **`IExtension`** interface provides generic access on custom properties and attributes through property **`Features`**:

```
public interface IExtension
{
  IEnumerable<IFeature> Features { get; }
  IFeature this[string uniqueId] { get; }
}
```

IAttributeExtension

Every object that supports only customized attributes has an **`Extension`** property of type **`IAttributeExtension`** (for example, **`BoreholeCollection.Extension`**).

The **`IAttributeExtension`** interface replaces the **`ICustomAttributeService`** interface (become Obsolete since 2014.1). It provides generic access for CRUD operations on custom attributes:

```
public interface IAttributeExtension : IExtension
{
  void DeleteAttribute(string ns, string name);
  IEnumerable<IAttribute> Attributes { get; }
  ITypedFeature<T> ResolveAttribute<T>(string uniqueId);
  ITypedFeature<T> FindAttribute<T>(string ns, string name);
  ITypedFeature<T> CreateAttribute<T>(string ns,
                                      string name);
}
```

IPropertyExtension

Every object that supports customized properties has an **Extension** property of type **IPropertyExtension** (for example, **Borehole.Extension**).

These domain objects support the **IPropertyExtension** type:

- **SeismicCube**
- **SeismicLine2DCollection**
- **Seismic2DCollection**
- **Seismic3DCollection**
- **HorizonInterpretation2D**
- **HorizonInterpretation3D**
- **FaultInterpretation**
- **WellLog**
- **DictionaryWellLog**
- **BitmapWellLog**
- **MultiTraceWellLog**
- **BoreholeSeismogram**
- **CommentWellLog**
- **Borehole**
- **Marker**

The **IPropertyExtension** interface inherits from the **IAttributeExtension** interface. It provides generic access for CRUD operations on custom attributes and custom properties.

```
public interface IPropertyExtension : IAttributeExtension
{
  IEnumerable<IProperty> Properties { get; }
  IEnumerable<IPrototype> Prototypes { get; }
  ITypedFeature<T> ResolveProperty<T>(string uniqueId);
  IEnumerable<IFeature> FindProperties(string name);
  ITypedFeature<T> FindProperty<T>(string name);
  ITypedFeature<T> CreateProperty<T>(string name);
  ITypedFeature<T> CreateProperty<T>(string name,
                                     string uniqueId);
  ITypedFeature<T> CreateProperty<T>(string name,
                                     string uniqueId,
                                     string templateId);
  void DeleteProperty(string uniqueId);
}
```

IPropertyManager

IPropertyManager and **PropertyDefinition** are introduced in Ocean for Studio 2016.2 to enhance the extension API. It is used to solve two problems of the extension API:

- An instance of a domain object is required to create properties.
- Property value can only be set one at a time.

The **IPropertyManager** interface provides generic access and creation of **PropertyDefinition**.

```
public interface IPropertyManager
{
  PropertyDefinition CreatePropertyDefinition(Type dataType,
                                              string propertyName);
  PropertyDefinition CreatePropertyDefinition(Type dataType,
                                              string propertyName,
                                              string templateId);
  PropertyDefinition CreatePropertyDefinition(string uniqueId,
                                              Type dataType,
                                              string propertyName);
  PropertyDefinition CreatePropertyDefinition(string uniqueId,
                                              Type dataType,
                                              string propertyName,
                                              string templateId);
  IEnumerable<PropertyDefinition> GetPropertyDefinitions();
}
```

The **PropertyDefinition** class represents both custom and well-known properties and supports reading and updating property values in batch.

```
public class PropertyDefinition
{
  string UniqueId         { get; private set; }
  string Name             { get; private set; }
  Type   DataType         { get; private set; }
  string TemplateUniqueId { get; private set; }
  DomainObjectBase Owner  { get; private set; }
  Dictionary<string, object> GetValues(IList<string> parentIds);
  void SetValues(Dictionary<string, object> new Values);
}
```

Note: Only **BoreholeCollection** implements the **IPropertyManager** interface, therefore only domain object **Borehole** is supported with the new property API.

Examples

These examples use the extension API.

Create a custom property and set the value

This example creates a custom string property named **`CustomPropertyName`**. You can use overloaded methods to specify uniqueId and templateId for the new property.

```
ITypedFeature<string> customProperty =
                      borehole.Extension.CreateProperty<string>
                      ("CustomPropertyName");
customProperty.Value = "PropertyValue";
```

Access an existing property

These examples use the generic extension API to access an existing string property.

- Get a property by UniqueId

```
ITypedFeature<string> customResolveProperty =
                      borehole.Extension.ResolveProperty<string>
                      ("CustomPropertyUniqueId");
```

- Get a property by Name

```
ITypedFeature<string> customFindProperty =
      borehole.Extension.FindProperty<string> ("PropertyName");
```

Note: Name is not unique for object properties, so this method returns the first property or it returns null if no properties are found. To find all properties assigned to the object with the specified name, use:

```
IEnumerable<IFeature> properties =
      borehole.Extension.FindProperties("PropertyName");
```

Unset value for a custom property

This example uses the generic extension API to unset value for a custom property:

```
borehole.Extension.DeleteProperty("CustomPropertyUniqueId");
```

Note: The **`DeleteProperty`** method only unset property value for this borehole, the property definition still exists.

Create a custom attribute

This example uses the generic extension API to create a custom attribute:

```
ITypedFeature<string> customAttribute =
                     borehole.Extension.CreateAttribute<string>
                     ("namespaceName", "attributeName");
customAttribute.Value = "Value";
```

Read an existing attribute and set the value

This example uses the generic extension API to read a custom string attribute and set the value:

```
ITypedFeature<string> attribute = ex.FindAttribute<string>
          ("customAttributeNamespace", "customAttributeName");
```

```
attribute.Value = "new value";
```

Delete a custom attribute

This example uses the generic extension API to delete a custom attribute:

```
borehole.Extension.DeleteAttribute("namespaceName",
                                   "attributeName");
```

Use LINQ with custom properties

This example uses LINQ with custom properties:

```
IQueryable<Borehole> extensionQuery =
       from boreholes in project.Queryables.Well.AllBoreholes
       where borehole.Extension.FindProperty<string>
             ("PropertyName").Value == "PropertyValue"
       select borehole;
```

Create a custom borehole property from top-level well folder

This example uses **`IPropertyManager`** to create a custom string property for **`Borehole`**:

```
BoreholeCollection bcRoot =
    WellRoot.Get(project).GetOrCreateBoreholeCollection();
PropertyDefinition newPropDef = bcRoot.CreatePropertyDefinition
    (Type.GetType("System.String"),
     "NewBoreholeStringProperty1");
```

Set value for a custom borehole property in batch

This example uses **`PropertyDefinition`** to set value of a custom property in batch for all the boreholes within a specific well folder:

```
BoreholeCollection bcDWells =
    FindBoreholeCollectionByName(project, "D Wells");
Dictionary<string, object> newPropValues =
    new Dictionary<string, object>();
foreach (Borehole b in bcDWells)
    newPropValues.Add(b.UniqueId, "NewStringValue");
newPropDef.SetValues(newPropValues);
```

Get all borehole properties and read property values in batch

This example uses **`IPropertyManager`** to get all the property definitions for domain object **`Borehole`**, including custom and well-known, also including properties created using the old property access API. It also retrieves value of each property for all the boreholes under a specific well folder:

```
IEnumerable<PropertyDefinition> propDefs =
    bcDWells.GetPropertyDefinitions();
IList<string> parentIds = new List<string>();
foreach (Borehole b in bcDWells)
    parentIds.Add(b.UniqueId);
```

```
// loop through property definitions and retrieve property values
foreach (PropertyDefinition propDef in propDefs)
{
    Dictionary<string, object> values =
        propDef.GetValues(parentIds);
}
```

Use well-known properties with the extension API

Well-known properties can be accessed as interface properties by using the extension API. For a domain object type with well-known properties, a specific interface provides generic access to its well-known properties. This example illustrates **`IMarkerEx`** interface for **`Marker`** domain object:

```
public interface IMarkerEx : IPropertyExtension,
      IAttributeExtension, IExtension
{ ...
  double ConfidenceFactor { get; set; }
  double Missing { get; set; }
  double TWTPicked { get; set; }
  bool UsedByDepthConversion { get; set; }
  bool UsedByGeoModeling { get; set; }
}
```

Set the value for well-known properties

ConfidenceFactor, Missing, and TWTPicked are well-known properties defined for Marker. This example uses the generic extension API to access them directly:

```
marker.Extension.ConfidenceFactor = confidenceFactor;
marker.Extension.Missing = missing;
marker.Extension.TWTPicked = twtPicked;
```

Use LINQ with well-known properties

```
IQueryable<Marker> querySample =
            from cmarker in project.Queryables.Well.AllMarkers
            where cmarker.Extension.TWTPicked == 0.2 &&
                  cmarker.Extension.ConfidenceFactor == 0.5
            select cmarker;
```

Use interface properties to access well-known properties

Interface properties and generic property access both behave as expected. However, use interface properties to access well-known properties is preferred instead of using a generic property access, as this example shows:

```
using (Repository project =
           DatabaseSystem.RepositoryService.OpenRepository
           (dataConnectionContext, repositoryName))
{
  WellRoot wellRoot = WellRoot.Get(project);
  BoreholeCollection boreholeCollection =
            wellRoot.GetOrCreateBoreholeCollection();
  Borehole bh =
            boreholeCollection.CreateBorehole("TestBorehole");

  //Preferred method
  String property1Val = bh.Extension.SimulationName;
  DateTime property2Val = bh.Extension.SimulationExportDate;

  //Generic method
  ITypedFeature<string> property1 =
       bh.Extension.ResolveProperty<string>("Simulation Name");
  ITypedFeature<DateTime> property2 =
       bh.Extension.ResolveProperty<DateTime>
      ("Simulation Export Date");
  property1Val = property1.Value;
  property2Val = property2.Value;

  DatabaseSystem.RepositoryService.SaveRepository(project);
}
```

Third-party well-known properties and attributes

Third-party developers could define well-known properties and attributes for third-party uses. Below example defines custom extensions for **`Borehole`** and **`Marker`** to specify third-party, well-known attributes and properties for boreholes and markers. Refer to class **`ThirdPartyRepositoryExSample`** of the sample application in chapter 3 for full details.

Declare extension interfaces for boreholes

```
namespace Slb.Ocean.Studio.Knowledge.Custom
{
  //Custom extension for Borehole, allows to specify third-party,
  //well-known attributes and properties for Borehole
  public interface IBorehole3DPartyEx : IBoreholeEx
  {
    [AttributeDefinition
            ("ThirdParty.WellKnownAttribute.CustomNamespace",
             "Borehole3dCustomAttributeWithCustomName")]
    string BoreholeUserDefinedCustomAttributeCN { get; set; }

    //if no definition is specified then property will be treated
    //as CustomAttribute with namespace = DeclaredType.FullName
    //and name = PropertyName
    //For current property namespace is
    //"Slb.Ocean.Studio.Knowledge.Custom.IBorehole3DPartyEx"
    //and name is "BoreholeUserDefinedCustomAttribute"
    string BoreholeUserDefinedCustomAttribute { get; set; }

    [PropertyDefinition("UserDefinedBoreholeProperty",
                        "UserDefinedBoreholeProperty",
                        "141D3D6322C84398AFD58284A3ADC136")]
    string BoreholeUserDefinedCustomProperty { get; set; }
  }
}
```

Declare extension interfaces for markers

```
namespace Slb.Ocean.Studio.Knowledge.Custom
{
  //Custom extension for Marker, allows to specify third-party,
  //well-known attributes and properties for Marker
  public interface IMarker3DPartyEx : IMarkerEx
  {
    [AttributeDefinition
            ("ThirdParty.WellKnownAttribute.CustomNamespace",
             "Marker3dCustomAttributeWithCustomName")]
    string UserDefinedWellKnownMarkerAttributeCN { get; set; }

    //if no definition is specified then property will be treated
    //as CustomAttribute with namespace = DeclaredType.FullName
    //and name = PropertyName
    //For current property namespace is
    //"Slb.Ocean.Studio.Knowledge.Custom.IMarker3DPartyEx"
    //and name is "UserDefinedWellKnownMarkerAttribute"
    string UserDefinedWellKnownMarkerAttribute { get; set; }

    [PropertyDefinition("UserDefinedMarkerProperty",
                        "UserDefinedMarkerProperty",
                        "F1A23E8F932148F7B244F4FAEF4ADEDA ")]
    string UserDefinedWellKnownMarkerProperty { get; set; }
  }
}
```

Implement a repository extender class

Use abstract class **`CustomDataExtenter`** to implement a repository extender class.

```
public abstract class CustomDataExtender
{
  protected void AddCustomExtension<TBase, TCustom>()
                      where TCustom : TBase;
  protected abstract void RegisterCustomExtensionsImpl();
  protected abstract void DefineTemplatesImpl
```

```
                        (Repository repository);
  protected TemplateCollection
       CreateWellKnownTemplateCollection(Repository repository,
                        string name, string uniqueId);
  protected void CreateWellKnownTemplate
       (TemplateCollection collection, string name,
        IUnitMeasurement unitMeasurement, IUnit unit,
        string uniqueId);
  protected void CreateWellKnownTemplate
       (TemplateCollection collection, string name,
       IUnitMeasurement unitMeasurement, string uniqueId);
}
```

Repository extender class example

This example defines a custom repository extender class by implementing abstract methods **`RegisterCustomExtensionsImpl`** and **`DefineTemplatesImpl`** of **`CustomDataExtender`.**

```
internal class RepositoryExtensionCustomizerSample :
                                             CustomDataExtender
{
  protected override void RegisterCustomExtensionsImpl()
  {
    AddCustomExtension<IBoreholeEx, IBorehole3DPartyEx>();
    AddCustomExtension<IMarkerEx, IMarker3DPartyEx>();
  }

  //Create all third-party well known templates which
  //are used in third-party custom properties
  protected override void DefineTemplatesImpl
                                   (Repository repository)
  {
    TemplateCollection collection =
              CreateWellKnownTemplateCollection
                          (repository,
                          "TestTemplateCollection",
                          "6A98F1BD981244C880EC91238DD9AD23");
    IUnitMeasurement unitMeasurement =
              CoreSystem.GetService<IUnitServiceSettings>()
              .CurrentCatalog.GetUnitMeasurement("Length");
    CreateWellKnownTemplate(collection,
                 "TestTemplateForBorehole", unitMeasurement,
                 "141D3D6322C84398AFD58284A3ADC136");
    CreateWellKnownTemplate(collection,
                 "TestTemplateForMarker", unitMeasurement,
                 "F1A23E8F932148F7B244F4FAEF4ADEDA");
  }
}
```

Register the custom repository extender class

You must register a custom repository extender class before you can use it.

Note: You must register the repository extension during the application initialization.

```
// Customize extension with third-party well known templates and
properties
using (Repository project =
            DatabaseSystem.RepositoryService.OpenRepository
            (dataConnectionContext, repositoryName))
{
  ExtensionProvider.CustomizeWith
                <RepositoryExtensionCustomizerSample>(project);
}
```

Access the marker property IsZoneProperty

Follow these steps to access the **`IsZoneProperty`** property for **`Marker`**.

1. Find or create a property for a marker, this can be any custom property:

```
ITypedFeature<string> markerProperty =
  marker.Extension.FindProperty<string>("MarkerPropName") ??
  marker.Extension.CreateProperty<string>("MarkerPropName");
```

Note: Always check if a property with the given name already exisits using the generic extension API, only create a new property if it returns null.

2. Convert this property to the **`IProperty`** interface:

```
IProperty markerProperty1 = markerProperty as IProperty;
```

3. Convert the prototype for this property to the marker specific prototype, **`IMarkerPropertyPrototype`**:

```
IMarkerPropertyPrototype markerPrototype =
markerProperty1.GetPrototype<IMarkerPropertyPrototype>();
```

4. Set or get the **`IsZoneProperty`** from the marker prototype:

```
markerPrototype.IsZoneProperty = true;
bool isZone = markerPrototype.IsZoneProperty;
```

Do not use the new extension API and the old API together

Do not create a property with the old property access API and then access the properties with the new extension API in the scope of one session. This example shows the invalid and valid access methods:

```
using (Repository project =
        DatabaseSystem.RepositoryService.OpenRepository
```

```
           (dataConnectionContext, repositoryName))
{
  WellRoot wellRoot = WellRoot.Get(project);
  BoreholeCollection boreholeCollection =
            wellRoot.GetOrCreateBoreholeCollection();
  BoreholePropertyCollection bpc =
            boreholeCollection.BoreholePropertyCollection;
  Borehole borehole =
            boreholeCollection.CreateBorehole("TestBorehole");

  //Create custom borehole properties using old API
  DictionaryBoreholeProperty bhProperty1 =
    bpc.CreateDictionaryProperty(typeof(string), "Property1");
  string bhProperty1UniqueId = bhProperty1.UniqueId;
  DictionaryBoreholeProperty bhProperty2 =
    bpc.CreateDictionaryProperty(typeof(string), "Property2");
  string bhProperty2UniqueId = bhProperty2.UniqueId;
  BoreholeProperty bhProperty3 =
    bpc.CreateProperty(typeof(DateTime), "Property3");
  string bhProperty3UniqueId = bhProperty3.UniqueId;

  //Invalid method, using new extension API
  ITypedFeature<string> property1 =
             borehole.Extension.ResolveProperty<string>
             (bhProperty1UniqueId);
  ITypedFeature<string> property2 =
             borehole.Extension.ResolveProperty<string>
             (bhProperty2UniqueId);
  ITypedFeature<DateTime> property3 =
             borehole.Extension.ResolveProperty<DateTime>
             (bhProperty3UniqueId);

  //Valid method, using old API
  borehole.PropertyAccess.
      SetPropertyValue(bhProperty1, "testValue1");
  borehole.PropertyAccess.
      SetPropertyValue(bhProperty2, "testValue2");
  borehole.PropertyAccess.
      SetPropertyValue(bhProperty3, new DateTime(1,1,1));

  DatabaseSystem.RepositoryService.SaveRepository(project);
}
```

5 Units in the Ocean for Studio

In this chapter

Overview of units in Ocean for Studio

Unit support

Unit support in Ocean for Studio is provided by the Ocean Services modules.

Unit types

The **Slb.Ocean.Units** assembly provides these unit types.

- Unit
- Unit measurement
- Unit system
- Unit catalog

Unit

A unit is a fundamental item that is a standard for the basis of measurement. A simple unit measures one value. For example, meters, feet, pounds, and kilograms are simple units. Compound units measure one value in relation to another value. For example, meters per second or pounds per square inch are compound units. To be meaningful, a measurement must have a unit label. For example, 12 and 55 have no meaning as measurements without the corresponding units (12 inches and 55 MPH).

Unit measurement

Units are grouped by categories of measurement. A unit measurement group of distance implies that units that do not measure length are not permitted because it makes no sense to express a distance in a unit such as kilograms. Unit measurements may be derived. For example, the vertical depth unit measurement is a subcategory of the distance unit measurement that has been restricted to being vertical. A unit measurement object lists its permitted units and refers to its base measurement.

Unit system

A unit system is a mapping between units and unit measurements. In the English unit system, a unit measurement of distance is expressed in units of feet. This is one reason for the subcategories of unit measurement. For example, a big distance might be expressed in miles, while a small distance might be expressed in inches.

Unit catalog

A unit catalog is a collection of units, unit measurements, and unit systems.

How to work with units

How to access units

The unit catalog has a method that returns an **IEnumerable** list of all unit measurements in the catalog. An object that supports the **IUnitMeasurement** interface has a **Units** member, which is an **IEnumerable** of the units associated with that UnitMeasurement. To access the Units property of a specific **UnitMeasurement** object, you must first access that **UnitMeasurement** object from the catalog. This example displays the units associated with the unit measurement of a particular template:

```
foreach (IUnit myUnit in myTemplate.UnitMeasurement.Units)
{
  Console.WriteLine("{0}", myUnit.Name);
}
```

Unit conversion

Unit conversion is not performed directly by the **Unit** object. The **CanConvert** method of the **IUnit** interface indicates if the particular unit belongs to the same base **UnitMeasurement** category as the parameter unit and if the particular unit is convertible to the parameter unit.

To convert the units, you must have an instance of an **IUnitConverter** object that is retrieved from the unit catalog. An **IUnitConverter** object converts one pair of units in one direction. For example, an **IUnitConverter** will convert from feet to meters but not from meters to feet. If you require two-way conversion, you must have two separate **IUnitConverter** objects.

You can access the unit catalog from the **IUnitService** interface, which has a collection of all available catalogs. (Currently, there is only the Oilfield Services Data Dictionary (OSDD) unit catalog.) You can also access the unit catalog through an **IUnitServiceSettings** object, which returns the current active unit catalog used by the application. Both of these can be retrieved with a call to the **CoreSystem.GetService<T>()** method.

This code demonstrates a simple unit conversion between **IUnit** objects oldUnit and newUnit:

```
IUnitCatalog UnitCatalog = CoreSystem.GetService
                      <IUnitServiceSettings>().CurrentCatalog;
IUnitConverter converter =
                   UnitCatalog.GetConverter(oldUnit, newUnit);
double dOriginalValue = 123.45;
double dNewValue = converter.Convert(dOriginalValue);
```

Unit service

Many of the Ocean for Studio routines rely heavily on the Ocean Core and Ocean Services implementations, which provide a common framework for a wide range of Schlumberger applications including Studio and Petrel. Using these common services ensures compatibility and rapid development while providing you with a consistent foundation. The Ocean foundation is used for units, coordinates, and geometry in the Ocean for Studio.

The Unit service is accessed with code similar to this example:

```
IUnitService unitService =
                     CoreSystem.GetService<IUnitService>();
```

Unit property and templates

The unit property cannot be changed in predefined templates. All of the built-in templates are immutable and cannot be altered. However, you can change templates that you create yourself. The **`Template`** class for continuous templates (or **`DictionaryTemplate`** class for discrete templates) in Studio is relatively sparse, containing a name and description and inheriting the **`UnitMeasurement`** from the **`PropertyVersion`** class. The **`UnitMeasurement`** defines the kinds of values that the template describes. The specific unit involved would normally be chosen by the presentation module rather than the underlying library code. Ocean for Studio uses the International System of Units (SI) units almost exclusively.

A template's presentation parameters are stored in a **`TemplateSettingsInfo`** object. The template itself does not have any public association to its **`TemplateSettingsInfo`**, but it can be retrieved by using the **`ITemplateSettingsInfoFactory`** service as in this example:

```
TemplateSettingsInfo info =
    CoreSystem.GetService<ITemplateSettingsInfoFactory>
    ().GetTemplateSettingsInfo(myTemplate);
```

The **`TemplateSettingsInfo`** object allows direct access to the template's presentation parameters, including its unit. The predefined templates's presentation parameters cannot be modified.

Note: Ocean for Studio does not state that the unit can be set through this object, but it is settable. You must check your template's **`IsWritable`** property before attempting to modify it, or an **`InvalidOperationException`** will be thrown. This behavior could change in future releases.

Assembly references

All required Ocean and Studio assemblies are installed in the Runtime for Studio directory. As with all added assembly references, they should not be copied to a local directory. The **`DomainObjectSetup`** component manages the locating and loading of the assemblies at runtime.

The required assemblies for working with units include these references:

- **Slb.Ocean.Core.dll**: Provides **`CoreSystem`**, used for retrieving references to **`IUnitService`** and **`IUnitServiceSettings`** for accessing current catalogs.
- **Slb.Ocean.Units.dll**: Used for retrieving references to:

- **`IUnitService`**
- **`IUnitServiceSettings`**
- **`IUnitCatalog`**
- **`IUnitSystem`**
- **`IUnitMeasurement`**
- **`IUnitConverter`**
- **`IUnit`**

The reference to **Slb.Ocean.Studio.Data.Petrel.dll** is assumed, and although it is not strictly necessary to use the Ocean unit services, it is required for the **`DomainObjectSetup`** component and any other Studio data access-related workflows.

Unit and measurement sources

Several sources for units and measurement information are available:

- Oilfield Services Data Dictionary (OSDD)
- Ocean Unit Service
- Ocean Documentation

OSDD

The OSDD unit catalog is the source of the unit systems, unit measurements, and units used in the Ocean core. The Curve Mnemonic Dictionary is the publicly accessible version of the OSDD. The database delivers descriptions of more than 50,000 Schlumberger logging tools, analytical software, and log curves with parameters. It also provides definitions of physical property measurements and relevant units of measurement. Special tables enumerate mineral properties and depositional environments.

The Curve Mnemonic Dictionary is accessible at this URL:

http://www.apps.slb.com/cmd/

Ocean unit service

The Ocean unit service consists of a unit catalog, with units and unit measurements defined by OSDD. The service offers a number of convenient methods that let you convert values between units of the same unit measurement. The unit service also offers a number of predefined unit systems.

The API description of the Ocean unit service can be accessed through **Slb.Ocean.Studio.Data.Petrel.chm** of Ocean for Studio online help.

Ocean documentation

The Ocean Core and Services documentation (can be accessed through Ocean for Petrel Developer’s Guide) has information about the interfaces themselves, but the actual contents of the OSDD unit catalog are not directly documented. You can write a simple program to exercise the interfaces of the Ocean unit service and make a data dump of all of the available systems, measurements, and units.

Example

This code sample shows several different examples of working with units in Ocean for Studio.

```
using System;
using System.Configuration;
using System.IO;
using System.Linq;
using System.Security;
using Slb.Ocean.Core;
using Slb.Ocean.Units;
using Slb.Ocean.Studio.Data.Petrel;
using Slb.Ocean.Studio.Data.Petrel.UI;

namespace UnitExamples
{
  class Program
  {
    private Repository Project { get; set; }
    private static DataConnectionContext dataConnectionContext;

    static void Main(string[] args)
    {
      Console.WriteLine("Initializing DomainObject");
      DomainObjectSetup.Initialize();
      Console.WriteLine("Successfully initialized 
DomainObject");
      Program p = new Program();
      p.Execute();
    }

    void Execute()
    {
      string userId = "BaseUser";
      string password = "BaseUser";
      string repositoryName = "BaseRepository";
      dataConnectionContext =
        DataConnectionContext.Create("BaseHost:1521/sds2008",
                   userId, ToSecuredString(password));

      Console.WriteLine("Synchronizing coordinate catalog...");
      DatabaseSystem.CoordinateService
    .SynchronizeCoordinateCatalog(dataConnectionContext, true);

      Console.WriteLine("Opening repository...");
      Project = DatabaseSystem.RepositoryService.
        OpenRepository(dataConnectionContext, repositoryName);
      Console.WriteLine("Describing unit systems...");
      DescribeUnitSystems();
      Console.WriteLine("Enumerating templates...");
      GetTemplateCollections();
      DatabaseSystem.RepositoryService.
          SaveRepository(Project);
    }

    static SecureString ToSecuredString
                                  (string unsecuredString)
    {
        if (string.IsNullOrEmpty(unsecuredString))
            return null;
        var secureString = new SecureString();
        foreach (var symbol in unsecuredString)
        {
            secureString.AppendChar(symbol);
        }
        secureString.MakeReadOnly();
        return secureString;
    }

    void DescribeUnitSystems()
    {
      IUnitService unitService =
                          CoreSystem.GetService<IUnitService>();
      foreach (var catalog in unitService.Catalogs)
      {
        Console.WriteLine("Unit catalog {0}", catalog.Name);
        foreach (var system in catalog.Systems)
        {
          Console.WriteLine("\tUnit System: {0}", system.Name);
        }
      }
      IUnitSystem unitSystem =
          unitService.GetCatalog("OSDD").GetSystem("English");
    }

    void GetTemplateCollections()
    {
      foreach (var tc in Project.TemplateCollections)
      {
        Console.WriteLine("Template Collection {0}", tc.Name);
        if (tc.Name == "Well log templates")
        {
          AddATemplate(tc);
          DoSomethingWithTemplateCollection(tc);
        }
      }
    }

    void AddATemplate(TemplateCollection tc)
    {
      if (tc.Templates.Select(t => t.Name ==
"MyTemplate").Count() == 0)
      {
        IUnitCatalog canonicalUnitCatalog = CoreSystem
           .GetService<IUnitServiceSettings>().CurrentCatalog;
        IUnitMeasurement unitMeasurement =
           canonicalUnitCatalog.GetUnitMeasurement("Density");
        tc.CreateTemplate("MyTemplate", unitMeasurement);
      }
    }

    void DoSomethingWithTemplateCollection
                                    (TemplateCollection tc)
    {
      IUnitCatalog canonicalUnitCatalog = CoreSystem
         .GetService<IUnitServiceSettings>().CurrentCatalog;
      foreach (var t in tc.Templates)
      {
        if (t.Name == "MyTemplate")
        {
          TemplateSettingsInfo info = CoreSystem.GetService
              <ITemplateSettingsInfoFactory>(typeof(Template))
              .GetTemplateSettingsInfo(t);
          Console.WriteLine("[{0}] [{1}] [{2}]", t.Name,
                      t.UnitMeasurement.Name, info.Unit.Name);
          foreach (var newUnit in t.UnitMeasurement.Units)
          {
            IUnitConverter converter = canonicalUnitCatalog.
                                 GetConverter(info.Unit, newUnit);
            double dOrigVal = 123.45;
            double dNewVal = converter.Convert(dOrigVal);
            Console.WriteLine(" [{0}] {1} converts to [{2}]{3}",
            dOrigVal,info.Unit.Symbol,dNewVal,newUnit.Symbol);
            if (t.IsWritable)
            {
              info.Unit = newUnit;
            }
          }
        }
      }
    }
  }
}
```

6 Develop Studio Manager plug-ins with the Studio Manager API

In this chapter

Overview

This chapter describes Studio Manager plug-ins, the Studio Manager sample plug-in, how to create a Studio Manager plug-in with Ocean for Studio, and how to access a Studio repository with a Studio Manager plug-in.

About Studio Manager plug-ins

All applications developed for interactive workflows must be developed as Studio Manager plug-ins.

These requirements apply to Studio Manager plug-ins:

- A Studio Manager plug-in that uses Ocean for Studio does not require a Runtime license.
- Client applications do not require any distribution of Schlumberger components or software.
- Runtime for Studio is automatically available with the Studio Manager installation.

Note: Batch-oriented applications that use Runtime for Studio must include the Runtime for Studio setup **`Slb.Ocean.Studio.Data.Petrel.Setup.dll`** with their package. Runtime for Studio is installed separately and is not redistributed with the application.

The Studio Manager sample plug-in

This section describes the Studio Manager sample plug-in and how to install the Studio Manager sample plug-in.

Overview

Deploying a Studio Manager plug-in requires:

- The plug-in assembly. Place the assembly in the Studio Manager plug-in installation folder.
- A **`.plugininfo`** file for the plug-in. Place this file in the Studio Manager plug-in installation folder, also.
- The Plug-in Manager (**`StudioPluginManager.exe`**). The Plug-in manager is used to register the plug-in. When Studio Manager starts, it automatically detects the registered plug-in and loads it into the environment.

In some instances, plug-ins require more advanced installation.

Studio Manager sample plug-in

Ocean for Studio includes a sample plug-in. Ocean for Studio also includes the complete code, a Visual Studio project, a compiled version of the sample plug-in and all of its required data files.

Installing the sample plug-in

Follow these steps to install the sample plug-in:

1. Locate the **PlugInSDKSampleInstall.zip** file that is included with Ocean for Studio.

   This file contains the assembly **Slb.Studio.Manager.Samples.SDK.dll**, the **SdkSample.plugininfo** file and a **Data** folder that contains a number of **.csv** and **.bmp** files that are used by the sample plug-in.
2. Locate the Studio Manager installation. By default, Studio Manager is installed in this location:

   C:\Program Files\Schlumberger\Studio Manager 2020
3. The Studio Manager installation includes a **Plugins** directory. Create an **SDK** directory in the **Plugins** directory:

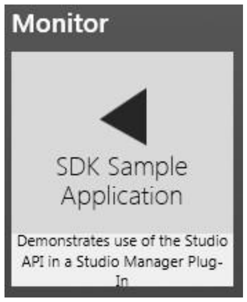
   C:\Program Files\Schlumberger\Studio Manager 2020\Plugins\SDK

Note: You need Administrative privileges if Studio Manager is installed in the default location.

4. Ensure that Studio Manager is **not** running.
5. Unzip **PlugInSDKSampleInstall.zip** into the SDK directory. You may need to unblock the **PlugInSDKSampleInstall.zip** file:
   – Right-click the **PlugInSDKSampleInstall.zip** file and click **Properties**.
   – Click **Unblock** at the bottom of the **Properties** dialog box.
   (You will not see this button if the **.zip** file is already unblocked, or if it was never blocked by Windows security.)
6. Open a command prompt.
7. Navigate to the Studio Manager installation directory.
8. Execute the Plug-in Manager to register Ocean for Studio plug-in:

   ```
   StudioPluginManager.exe register SDK
   ```
9. Start Studio Manager.

   You will see the SDK Sample Application in the App Gallery:


Monitor
SDK Sample Application
Demonstrates use of the Studio API in a Studio Manager Plug-In

Note: The sample plug-in will be enabled if you are connected to a project.

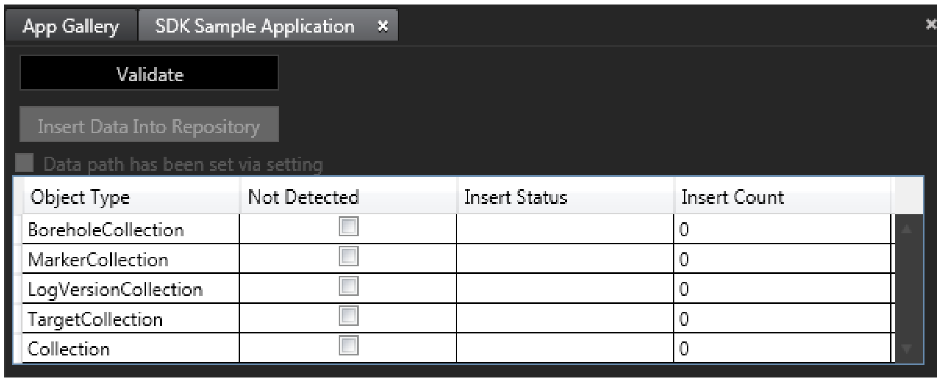
10. Connect to an empty repository in Studio Manager.

You may have to create an empty repository using the **Repositories** App.

Note: The storage coordinate system of the repository should be compatible with the **`Sphere_Vertical_Perspective`** coordinate system.

11. Click the **SDK Sample Application** icon to open the sample application.

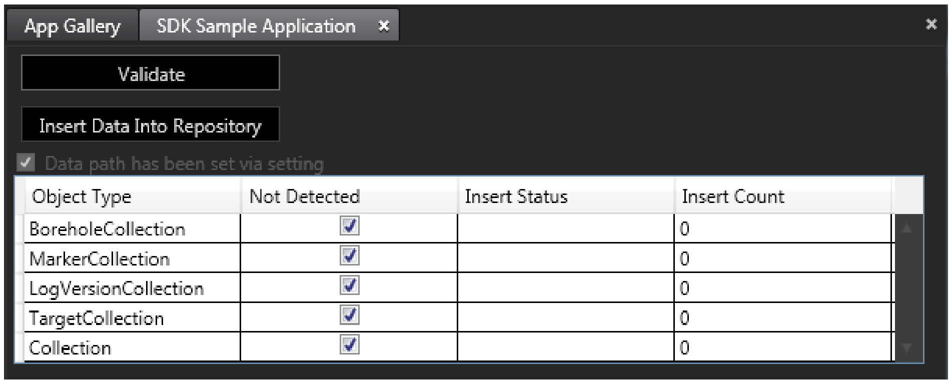
    The sample application opens in Studio Manager:


App Gallery
SDK Sample Application
Validate
Insert Data Into Repository
Data path has been set via setting

Object Type	Not Detected	Insert Status	Insert Count
BoreholeCollection	☐		0
MarkerCollection	☐		0
LogVersionCollection	☐		0
TargetCollection	☐		0
Collection	☐		0

12. Click **Validate**.

    The sample application validates the empty repository, **Insert Data into Repository** becomes enabled if no data of the specified object types is detected in the repository:


App Gallery
SDK Sample Application
Validate
Insert Data Into Repository
Data path has been set via setting

Object Type	Not Detected	Insert Status	Insert Count
BoreholeCollection	✓		0
MarkerCollection	✓		0
LogVersionCollection	✓		0
TargetCollection	✓		0
Collection	✓		0

13. Click **Insert Data into Repository**.

    Messages appear as the data is inserted into the repository.

Note: If you are using an SQL server and you do not have sufficient privileges to write Blob data through the file stream, you may not be able to successfully insert the data to the repository. You may receive error messages related to incorrect values or other exceptions.

14. Use other apps such as the **Data Table** app in Studio Manager to work with the data loaded into the repository.

Creating a Studio Manager plug-in

This section describes how to use Ocean for Studio to create a Studio Manager plug-in.

Prerequisites

Creating a Studio Manager plug-in requires:

- Visual Studio 2015
- Microsoft .NET 4.7.2
- Studio Manager 2020
- Setting the `StudioManager2020Home` environment variable

Note: The Studio Manager installer automatically sets the system environment variable `StudioManager2020Home` to the Studio Manager installation directory.

Set up the Visual Studio project

Follow these steps to set up your Visual Studio project:

1. Create a new class library using the .NET Framework 4.7.2.
2. Add these .NET assembly references:
   - **WindowsBase**
   - **PresentationCore**
   - **System.ComponentModel.Composition**
   - **System.Xaml**
3. Add these basic Studio assembly references:
   - **Slb.Studio.Manager.dll**
   - **Slb.Studio.Manager.Core.Common.dll**
   - **Slb.Studio.Manager.Core.Gui.dll**
   - **Slb.Studio.Manager.Core.Logic.dll**

Add an App-derived class

Follow these steps to add an `App`-derived class:

1. Add a new class to your project.

   Add the `[Export(typeof(App))]` attribute to your class to allow Studio Manager to locate the plug-in dynamically. The class does not need to be public; it can be given internal access.
2. Add using statements for these namespaces:

- **System.ComponentModel.Composition**
- **Slb.Studio.Manager.Apps**
- **Slb.Studio.Manager.Presentation**
- **Slb.Studio.Manager.Components**

3. Derive your class from the `SimpleApp` class. In your constructor, call the `SimpleApp` constructor with these values:
   - `Id` provides your plug-in's UniqueId.
   - `Category` provides the AppCategory in the App Gallery where this application should appear. `WellKnownAppCategories` supplies several default categories: Monitor, Organize, Correct, Load, Administer, and Browse.
   - A `CanRunInfo`-derived object that Studio Manager will use to determine whether your plug-in should be enabled or disabled in the App Gallery. The `DatabaseContextCanRunInfo` class is very useful here, or you can derive your own.

Note: The `DatabaseContextCanRunInfo` class locates in assembly reference **Slb.Studio.Manager.PetrelCore.Local.dll**.

   - Supply a list of the `Component`-derived objects that make up the user interface of the application. Each `Component`-derived object has a dockable view. Most applications will have only a single component. The `BasicComponent` class provides a simple default implementation.
   - Each component must have a UniqueId.
   - `NameInfo` can be supplied by the `BasicNameInfo` class, a simple implementation.
   - `VectorImageInfo` can be supplied by `BasicVectorImageInfo`, a simple implementation. Images must be vector graphics.

Add a ViewModelBase-derived class

The `ViewModelBase` provides basic implementations of the **ViewModel** of the **MVVM** (Model-View-ViewModel) design pattern and supports notification of dependencies. Your implementation can use the features of the `ViewModelBase` to provide a **ViewModel** for your application.

1. Add a new class to your project
2. Add using statements for these namespaces:
   - **System.Windows.Input**
   - **Slb.Studio.Manager**
   - **Slb.Studio.Manager.Components**
   - **Slb.Studio.Manager.Core.Gui.ViewModel**
3. Derive your class from `ViewModelBase`.
4. Override the protected virtual `Dispose (bool disposing)` method to perform cleanup activities as needed.

5. Use **`ICommand`** delegate properties to handle UI events from your view:
   - Create an **`ICommand`** property for the UI command handler
   - Instantiate a new **`DelegateCommand`** to route the command request to the handling method

Add an XAML UserControl-derived view

This view provides WPF GUI controls for your application user interface.

Add a ViewFactory-derived class

The **`ViewFactory`** generates the UI contents for a plug-in, it is the factory method that creates the **View** of the **MVVM** (Model-View-ViewModel) design pattern.

Follow these steps to add a **`ViewFactory`**-derived class:

1. Add a new class to your project.

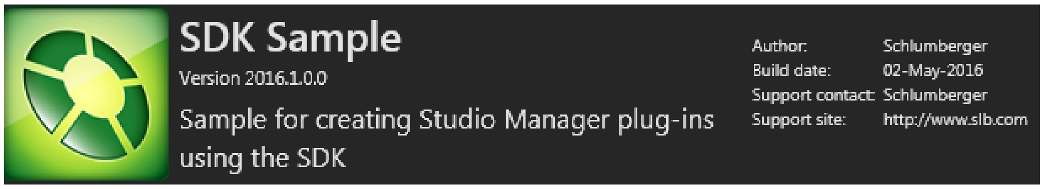
   Add the **`[Export(typeof(ViewFactory))]`** attribute to your class to allow Studio Manager to locate the **`ViewFactorty`** dynamically. The class does not need to be public; it can be given internal access.
2. Add using statements for these namespaces:
   - **System.ComponentModel.Composition**
   - **Slb.Studio.Manager.Views**
   - **Slb.Studio.Manager.Components**
3. Derive your class from the **`SimpleViewFactory`** class. In your constructor, call the **`SimpleViewFactory`** constructor with these values:
   - **`Id`** provides your view's UniqueId.
   - **`Factory`** provides the **`System.Func`** used to create the view of base type **`UIElement`**, it should return the XAML **`UserControl`**-derived view created in the step above.

Create a .plugininfo file

Plug-ins must be registered with Studio Manager to be loaded. Studio Manager locates and loads plug-ins dynamically at run time. Each plug-in must have the **`[Export(typeof(App))]`** attribute to be located by Studio Manager.

Use the **StudioPluginManager.exe** tool to register the plug-in. As a command-line parameter, you must supply the name of your plug-in's subdirectory that resides within the Studio Manager plug-in directory.

When you start Studio Manager, it locates the **.plugininfo** file in that subdirectory and reads the file. The information in the file is included in the Studio Manager plug-in catalog and is displayed in the **Plug-in manager** tab of the **Settings** dialog box:



SDK Sample
Version 2016.1.0.0
Sample for creating Studio Manager plug-ins using the SDK

Author:	Schlumberger
Build date:	02-May-2016
Support contact:	Schlumberger
Support site:	http://www.slb.com

A **.plugininfo** file should match this template.

```
{
   "Id":"Slb.Studio.Manager.Samples.SdkSample",
   "PluginDependencies":["Slb.Studio.Manager.Petrel"],
   "Author":"Schlumberger",
   "BuildDate":"02-May-2016",
   "Description":"Sample for creating Studio Manager plug-ins 
using the SDK",
   "ImageName":"",
   "Name":"SDK Sample",
   "SupportContact":"Schlumberger",
   "SupportSite":"http://www.slb.com",
   "Version":"2016.1.0.0"
}
```

The **`Id`** value should be unique.

`PluginDependencies` indicates whether this plug-in depends on any other plug-ins. A plug-in that uses Runtime for Studio should include **Slb.Studio.Manager.Petrel** in its dependency list.

`Author`, **`BuildDate`**, **`Description`**, **`Name`**, **`SupportContact`**, and **`Version`** are all displayed in the **Plug-in manager** window. ImageName is the name of an image file in the plug-in's directory that is displayed as an icon in the **Plug-in manager** window. If no image is specified, a default image from Studio Manager is used.

Use Ocean for Studio to access a repository

This section describes how to use Ocean for Studio to access a repository in a Studio Manager plug-in.

Add basic Studio assembly references

Add these Studio assembly references:

- **Slb.Ocean.Core.dll**
- **Slb.Ocean.Coordinates.dll**
- **Slb.Ocean.Studio.Data.Petrel.dll**
- **Slb.Studio.Manager.PetrelCore.Remote.dll**

Modify the code to use the current repository

Typically done in the **`ViewModel`** or in a class called from the **`ViewModel`**, this is where you add the functionality to access the repository.

1. Add using statements for these namespaces:
   - **Slb.Ocean.Studio.Data.Petrel**
   - **Slb.Studio.Manager.PetrelCore.Remote**
2. Use the **`RemoteSystem.Services.SdkContextService`** class to test whether a repository can be opened by calling **`CanOpenCurrentProject`**.
3. Before connecting to the repositiory for data operations, synchronize the Ocean coordinates catalog by calling **`SynchronizeCoordinateCatalog`**.
4. Open the repository and get a repository object by using **`OpenCurrentProject`**.

   A CRS with a valid transform to the repository's CRS is required to open the repository. You are responsible for properly disposing of the repository when it is no longer needed.
5. Use the **`Repository`** object to access data from the Studio repository.

   Refer to the Studio SDK API documentation for more information.

```
ISdkContextService contextService =
    RemoteSystem.Services.SdkContextService;
if (contextService.CanOpenCurrentProject)
{
  contextService.SynchronizeCoordinateCatalog(false);
  using (Repository repository  =
      contextService.OpenCurrentProject(m_Crs))
  {
    var queryBC = from bc in
        repository.Queryables.Well.AllBoreholeCollections
        select bc;
  }
}
```

Note: Include clean-up code in your **`ViewModel`**'s **`Dispose`** override if you retain a reference to the **`Repository`** object.

Handle events when Studio Manager changes the current repository

Studio Manager publishes events for certain condition changes. Subscribe to the events if your application can be affected by them, and change state appropriately. For example, handle an event for a change in connection to the active database (the active login session to the database containing one or more repositories) or the active repository.

1. Create an event handler **`ActiveConnectionHandler`** for the active database connection changed event and an event handler **`ActiveRepositoryHandler`** for the active repository connection changed event. The method signature is:

```
void handler (object sender, EventArgs e);
```

2. Add a using statement for this assembly reference:
   – **Slb.Studio.Manager.PetrelCore.Remote**
3. Subscribe to the events with the event handlers created in step 1:

```
RemoteSystem.Services.DatabaseConnectionService.ActiveConnecti
onChanged += ActiveConnectionHandler;
RemoteSystem.Services.DatabaseConnectionService.ActiveReposito
ryChanged += ActiveRepositoryHandler;
```

Unsubscribe from the events when you dispose of your **`Repository`** object.

Remember that you must raise your own property-changed notifications for the GUI to be updated with the new information. **`ViewModelBase`** supports property-changed notifications through use of the **`RaisePropertyChanged`** method:

```
RaisePropertyChanged("RepositoryName");
```

Enable and disable the application dynamically

Studio Manager uses the **`DatabaseContextCanRunInfo`** class to automatically respond to changes in active database connection, active repository, and user privilege. Studio Manager hides or displays your plug-in dynamically in response to changes in these conditions based on the requirements you supply. Your plug-in is visible in the App Gallery if it meets certain specified restrictions. Studio Manager hides your plug-in if the minimum conditions are not met after your plug-in starts. In this case, a message is displayed on the screen that describes the required conditions for your plug-in.

If the **`DatabaseContextCanRunInfo`** class does not provide sufficient control, you may derive your own **`CanRunInfo`** class and/or handle the various notification events from the **`DatabaseConnectionService`** yourself.

Keep the application responsive

You can keep the application responsive by moving background processing work into a different thread. Connecting to a repository or reading from or writing to the repository can be slower tasks that can noticeably impact the responsiveness of Studio Manager.

Additionally, status updates and progress messages are displayed by the GUI only when the system has a chance to react to your updates. To show messages (including writing to the Studio Manager message log) during your operations, you must start the operations on a background thread.

1. Add a using statement for:

```
using System.Threading.Tasks;
```

2. In the delegate's handler, start a new thread:

```
Task.Factory.StartNew(action, this);
```

3. From within the thread, remember to use the **`Dispatcher`** to update user interface properties:

```
Dispatcher.BeginInvoke(new Action<ViewModel>(sender =>
      { Mouse.OverrideCursor = Cursors.Wait; }), this);
```

Note: You must access the **`Repository`** object from the thread on that you create it on. That is, if you activate a background thread to do work on the repository, you must use **`OpenCurrentProject`** on that thread to obtain a **`Repository`** object that is valid on that thread. Failure to do so will result in an exception.

7 Studio Manager UI and UX guidelines

In this chapter

Overview

This chapter describes the standards and best practices for designing Studio Manager plug-ins. These guidelines ensure the consistency of the user interface and improve the user experience of Studio Manager and the plug-ins.

Although this information is primarily intended to serve as a reference for developers, it can be used by anyone involved in the user interface and user experience of Studio Manager.

All Studio Manager developers should read these guidelines and refer to them during the development process as a reference to create consistent Studio Manager interfaces.

Compliance

Throughout this chapter these keys indicate the required level of compliance:

- [Rule]
- [Guideline]
- [Option]

[Rule]

Rules are *absolute requirements* for compliance. Many of the rules are enforced through the API.

[Guideline]

Guidelines are strong recommendations. Guidelines are considered important for consistency and should be followed as closely as possible.

[Option]

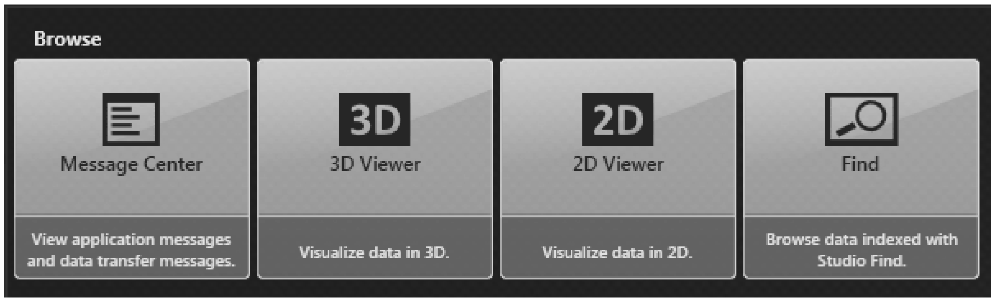
Optional elements, if provided, support the UI.

Anatomy of a Studio Manager plug-in

This section describes the standard components of a Studio Manager plug-in and shows examples of how they usually look.

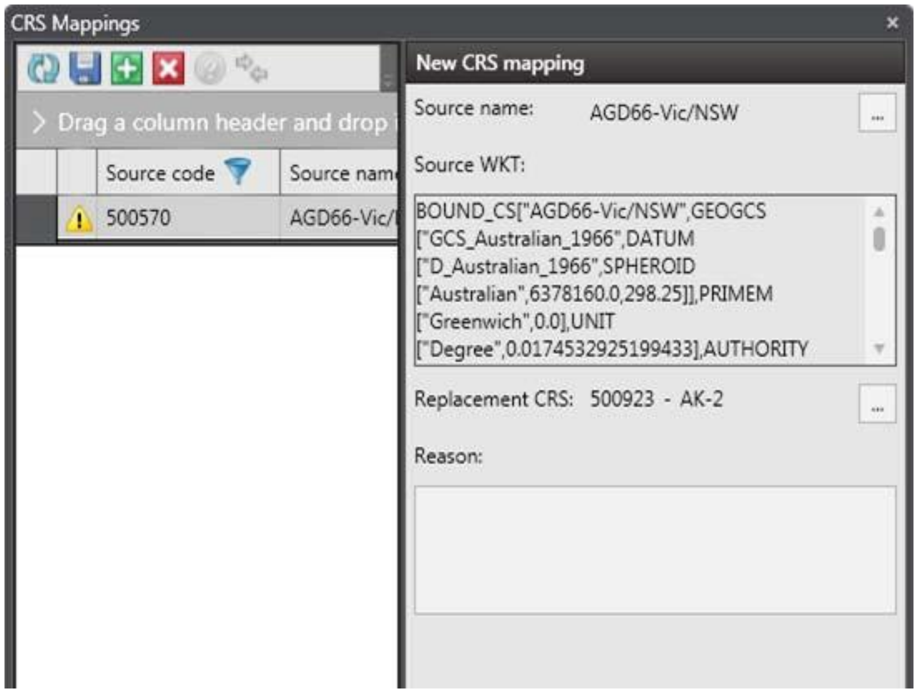
App Gallery titles

Titles appear in the App Gallery with the application icon. Applications are divided into distinct categories. Users click the App tile to launch an application. This figure shows typical applications in the App Gallery.


Browse
Message Center
View application messages and data transfer messages.
3D Viewer
Visualize data in 3D.
2D Viewer
Visualize data in 2D.
Find
Browse data indexed with Studio Find.

App views

The app view is the base of the UI. The view contains the content and controls. Whenever possible, keep all UI elements within the base app view. Views can be docked and undocked from the main application window. If you open additional windows or dialog boxes from your app view that are not clearly related to your application, it could be confusing to users. You can create as many views as you need within a plug-in to support your user workflows. This figure shows a typical app view.



Source name: AGD66-Vic/NSW

Source WKT:

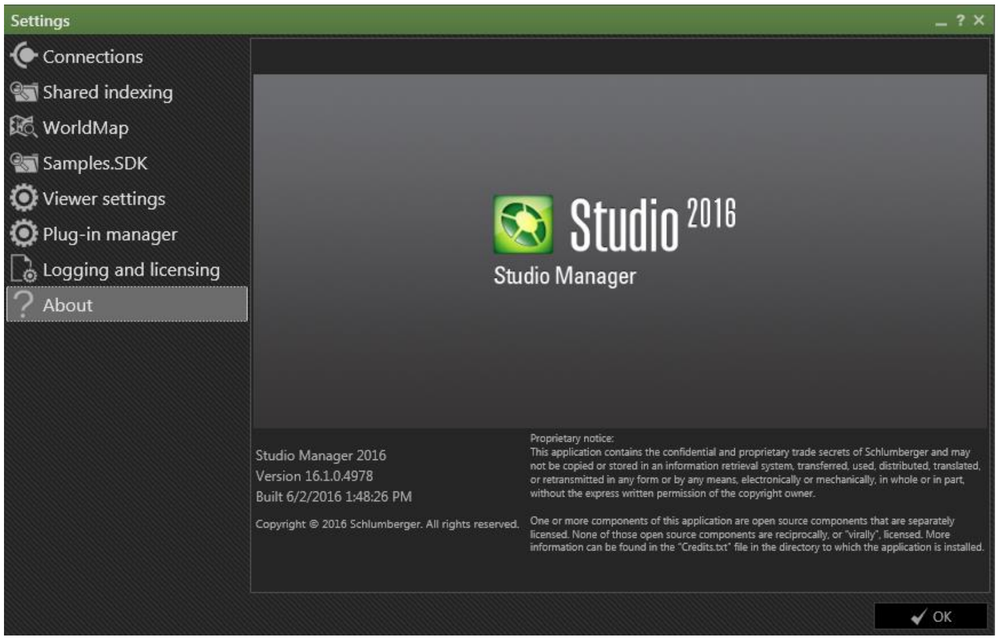
```
BOUND_CS["AGD66-Vic/NSW",GEOGCS
["GCS_Australian_1966",DATUM
["D_Australian_1966",SPHEROID
["Australian",6378160.0,298.25]],PRIMEM
["Greenwich",0.0],UNIT
["Degree",0.0174532925199433],AUTHORITY
```

Replacement CRS: 500923 - AK-2

Reason:

Settings dialog box

If your application requires special settings, the Studio Manager API provides a service for adding elements to the **Settings** dialog box. This figure shows the **Settings** dialog box.



Studio Manager 2016
Version 16.1.0.4978
Built 6/2/2016 1:48:26 PM

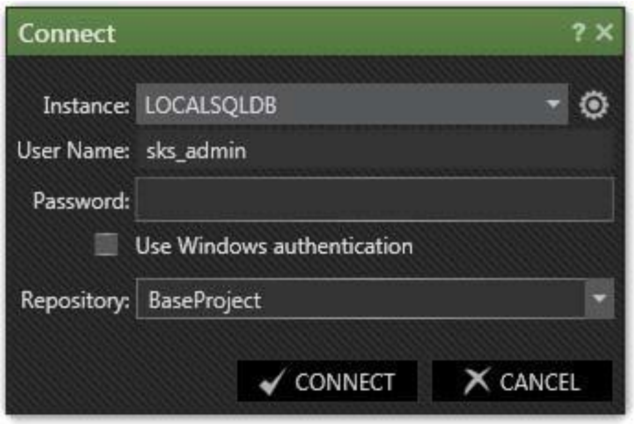
Copyright © 2016 Schlumberger. All rights reserved.

Proprietary notice:
This application 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, electronically or mechanically, in whole or in part, without the express written permission of the copyright owner.

One or more components of this application are open source components that are separately licensed. None of those open source components are reciprocally, or "virally", licensed. More information can be found in the "Credits.txt" file in the directory to which the application is installed.

Dialog boxes

If your application requires additional dialog boxes, the Studio Manager API provides services that generate basic dialog boxes. This figure shows a dialog box created with the Studio Manager API services.



Instance: LOCALSQLDB
User Name: sks_admin
Password:
Use Windows authentication
Repository: BaseProject

Studio Manager plug-in guidelines

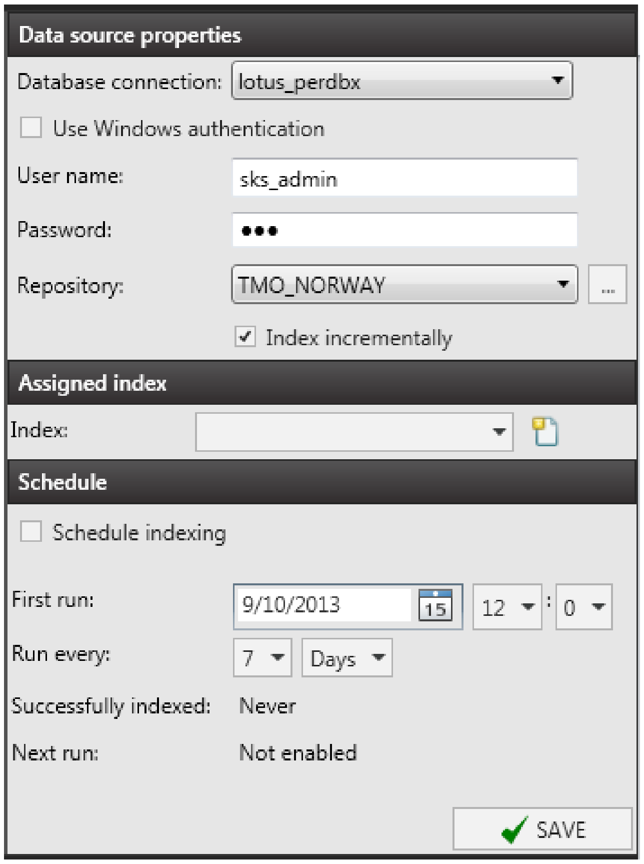
This section describes the guidelines for designing Studio Manager plug-ins.

View layout design

The layout is how your UI elements are organized within a view. This section provides a set of patterns and practices for setting margins, headers, padding, and similar elements. These patterns and practices are intended to speed your view design, maintain unity throughout Studio Manager, and help you to understand interactions across the application. Important layout design considerations include:

- Avoid visual clutter.
- Ensure that each element is truly necessary to the workflow you are presenting to users.
- Align and group all controls as they fit together within a view.
- Keep multiple controls of the same type consistent in size as well as placement.

This figure shows how UI elements are grouped into related sections:



Data source properties

Database connection: lotus_perdbx

☐ Use Windows authentication

User name: sks_admin

Password: •••

Repository: TMO_NORWAY

☑ Index incrementally

Assigned index

Index:

Schedule

☐ Schedule indexing

First run: 9/10/2013 12 : 0

Run every: 7 Days

Successfully indexed: Never

Next run: Not enabled

Grouping

Related UI elements must be grouped together. Elements can be grouped together by using spacing or separators. In the previous example, headers are used to divide three separate components of an entity within a workflow. Spacing is also an effective tool to provide separation and should not be avoided.

Alignment

UI elements should be aligned horizontally on the left. In the previous example, the labels for each section of this UI are left-aligned with each other. The text boxes, drop-down lists, and date controls are also left-aligned with each another in a separate column. The fewer alignment lines in your view, the easier groupings are to read.

Resolution

A view must support a minimum resolution of 1024x768 pixels. Views are contained within the Studio Manager application and are sized appropriately to fit within it. Although it may not be avoidable at some application window sizes, scrolling should be avoided whenever possible.

Resizing

All views must be resizable because the views can be undocked and resized as separate windows.

A default window size can be set for undocked views and should be set to encourage users to use your view at this size. However, this is not strictly enforced.

Some dialog boxes do not need to be resizable. If the dialog box does not benefit from being resized, disable resizing.

Testing

Views must be tested at the minimum resolution of 1024x768 pixels.

Font

Studio Manager uses the Segoe UI font, which is the MS Windows system typeface. The standard font size used is 9 point.

Decoration

You must not abuse bold, italics, underlining, strike through, or any other font decoration or attribute. These tools should be used sparingly and only when they are appropriate to emphasize important text within a UI element. You do not need to make headers or other specialized element text bold. Emphasis of text is taken care of by the UI element itself.

Color and contrast

You should use a dark font color on a light background and a light font color on a dark background. In this section on color, specific values are provided to assist you.

Color

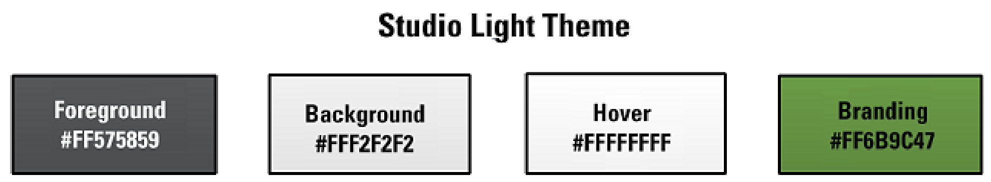
This section describes the standard colors and themes used for Studio Manager.

Studio Manager Palette

The primary color of Studio Manager is gray. Color is used sparingly and usually only to highlight information or for branding. The Studio branding is green. There are also two color themes used for Studio Manager, dark and light.

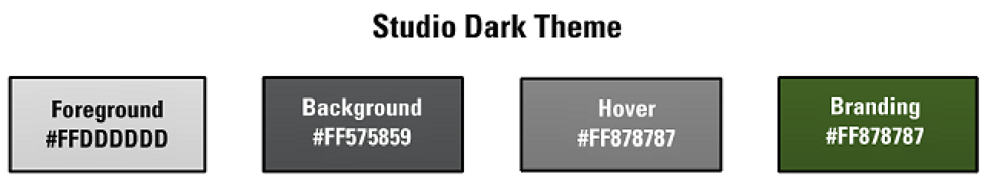
Studio Light Theme

The light theme is used within views. This provides a more familiar color environment for users with light backgrounds and dark text.



Studio Dark Theme

The dark theme is used within dialogs and the App Gallery. This creates a contrast that makes important messages are more visible.



Icons

Studio Manager icons are in two formats: raster and vector. Raster icons are used for toolbars and other fixed-size menus. Vector icons are more scalable and are used within controls such as buttons or app catalog tiles.

Raster icons

All Studio Manager raster icons must be in PNG format. Raster icons must not be scaled. You must provide an icon for each size of that image to be used in the app.

Naming conventions

Studio Manager icon files use this naming convention:

`<icon_name>``_`*`<size>`*`.png`

<icon_name> is a name that describes the function of the icon, in lowercase with each word in the name separated with an underscore (_).

`<size>` is the size of the icon in pixels. Common sizes are 16, 24, or 48 pixels square.

For example, if you create a data loading icon, the file name could look like this example:

`data_loading_16.png`

Icon size

The most common icon size in Studio Manager is 16 pixels. This is the icon size used for toolbars, tree view items, and most other controls that need an icon to mark identity.

Modifiers

Modifiers are small repeated portions of icons used to communicate a usage. These can include such things as a plus sign in the corner of an icon used to add items or an x in the corner of an icon to denote deletion. Studio Manager provides a list of icon modifiers and templates to reuse to maintain consistency across apps.

Guidelines

Icons should follow templates to match the style of other icons within the application, and icons should be simple. You should pay special attention to the smaller icons.

Icons must not be resized photos. Most small images become hard to interpret unless they are simplified.

Rounded tile or candy style icons, like those found in iOS, should not be used. In Studio Manager, most icons are flat in color and style.

Use an icon to convey a single concept. You must not reuse the same icon to convey multiple meanings. Using a single icon to denote deleting an item across multiple views in your app is fine. Using the same icon for both delete and cancel is not.

Vector Icons

Studio Manager also uses vector-based icons in controls. The top menu, app catalog icons, and button icons are all vector-based icons. Unlike raster icons, vector icons are built to be scalable to any size. All Studio Manager icons must be in path markup syntax. (See *http://msdn.microsoft.com/en-us/library/ms752293.aspx for details*.)

Guidelines

In addition to the rules for raster icons, vector icons have other special considerations:

- Vector icons must be only one color. This color is detailed for the control listing that this icon will be associated with in the UI controls geometry and color appendix.
- Vector icons must be a single path markup string. This does not mean the icon cannot be composed of multiple figures within the markup, but a single geometry must be used for each raster icon.

Tone

The tone of Studio Manager is simplicity. You should aim to reduce any unnecessary wording, use simple sentences, and use consistent wordings throughout your app.

Language

The default language of Studio Manager is American English (English-US) in both the interfaces and the user guide. You must not use British English (English-UK) in your applications.

The Schlumberger standard is the Merriam-Webster’s dictionary.

Guidelines

Be clear.

A phrase like: “You cannot use this type of input” is ambiguous. Clarify what that type of input was and possibly why it couldn’t be used, as this example does:

The `SeismicCube` ‘Top’ could not be deleted. It is in use by seismic interpretation ‘Top Interp.’

Be concise.

You should use simple sentence structure. Keep information brief and topical. Help can be suggested for longer and more detailed explanations.

Be consistent.

Terms like well or borehole may be interchangeable. This does not mean you should ever interchange them. You must pick a term when referring to a concept and stick to that term. You can check other applications in Studio Manager. You should also try to keep your language consistent with the rest of the application.

Speak the language of the users.

Phrases like “null reference exception” may be meaningful to those supporting your application. However, these messages can confuse users. You should present the users with a more readable message like: “An error occurred processing workflow ‘Workflow 1.’ Please contact your system administrator if you need assistance to resolve this problem. A more detailed message has been added to this log.”

Do not be patronizing.

Questions like: “Are you sure?”, or “Do you really want to exit?” can sound annoying or even hostile to a frustrated person. You should remove adjectives or qualifiers from your language. This will leave the language neutral: “Do you want to exit?”

Use an ellipsis mark.

Use an ellipsis mark (...) to indicate truncated text or commands that require additional input.

Use sentence case capitalization.

Here are directions for implementing sentence case capitalization:

- Capitalize the first word.
- Use lowercase for everything else except proper nouns.

Controls

This section describes the requirements for controls.

Label

Labels should apply these rules:

- Labels must follow the tone guidelines in “Tone” on page 7-9.

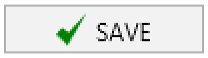
- Labels should be left-aligned.

Button

Buttons should apply these rules:

- Buttons should have a label. Buttons in toolbars and used in direct conjunction with a single control may not.
- Button labels must be in all capital letters.
- Button labels must follow all other rules in the language guidelines in "Tone" on page 7-9 except for capitalization.
- Buttons must have a tooltip.
- Buttons should have a verb as their label.
- Buttons can have an icon associated with them. That icon must be a vector icon.

This example shows a button:

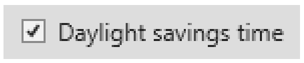


Check boxes

Check boxes should apply these rules:

- Check boxes must have an associated label. Follow the language guidelines in "Tone" on page 7-9 to keep your labels clear and consistent.
- Check boxes must enable or disable an option or activate state. If your control needs to execute an action it may be better to use a button.
- Check boxes must be ordered by importance and frequency of use.
- Multiple check boxes in a group should be aligned vertically.
- If users are selecting between are mutually exclusive options, you must use radio buttons.

This example shows a check box:

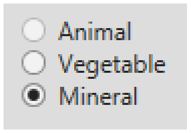


Radio Buttons

Radio buttons are used to select between multiple mutually exclusive options or states. If your control needs to execute an action it may be better to use a group of toggle buttons. Radio buttons should apply these rules:

- Radio buttons must have an associated label. Follow the language guidelines in "Tone" on page 7-9 to keep your labels clear and consistent.
- Radio buttons must be ordered by importance and frequency of use.
- Multiple radio buttons in a group should be aligned vertically.
- If the options users are selecting between are not mutually exclusive, you must use check boxes.

This example shows radio buttons:

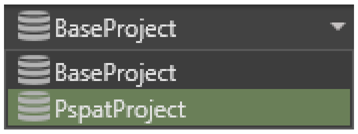


Combo boxes

Combo boxes should be expected to contain three or more items. If you have fewer items, use radio buttons instead. Combo boxes should apply these rules:

- Combo boxes must be for single selection only. They can only display one selected item at a time once collapsed. Users will find having to open the combo box just to see their selections inconvenient.
- Combo boxes should have content organized on some sort of logical order. For larger sets, this may be something like alphabetical order. If there is a hierarchy to the data or data that is commonly selected that may also be good to organize at the top of the list of available items.
- Combo boxes should have a default selection. This may be something like the last option used or the most used item. Users will appreciate the convenience of not having to select an item from this control every time it is displayed.

This example shows a combo box:



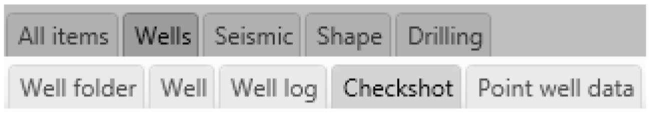
Tabs

Tabs should have names that are nouns. Verbs denote an action and actions are more commonly associated with buttons and commands. Tabs should apply these rules:

- Tabs must be used when multiple pages of data are kept within the same view.
- Tabs are used only if there are two or more pages. A single page of data can be labeled or given a header.
- Tabs must be organized on a single row if they pertain to the same group of pages.
- Tabs can be nested, but only if there is a strong but related separation between the pages they separate. In many cases this separation can be better handled by filtering the pages you want to view via another control like a combo box.
- Tabs on the same level should be dependent on one another.
- Tabs must be single selection. You can see only one page at a time within your view.
- Tabs must not execute actions upon selection. They are used only to change pages within a view.
- Tabs can be disabled. If a tab is disabled, it must have a tool tip that gives a clear explanation as to why the tab is disabled. If this explanation cannot be told in only

a few words, it may be better to enable the column and disable the view within the column. This will give you space to provide an adequate explanation about why this page is unavailable.

This example shows a tab:

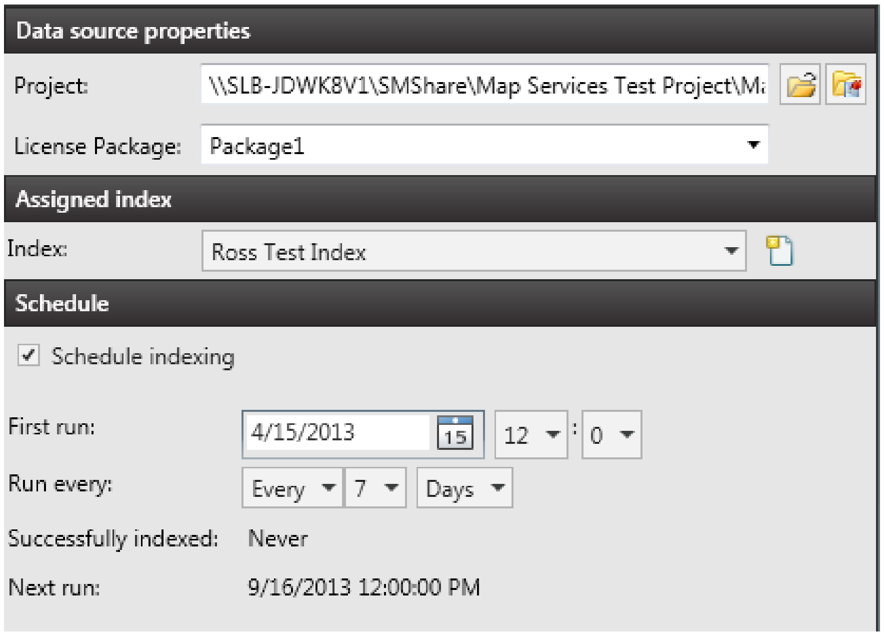


Group box

Group boxes in Studio Manager are labeled sets of controls. Group boxes should apply these rules:

- Group boxes should separate related controls into understandable units.
- Group boxes must always have a label.
- Group boxes that contain only controls are indicated with a header.
- Group boxes must never be nested. Two headers back to back will look distracting to users. Try to combine the categories if possible. If you cannot, make those groupings into separate pages if possible using tabs or filters.
- Group boxes themselves must never be disabled. Disable individual controls within the group or consider replacing the content of the group with a label explaining why all the content is missing. If the group box should not be present at all in a given state, consider collapsing it and only displaying it when users need it.

This example shows a group box:



Data source properties

Project: \\SLB-JDWK8V1\SMShare\Map Services Test Project\M:

License Package: Package1

Assigned index

Index: Ross Test Index

Schedule

☑ Schedule indexing

First run: 4/15/2013 12 : 0

Run every: Every 7 Days

Successfully indexed: Never

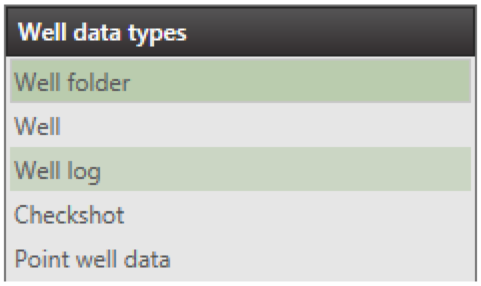
Next run: 9/16/2013 12:00:00 PM

List view

If you have a single column of data to display to users and this column has no hierarchy, you should use a list view. List views should apply these rules:

- List views can have single or multiple selections.
- List views must be sorted in a logical order. For shorter lists, this could be common selections first. For a longer list you should use alphabetical order as well.
- List views should display all the list items if possible. Not having to use a scroll bar will reduce the number of clicks required by users.
- List views must not execute any actions on selection.
- List views should allow selection only when they are more than a display.
- List views should have a title unless it is completely redundant.

This example shows a list view:

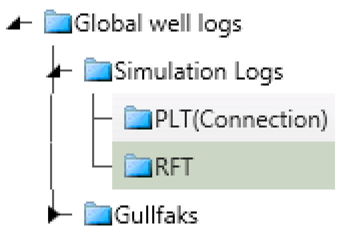


Tree view

If you have a column of data to display to users and this column has hierarchy, you should use a tree view. Tree views should apply these rules:

- A tree view should have limited depth. Keep structures simple so users do not have to perform too many clicks to browse the data within the control. If your data is deep or complex, you must make the tree view searchable or filterable.
- Tree views should have a title unless it is completely redundant.

This example shows a tree view:



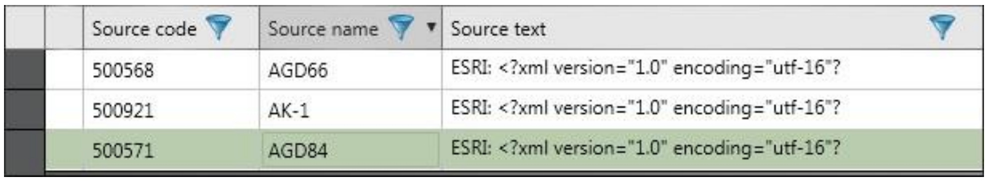
Grid view

Grid views are table views of data. Grid views should apply these rules:

- Grid views must sort data. Users must be able to sort their data in alphanumeric, number or some other logical order.
- Grid views must not scroll if there is enough space. Users will appreciate not having to jog the control back and forth while viewing the data.
- Grid views must not execute actions when users change the current selection.

- Cell content must not be wrapped. Use ellipsis marks when cell content is long.

This example shows a grid view:



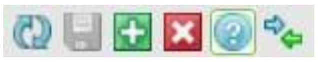
		Source code	Source name	Source text
		500568	AGD66	ESRI: <?xml version="1.0" encoding="utf-16"?
		500921	AK-1	ESRI: <?xml version="1.0" encoding="utf-16"?
		500571	AGD84	ESRI: <?xml version="1.0" encoding="utf-16"?

Toolbar

Toolbars should apply these rules:

- Toolbars should only have buttons without text. In some cases, such as a text toolbar, there may be combo boxes within the toolbar.
- Each button on a toolbar must have a tooltip.
- Toolbars must not be floating.
- Toolbars must be placed above the control they perform actions upon.

This example shows a toolbar:



Windows and dialog boxes

Windows and dialog boxes should apply these rules:

Titles

- Windows and dialog boxes must have a title.
- Titles must follow all other rules in the language guidelines in "Tone" on page 7-9.
- Titles should have simple names, if possible, consisting of a singular verb or noun.
- Windows and dialog box header icons must be produced as a vector.

Raster icons are not allowed for headers of dialog boxes.

Controls

- Windows and dialog box controls must be intuitive.

  Controls should be placed in the order of the workflow intended to be performed with the dialog box.
- Windows and dialog boxes encompassing multiple workflows should have a subdivided interface.

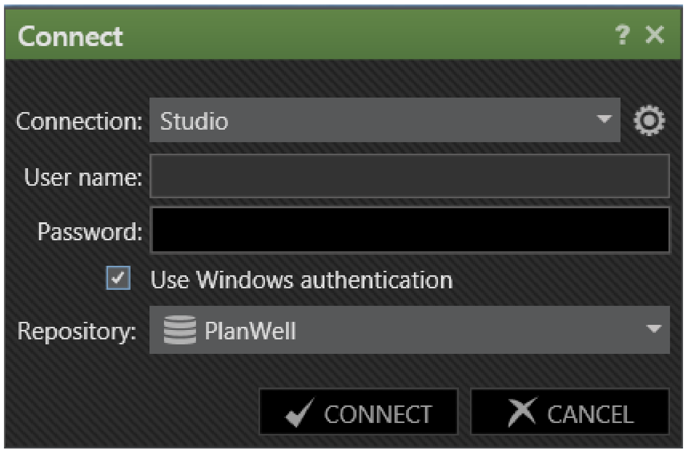
  Make use of sectional groups if possible.
- Controls irrelevant to the current workflow being performed in the dialog box must be hidden or disabled.

  If a section of controls is irrelevant to the current workflow, you must hide it.

  If a button or tool is irrelevant, but may become relevant, you must disable this control and leave it visible.

- Windows and dialog box controls should be laid out in a top-down, left-to-right order.

This example shows a typical dialog box:



A Ocean for Studio data validation

In this chapter

Data validation

This section lists the data validation that Studio SDK API performs. Some validation is performed explicitly, and some is provided implicitly by design of the API as noted.

Quality tags

These rules apply to all of the data types except Marker.

- **Check for Mandatory Quality Tags**: Ensures that all the quality tags marked mandatory by Studio Repository Admin are not null.
- **Check for Valid Quality Tags**: Ensures that all the quality tags meet the condition set by Studio Repository Admin.

General

This rule applies to all data types:

- The user must not write to non-writeable properties

Borehole

These internal formats are related to **`Wellhead`** and **`WorkingReferenceLevel.Offset`**:

- X_Y_TVDSS
- X_Y_TVDSS_MD
- DX_DY_TVDSS
- DX_DY_TVDSS_MD

These rules apply to the **`Borehole`** data type:

- **`Borehole`** must always contain:
  - **`Wellhead.X`** value
  - **`Wellhead.Y`** value
  - **`WorkingReferenceLevel.Offset`** value
  - **`Name`**
- The source CRS must:
  - Have a projection and a datum.
  - Be present in the Studio Database Server.
- If a borehole position is shown as infinity, not a number, **`Double.MaxValue`**, **`Double.MinValue`**, or greater than 1E35f, borehole is marked as invalid.
- If a borehole does not have a **`TrajectoryCollection`**, then the borehole can have only a single trajectory.
- If a borehole has a **`TrajectoryCollection`**, then it can have multiple trajectories. The trajectories can be either Surveys or Plans.

- Borehole can have a definitive survey out of all its trajectories of Survey type; it can have an active plan out of all its trajectories of Plan type.
- Borehole must have a working trajectory to be saved. The working trajectory is either the definitive survey or the active plan in this order of priority.

Note: Starting from Ocean for Studio 2015.1, a borehole will always have a **`TrajectoryCollection`** no matter it has a single trajectory or have multiple trajectories.

Trajectory

These rules apply to the **`Trajectory`** data type:

- **`Trajectory.ExplicitRecords`** must have two or more points.
- MD values of **`Trajectory.ExplicitRecords`** must be increasing with no duplicate values.
- Trajectory maximum depth must be equal to or must not be outside the given tolerance compared to the borehole's **`MDRange.Max`**.

Note: Starting from Ocean for Studio 2015.1, most properties and methods of **`Trajectory`** become Obsolete, except for access to **`TrajectoryCollection`** and **`OriginalCRS`**.

CheckShot

These rules apply to the **`CheckShot`** data type:

- **`CheckShot`** time values must always be present.
- This must be the only **`PointWellLog`** that exists for its combination of **`Borehole`** and **`PointWellLogVersion`** or **`CheckShotVersion`**.
- **`SrdReference`** must be set for all **`CheckShot`**.

WellLog

These rules apply to the **`WellLog`** data type:

- The depth array must be increasing and must not have duplicate depth values.
- **`SrdReference`** must be set for all **`WellLog`** that are **`OneWayTime`** or **`TwoWayTime`**.

MarkerCollection

These rules apply to the **`MarkerCollection`** data type:

- Zones:
  - A zone must have an absolute top horizon and an absolute bottom horizon.
  - Zones can have at most two horizons (absolute top and bottom) that have only one zone attached to them.
- Horizons:

  - Horizons must have zones attached to them.
  - A horizon with only one zone attached to it is an end point/edge for the stratigraph.
- **`SrdReference`** must be set for all **`MarkerCollection`**.

Marker

These rules apply to the **`Marker`** data type:

- The parent Borehole must be present.
- Marker depth must be present.
- The **`MDRange.Max`** of the parent Borehole must be deeper than the marker depth.

PointSet

This rule applies to the **`PointSet`** data type:

- **`SrdReference`** must be set for all **`PointSet`** created in **`ELEVATION_TIME`**.

PolylineSet

This rule applies to the **`PolylineSet`** data type:

- **`SrdReference`** must be set for all **`PolylineSet`** created in **`ELEVATION_TIME`**.

RegularHeightFieldSurface

These rules apply to the **`RegularHeightFieldSurface`** data type:

- **`SpatialLattice`** must be present.
- Lattice size must be valid.
- Lattice must be orthogonal.
- Name must always be present.
- Mandatory attributes (grid values, lattice information, CRS) must be present.
- CRS must be present in the Studio Database Server.
- **`SrdReference`** must be set for all **`RegularHeightFieldSurface`** created in **`ELEVATION_TIME`**.

Template

This rule applies to the **`Template`** data type:

- If template does not exist in the target, no validation is required.
- The template name must be globally unique as two templates cannot have the same name within a Studio repository.

HorizonInterpretation2D

These rules apply to the **`HorizonInterpretation2D`** data type:

- **`SeismicCollection`**:
  - Must be present.
  - Member type must be **`SeismicLine2DCollection`**.
- Mandatory attributes (horizon values, domain) must be present.
- Domain must be Depth or Time.
- Must have at least one valid (transferrable) line interpretation.
- Must have interpretation references to at least one 2D seismic line of geometry.
- **`SrdReference`** must be set for all **`HorizonInterpretation2D`** created in **`ELEVATION_TIME`**.

HorizonInterpretation3D

These rules apply to the **`HorizonInterpretation3D`** data type:

- **`SeismicCollection`**:
  - Must be present.
  - Member type must be **`SeismicCube`**.
  - Must not have any other **`HorizonInterpretation3D`**.
- Mandatory attributes (horizon values, domain) must be present.
- Domain must be Depth or Time.
- **`SrdReference`** must be set for all **`HorizonInterpretation3D`** created in **`ELEVATION_TIME`**.

HorizonInterpretation

These rules apply to the **`HorizonInterpretation`** data type:

- Domain must be Depth or Time.
- Must have a name.

FaultInterpretation

These rules apply to the **`FaultInterpretation`** data type:

- **`FaultInterpretation`** must always contain:
  - Name
  - Domain
  - Fault values
- Domain must be Depth or Time.
- **`SrdReference`** must be set for all **`FaultInterpretation`** created in **`ELEVATION_TIME`**.

Seismic3DCollection

These rules apply to the **`Seismic3DCollection`** data type:

- **`Seismic3DCollection`** must:
  - Have a name
  - Be the same size or smaller than the survey
  - Can be decimated relative to the survey
  - Contain only 3D seismic
- **`Seismic3DCollection`** origin must be close to a survey trace position.
- **`Seismic3DCollection`** inline and crossline axes must be almost parallel to the corresponding axis survey.

Note: **`Seismic3DCollection`** is a **`SeismicCollection`** with MemberType = typeof(**`SeismicCube`**). The uppermost **`Seismic3DCollection`** represents the survey. This is the **`Seismic3DCollection`** that has no parent, or a parent that is not another **`Seismic3DCollection`**.

Seismic2DCollection

These rules apply to the **`Seismic2DCollection`** data type:

- **`Seismic2DCollection`** must:
  - Have a name.
  - Contain only 2D seismic.
- CDP numbers count, **`ShotPoint`** numbers count, and **`LineNavigation`** count must be equal in **`Seismic2DCollection`**.

SeismicCube

These rules apply to the **`SeismicCube`** data type:

- A **`SeismicCube`** must:
  - Have an external file reference
  - Have an external file at the given path
  - Be ZGY
- Number of samples in internal index space must be greater than:
  - 1 in the Y axis
  - 1 in the X axis
- Always contain:

  - Origin
  - iUnitVector
  - jUnitVector
  - kUnitVector
  - Annotation at origin
  - Annotation increment
  - Storage type
  - Vertical domain of either Depth or Time
  - Template
  - Clipping range
- kUnitVector in the vertical domain direction must not be positive.
- **`SrdReference`** must be set for all **`SeismicCube`** created in **`ELEVATION_TIME`**.

SeismicLine2DCollection

These rules apply to the **`SeismicLine2DCollection`** data type:

- A **`SeismicLine2DCollection`** must:
  - Have an external file reference
  - Have an external file at the given path
  - Be SEG-Y
- Always contain:
  - Name
  - Line navigation with valid values
  - Shot point numbers with valid values
  - CDP numbers with valid values
  - Size J values (number of traces)
  - Size K values (number of samples per trace)
  - Vertical start
  - Vertical sampling interval as a positive value
  - Storage type
  - Vertical domain of either Depth or Time
  - Template
  - Clipping range
  - Correct size restrictions
  - At least one **`SeismicLine2D`** that is not virtually cropped
- **`SrdReference`** must be set for all **`SeismicLine2DCollection`** created in **`ELEVATION_TIME`**.

B Ambiguity in planned trajectory calculations

In this chapter

Overview

This appendix explains a potentially confusing aspect of the Studio SDK API. This appendix does not provide a complete explanation of well path computation.

Multiple solutions for planned trajectory calculations

When calculating a Planned Trajectory section using TVD, inclination, and DLS, the associated algorithms do not produce a unique solution when the initial inclination is a nonzero value. This occurs because of the way the minimum-curvature solution is determined.

When determining a minimum-curvature arc, TVD is calculated from:

- The initial and final inclinations
- The initial and final azimuth values
- The measured depth

These same values can be used to calculate DLS. The measured depth is not known, but TVD and DLS are supplied.

Use the minimum-curvature and DLS equations

The calculation can be worked backward to solve for the measured depth as a function of the final azimuth. This is done using both the minimum-curvature equation and the DLS equation, which yield two independent functions that take the final azimuth as their only parameter. Setting these two independent functions equal to each other enables us to solve either algebraically or numerically to obtain the final azimuth values that provide a solution.

This is the DLS formula as a function of final azimuth:

$$DLS = \cos^{-1}\Big[(\cos I_i \times \cos I_f) + (\sin I_i \times \sin I_f) \times \cos(A_f - A_i)\Big] \times \left(\frac{30}{MD}\right)$$

where:

A_i	Initial azimuth (given)
A_f	Final azimuth (unknown)
I_i	Initial inclination (given)
I_f	Final inclination (given)
DLS	Dogleg severity in degrees per 30 meters (given)
MD	Measured depth (unknown)

Assume that all angle measurements are represented in the range 0 to 360°.

The ambiguity

The function shows an ambiguity in the term that contains $\cos(A_f - A_i)$ because for any given value of (A_i) there are two values of (A_f) that yield the same result.

For example, with these values:

$A_i = 45$

$A_f = 55$

$I_i = 5$

$I_f = 10$

MD = 100

the result is that DLS = 0.5398°/30m.

This same DLS result can also be obtained by setting $A_f = 35$ instead. This is easily verified because cos(10°) = cos(-10°).

Thus, for any given set of input values for the initial azimuth, initial and final inclinations, TVD, and DLS, exactly two final azimuth values can be calculated by the calculation engine.

Provide an azimuth value that disambiguates the calculation

The Planned Trajectory API does not distinguish between the two final azimuth values that can be calculated by the calculation engine. It is the responsibility of the end user to provide an azimuth value to disambiguate the calculation.

This ambiguity may be encountered in two general workflows:

- A data-loading workflow
- An interactive workflow

The sample program for the **`PlannedTrajectory`** class demonstrates the use of both methods described for these workflows.

Data-loading workflow

In this workflow, the program populates a Studio repository with data from another source.

In this case, the complete Planned Trajectory data should be available. The software can use the unambiguous **`SetTvdInclinationDlsAzimuth`** method, which takes four parameters and can completely specify the details of the curved section in one step. (If the supplied values are not valid for the trajectory engine’s calculations, an error is returned.)

Interactive workflow

In this workflow, a user may construct a Planned Trajectory in pieces without knowing which value to supply for the azimuth.

In this case, the user should use the three-parameter **`SetTvdInclinationDls`** method. After calculating the trajectory, a return code will indicate that the user must select from one of two possible azimuth values. These values can be accessed via the **`SuggestedValues`** property.

Setting the Azimuth property to one of these two values yields a complete computation for that section. This achieves the same result as using the four-parameter **`SetTvdInclinationDlsAzimuth`** method, because all four required parameters are now known.

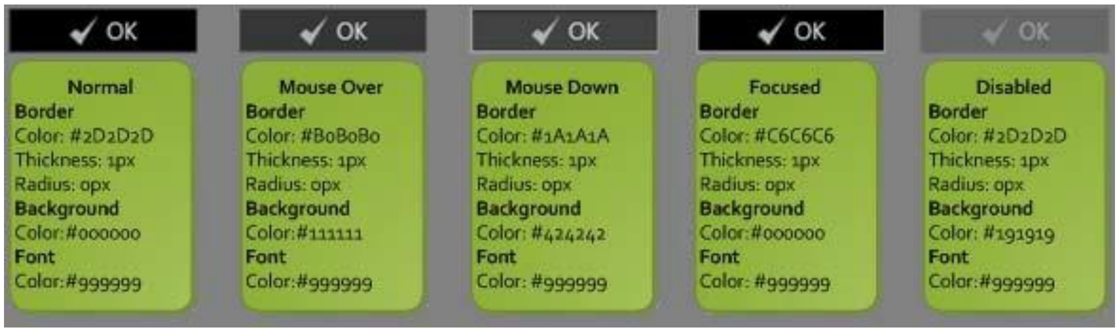
C UI controls geometry and color

In this chapter

UI controls

This appendix shows the standard geometry and color for Studio Manager user interface controls.

Buttons



Normal
Border
Color: #2D2D2D
Thickness: 1px
Radius: 0px
Background
Color:#000000
Font
Color:#999999

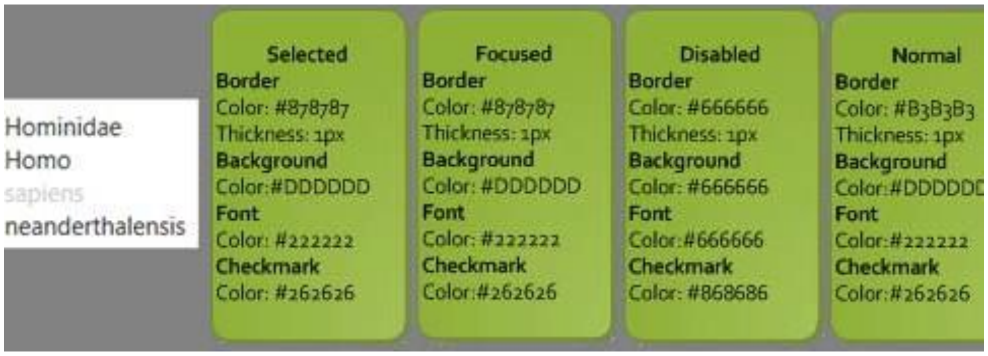
Mouse Over
Border
Color: #B0B0B0
Thickness: 1px
Radius: 0px
Background
Color:#111111
Font
Color:#999999

Mouse Down
Border
Color: #1A1A1A
Thickness: 1px
Radius: 0px
Background
Color: #424242
Font
Color: #999999

Focused
Border
Color: #C6C6C6
Thickness: 1px
Radius: 0px
Background
Color:#000000
Font
Color: #999999

Disabled
Border
Color: #2D2D2D
Thickness: 1px
Radius: 0px
Background
Color: #191919
Font
Color:#999999

Check boxes

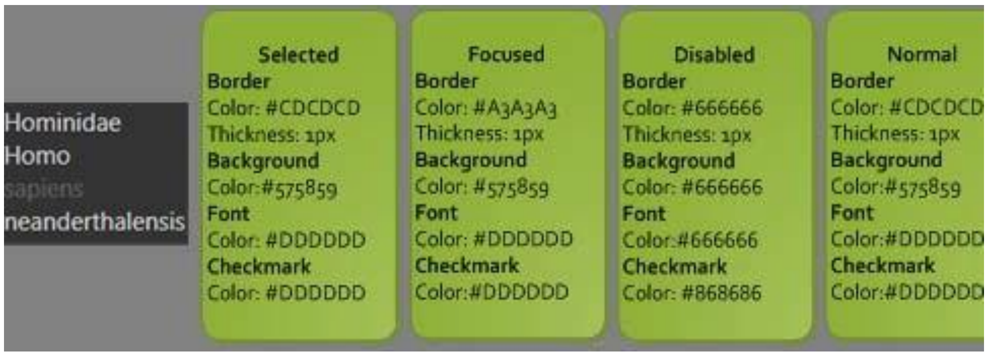


Selected
Border
Color: #878787
Thickness: 1px
Background
Color:#DDDDDD
Font
Color: #222222
Checkmark
Color: #262626

Focused
Border
Color: #878787
Thickness: 1px
Background
Color: #DDDDDD
Font
Color: #222222
Checkmark
Color:#262626

Disabled
Border
Color: #666666
Thickness: 1px
Background
Color: #666666
Font
Color:#666666
Checkmark
Color: #868686

Normal
Border
Color: #B3B3B3
Thickness: 1px
Background
Color:#DDDDDD
Font
Color:#222222
Checkmark
Color:#262626



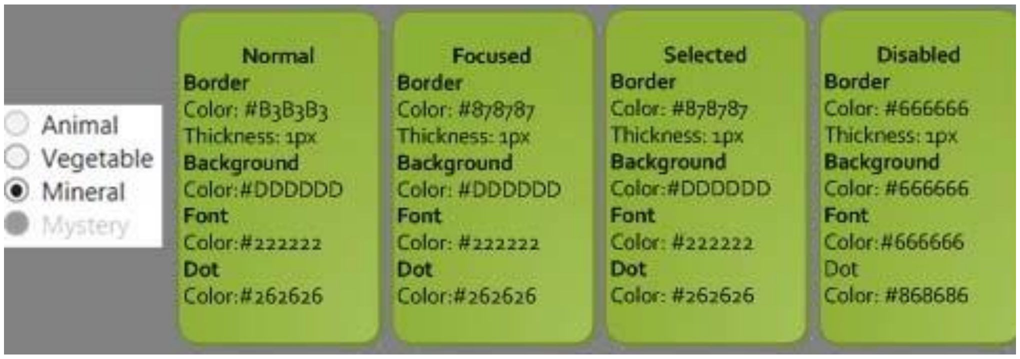
Selected
Border
Color: #CDCDCD
Thickness: 1px
Background
Color:#575859
Font
Color: #DDDDDD
Checkmark
Color: #DDDDDD

Focused
Border
Color: #A3A3A3
Thickness: 1px
Background
Color: #575859
Font
Color: #DDDDDD
Checkmark
Color:#DDDDDD

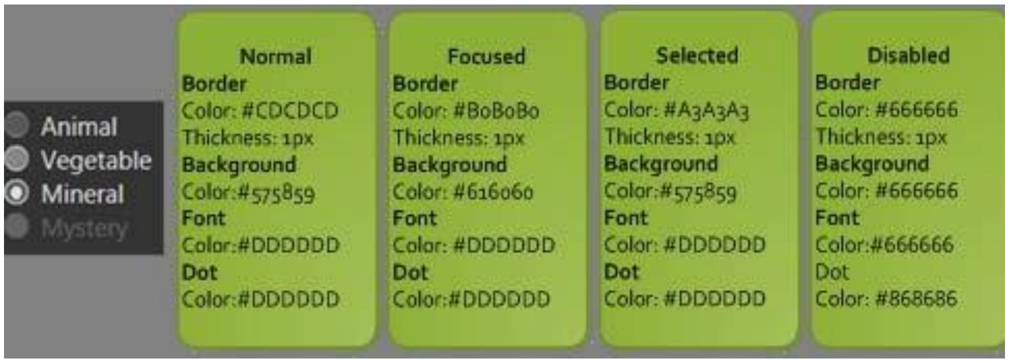
Disabled
Border
Color: #666666
Thickness: 1px
Background
Color: #666666
Font
Color:#666666
Checkmark
Color: #868686

Normal
Border
Color: #CDCDCD
Thickness: 1px
Background
Color:#575859
Font
Color:#DDDDDD
Checkmark
Color:#DDDDDD

Radio buttons

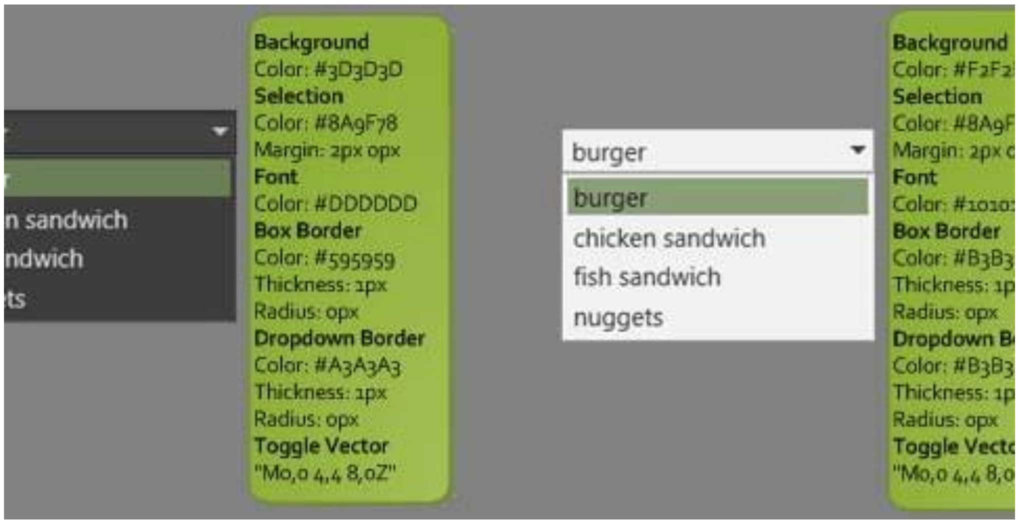


Normal	Focused	Selected	Disabled
Border	**Border**	**Border**	**Border**
Color: #B3B3B3	Color: #878787	Color: #878787	Color: #666666
Thickness: 1px	Thickness: 1px	Thickness: 1px	Thickness: 1px
Background	**Background**	**Background**	**Background**
Color:#DDDDDD	Color: #DDDDDD	Color:#DDDDDD	Color: #666666
Font	**Font**	**Font**	**Font**
Color:#222222	Color: #222222	Color: #222222	Color:#666666
Dot	**Dot**	**Dot**	Dot
Color:#262626	Color:#262626	Color: #262626	Color: #868686



Normal	Focused	Selected	Disabled
Border	**Border**	**Border**	**Border**
Color: #CDCDCD	Color: #B0B0B0	Color: #A3A3A3	Color: #666666
Thickness: 1px	Thickness: 1px	Thickness: 1px	Thickness: 1px
Background	**Background**	**Background**	**Background**
Color:#575859	Color: #616060	Color:#575859	Color: #666666
Font	**Font**	**Font**	**Font**
Color:#DDDDDD	Color: #DDDDDD	Color: #DDDDDD	Color:#666666
Dot	**Dot**	**Dot**	Dot
Color:#DDDDDD	Color:#DDDDDD	Color: #DDDDDD	Color: #868686

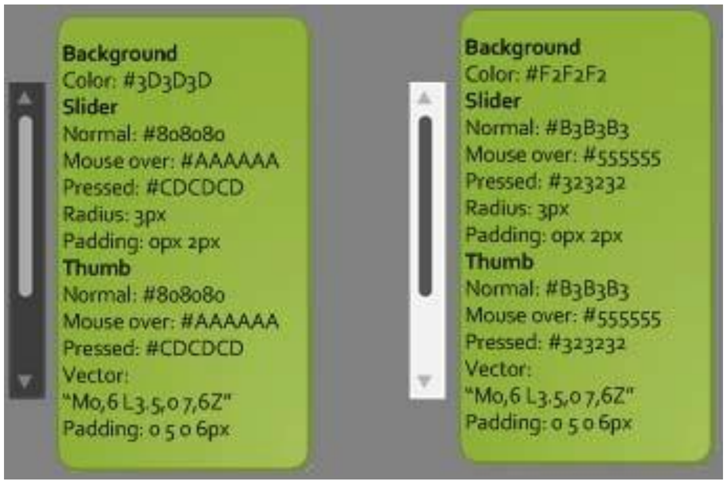
Combo boxes



Background
Color: #3D3D3D
Selection
Color: #8A9F78
Margin: 2px 0px
Font
Color: #DDDDDD
Box Border
Color: #595959
Thickness: 1px
Radius: 0px
Dropdown Border
Color: #A3A3A3
Thickness: 1px
Radius: 0px
Toggle Vector
"M0,0 4,4 8,0Z"

Background
Color: #F2F2
Selection
Color: #8A9F
Margin: 2px 0
Font
Color: #1010
Box Border
Color: #B3B3
Thickness: 1p
Radius: 0px
Dropdown B
Color: #B3B3
Thickness: 1p
Radius: 0px
Toggle Vecto
"M0,0 4,4 8,0

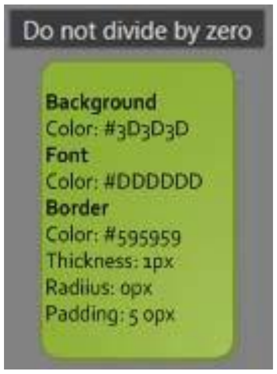
Scroll bar



Background
Color: #3D3D3D
Slider
Normal: #808080
Mouse over: #AAAAAA
Pressed: #CDCDCD
Radius: 3px
Padding: 0px 2px
Thumb
Normal: #808080
Mouse over: #AAAAAA
Pressed: #CDCDCD
Vector:
"M0,6 L3.5,0 7,6Z"
Padding: 0 5 0 6px

Background
Color: #F2F2F2
Slider
Normal: #B3B3B3
Mouse over: #555555
Pressed: #323232
Radius: 3px
Padding: 0px 2px
Thumb
Normal: #B3B3B3
Mouse over: #555555
Pressed: #323232
Vector:
"M0,6 L3.5,0 7,6Z"
Padding: 0 5 0 6px

Tooltips



Do not divide by zero

Background
Color: #3D3D3D
Font
Color: #DDDDDD
Border
Color: #595959
Thickness: 1px
Radiius: 0px
Padding: 5 0px