Deltek ProPricer

ProPricer Basic API Programming Guide

© Deltek, Inc. All Rights Reserved

Contents

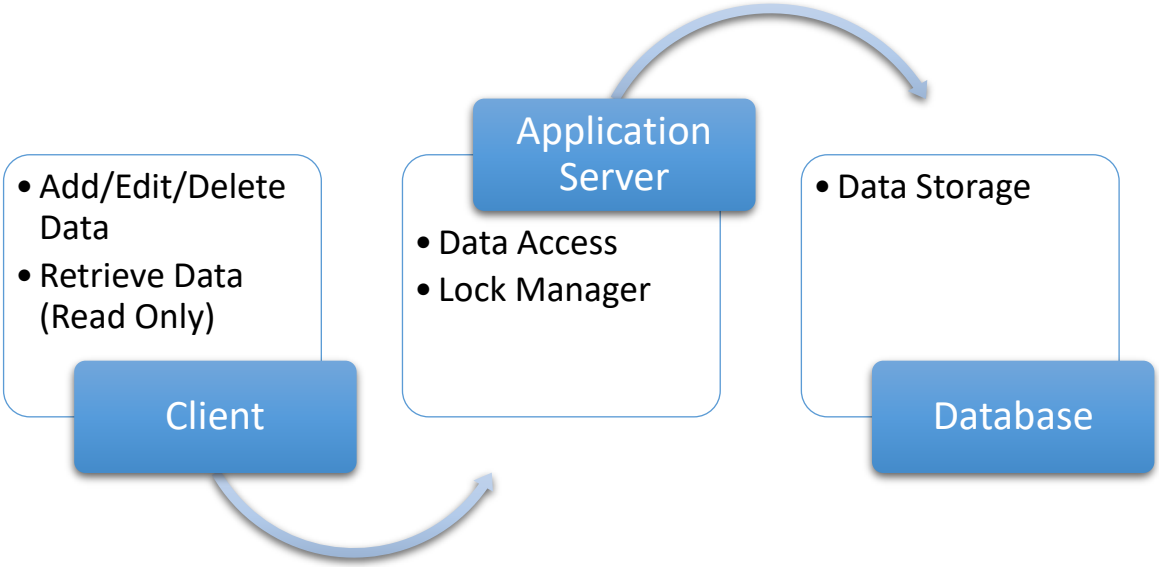
ProPricer/API client communication 1
Licensing 2
Licensing model 2
Dynamic link libraries 3
API development environment 4
Entities 4
Workspace 4
Methods 4
Async 5
Application server connections 6
Manual connection 6
Connection file 6
Establishing a connection 7
Global libraries 27
Rate tables 41
Proposals 52
Resources 59
Proposal operations 69
Audit Log entries 80
Commonly used items 84

ProPricer/API client communication

The ProPricer 9 API provides access to all ProPricer data to allow integration with other applications. Using this API, you can connect to a running ProPricer server (locally or in a separate computer), log in to a specific database, and interact with all data in ProPricer. You can read, change, add, and even delete ProPricer data.

The client is the program that the user sees when accessing the database. Whether using SQL or Oracle, it is necessary to deploy the client software after the application server and database have been successfully installed.

The current version of ProPricer supports both Microsoft SQL and Oracle databases. Each type of database has unique characteristics and applications in the workplace. The API functionality and code are the same, regardless of which type of database is selected.



Licensing

Licensing model

The API license requires a separate registration code and activation.

API client

The API client dynamic link libraries (.dlls) contain all the functionality needed to create standalone applications that use the ProPricer data. Contractor Edition, Estimator Edition, or Government Edition do not need to be installed for the application to run and connect to an application server.

If you are using the API .dlls, a separate license must be purchased for the application to run and connect to the application server. Single or concurrent licenses are available. Within the code of the application, you will have to account for registration and activation.

Dynamic link libraries

The dynamic link libraries (.dlls) are shared libraries that contain instructions you can use to call upon specific ProPricer client functionality to do certain things.

EBS.Core.dll

This library contains the base classes for Collections, IO, and Threading. It includes some algorithms, like Math, Sort, and some useful helpers for Stream, Array, Dispose, and more.

EBS.Core.Rpc.dll

This library provides the communication protocol and behavior.

EBS.Core.Logging.dll

This library provides extensions to configure Serilog logging from application settings.

EBS.ProPricer.EditionManager

This library provides access to ProPricer edition metadata, and it contains registration helpers and base classes to implement a registration model.

EBS.ProPricer.Client.dll

This library provides access to the ProPricer Data Model. This is the main .dll for accessing data.

EBS.ProPricer.Client.config.json

This configuration file allows logging to be enabled, and invalid SSL certificates (for example, self-signed certificates) to be used.

EBS.ProPricer.Configuration.dll

This library provides the ProPricer edition configuration.

EBS.ProPricer.Registration.dll

This library provides the ProPricer Registration and Activation Model. It handles the ProPricer Key, both Single and Concurrent. It also manages the communication with the ProPricer licensing server for Activation/Deactivation/Refresh Activation.

PLUSManaged.dll

Third party software to provide License Management.

API development environment

The API development environment can be any Integrated Development Environment (IDE) that can take advantage of Microsoft .NET 8. In this guide, all exercises are done in Visual Studio 2022 using C#. Other .NET languages can be used as the application dictates.

Entities

The ProPricer API is organized in a very structured way. It's an entity-based structure, and all entities have a collection and an object. Before you can work with an entity, you need to open it by calling the Open() method. At the end, you must close the entity by calling the Close() method.

When working with entities, add a new element using the AddNew() method on a collection. You start editing an entity using BeginEdit(), and in either case, you finish the edits with EndEdit(). You should have try/catch after AddNew() or BeginEdit() methods, and make sure to call CancelEdit() in catch block.

> Note: Starting with ProPricer v9.3 build 107, you should use .Items() to access the items in a collection.

Workspace

The first step will be to establish the workspace. The top entity is the Workspace class. When you log in to a ProPricer server, you open the Workspace entity. This is a prerequisite before accessing any other ProPricer entity. This entity is a starting point for any type of proposal or global access. All other entities are accessed through an instance of the Workspace class. For example, the Users collection is in workspace.Users.

Methods

The methods associated with the workspace differ depending on the action you are attempting to accomplish. Each method is exposed to Microsoft IntelliSense. IntelliSense is the general term for several features: List Members, Parameter Info, Quick Info, and Complete Word. These features help you learn more about the code you are using, keep track of the parameters you are typing, and add calls to properties and methods with only a few keystrokes.

Async

The ProPricer API offers only synchronous calls for most methods. The two exceptions are connecting to the ProPricer Server using DataCenter.OpenAsync() and opening the workspace using dataServer.OpenWorkspaceAsync().

These methods must be async-only because the underlying communication library (WebSocket) has no sync variants of async methods. Thus, the user of the library (in this case, the API app) cannot provide real synchronous equivalents of its APIs because it would need to solve how to call async code from non-async. This issue is delegated to the caller by providing async-only methods.

In general, the recommended approach is to use async "all the way down." In very specific cases where async cannot be used, there are several ways one can wait synchronously on an async method (Task or ValueTaks) - .Wait(), .Result or .GetAwaiter().GetResult()

The problem is that they all can cause deadlock in special single-threaded environments like WinForms. ASP.NET Core and Console have no such issue.

When async is not possible, then GetAwaiter().GetResult() is preferred since it eliminates (unwraps) AggregateExceptions.

> Note: Starting with ProPricer v9.5 build 103, you should use async methods to connect to ProPricer Server.

Application server connections

There are a variety of ways to connect to the ProPricer application server. The two most common ways are through a manual process or using the connection file (.ppc) parameters to access the server.

Manual connection

A manual connection lets the developer hard code the server connection values directly in the code. The end user of the application cannot input or change the server variables, and the application will only connect using those parameters. This method is good for testing the application and the connection results. If there is more than one connection available to the user, this would not be recommended.

These are the main connection parameters to define. They can also be viewed with ClientConfig.exe that is installed with the application server and client.

- Server Name: the name or address of the application server.
- Port: the application server port that is found on the Options tab of the application server.
- Database Connection: the connection name found for the connections created within the application server.

Connection file

The ProPricer connection file (.ppc) is an encrypted XML file that contains all the connection parameters. The file is generated by the Server Manager and can be stored locally or on a network drive. Accessing the connection file allows for easier connections since the server, port, and connection name have already been defined. Allowing the user to select the connection file gives them more flexibility in changing databases to perform their tasks.

These are the values stored within the connection file:

- Server Name: application server name.
- Port: application server port.
- Connection Name: database connection name.

Establishing a connection

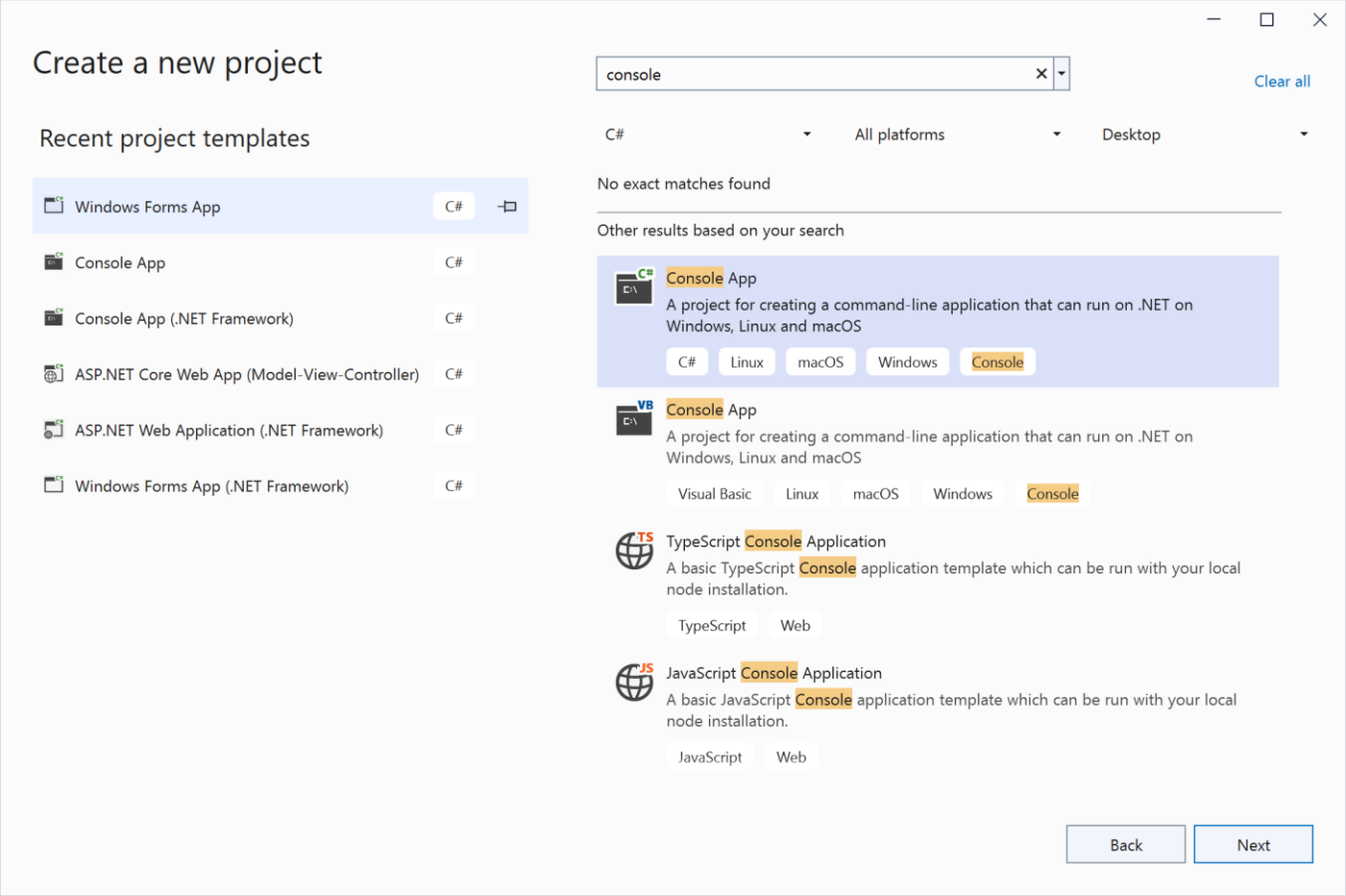
The first step to any ProPricer API application is to establish a connection to the ProPricer application server. Once you are connected to the server, you can perform a variety of tasks within your application.

The purpose of the connection to the application server is to gain access to all the ProPricer data. The application server controls all interaction between any client and the database. The application server also controls locking, and enforces user rights.

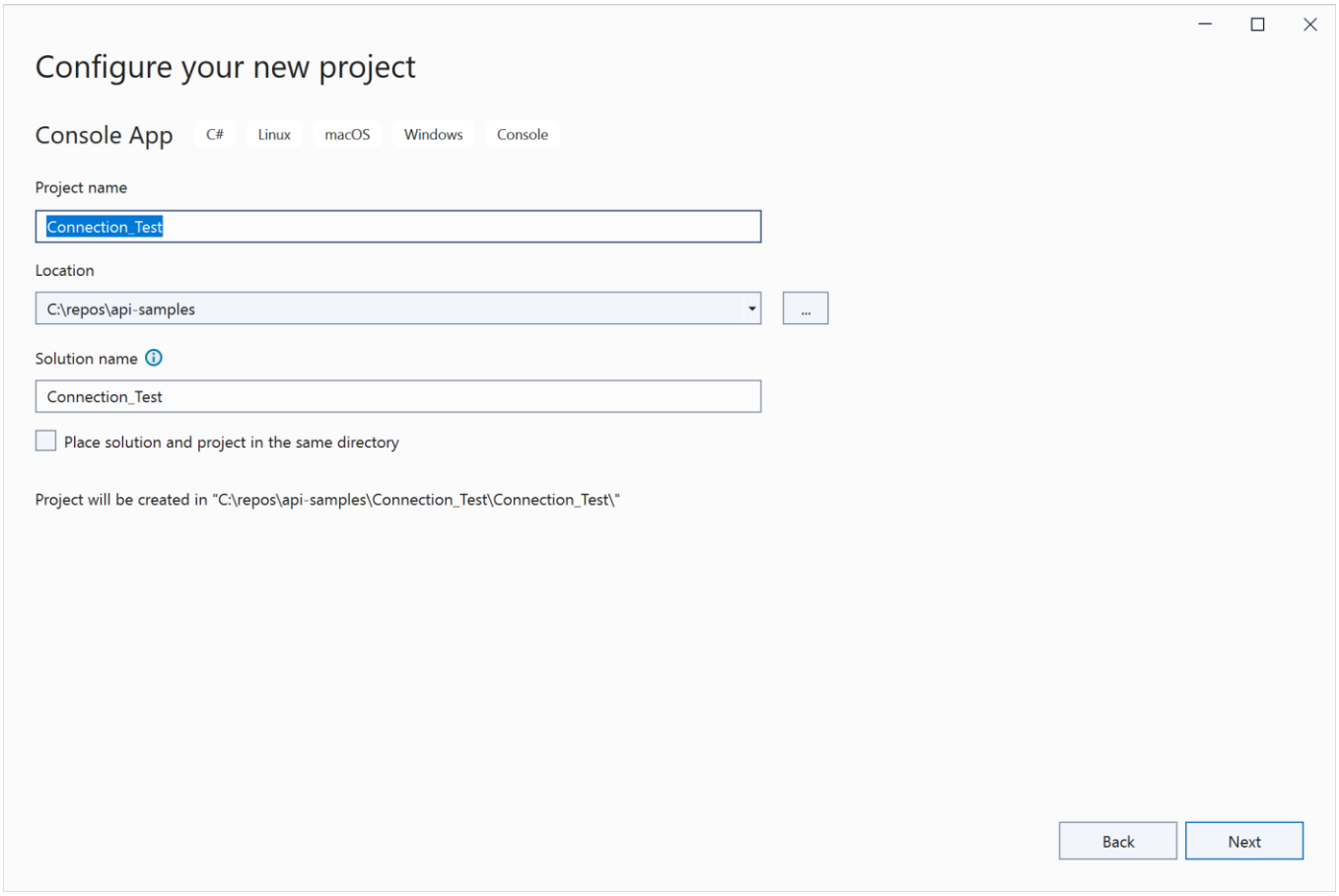
The connection allows you to read, add, edit, and delete ProPricer data based on your application requirements.

How to create a manual application server connection

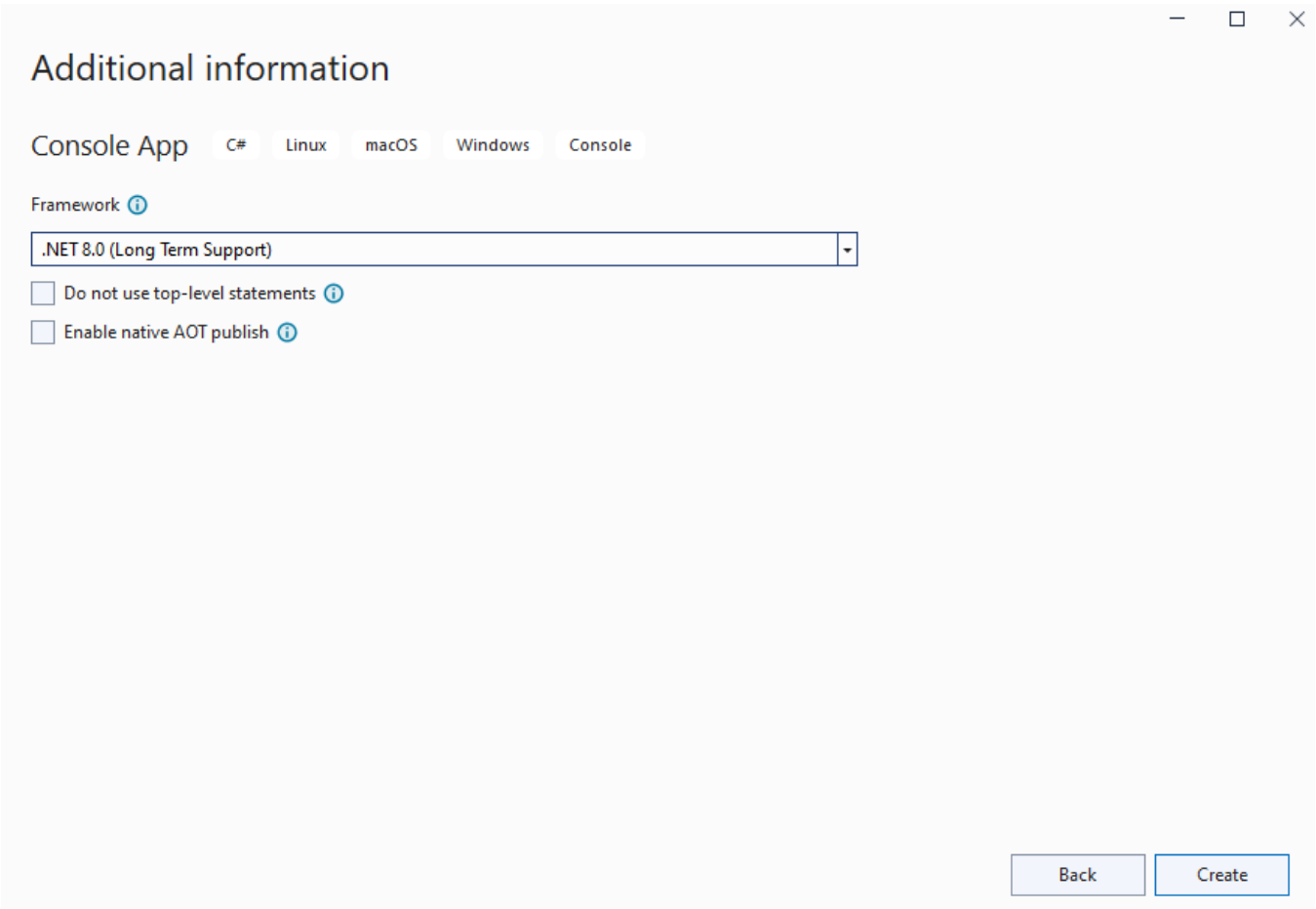
1. In Visual Studio, start a new project. The **Create new project** wizard will appear.
2. In the template list, select **Console App**, and then click **Next**.



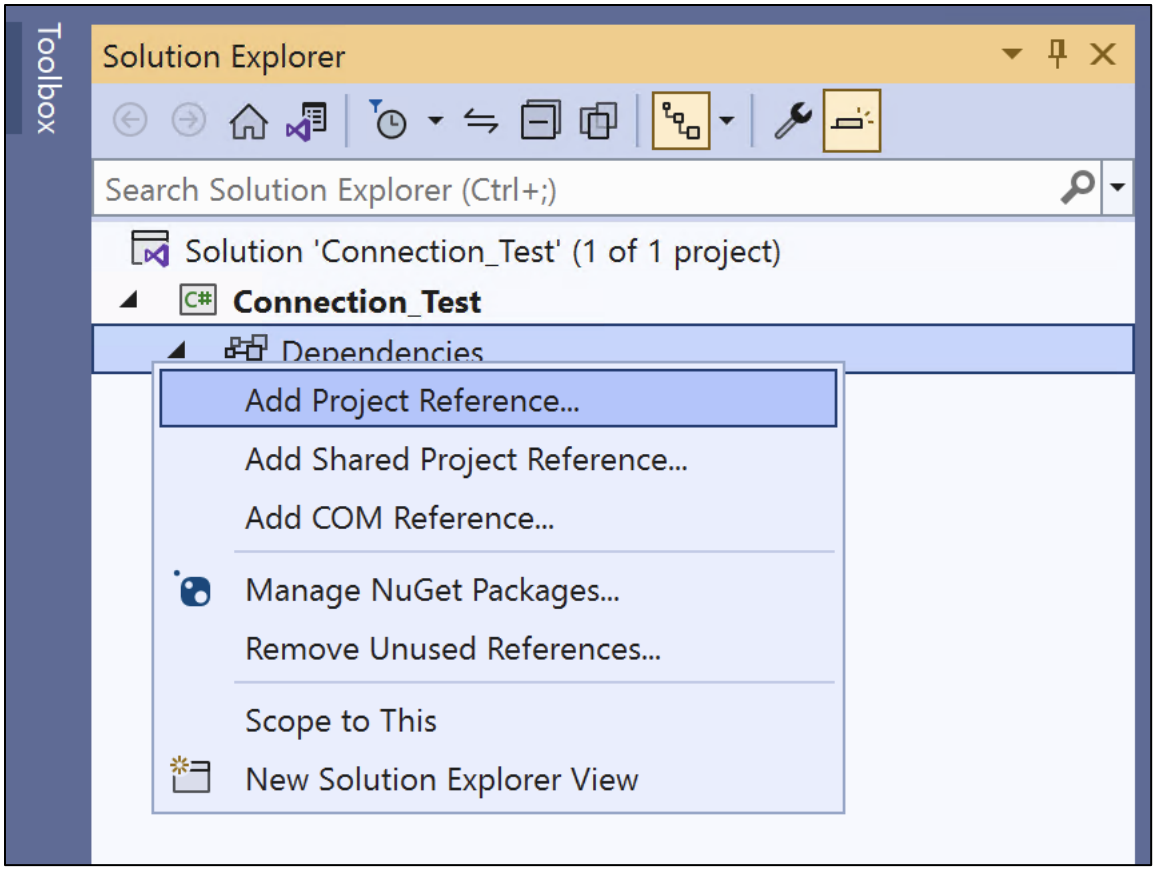
3. Enter *Connection_Test* for the project name, select a save location, and then click **Next**.



4. Select the framework **.NET 8.0 (Long Term Support)**, and then click **Create**.



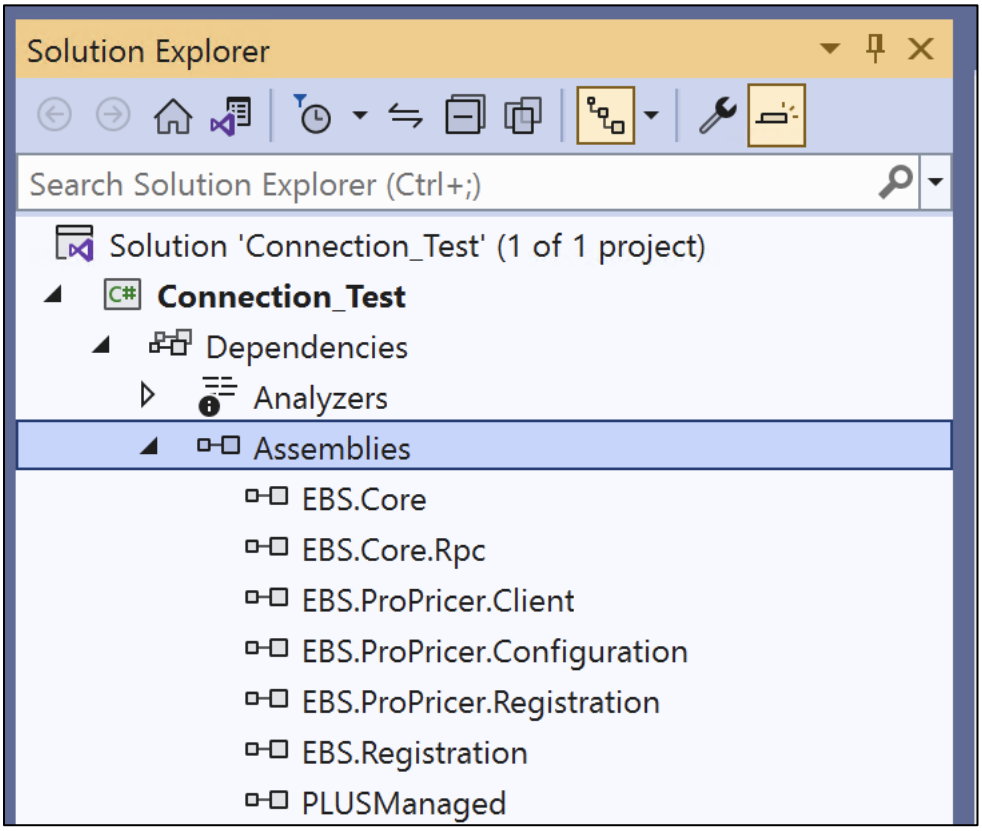
5. When the new solution opens, go to the Solution Explorer. In the Connection_Test project, right-click **Dependencies** > **Add Project Reference** to add the ProPricer API references (.dlls).



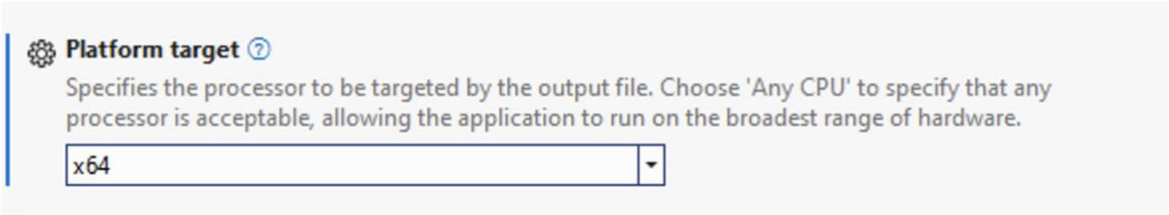
6. The Reference Manager will appear. Click **Browse** to locate all the .dlls located in the API.zip file previously provided and extracted to a local folder.

7. Click **OK** to add the references.

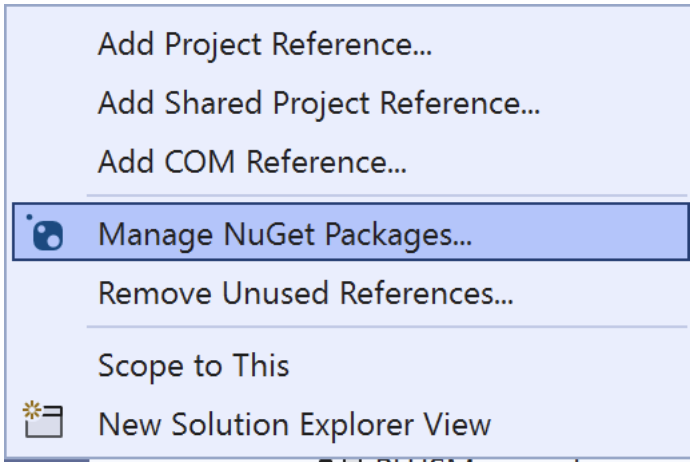
8. Verify that the Solution Explorer shows the added references under **Assemblies**.



9. In the project properties, set **Platform target** to **x64**.



10. Right-click **Dependencies** > **Manage NuGet Packages** to add the required packages.



11. On the **Browse** tab:

    a. Enter *Microsoft.AspNetCore.Http.Connections.Client* in the search box, select version 8.0.15, and click **Install**.

    b. Enter *Microsoft.Extensions.Configuration.Json* in the search box, select version 9.0.4, and click **Install**.

    c. Enter *Microsoft.Extensions.DependencyInjection* in the search box, select version 9.0.4, and click **Install**.

    d. Enter Microsoft.Extensions.Logging.Configuration in the search box, select version 9.0.4, and click **Install**.

    e. Enter *Microsoft.Extensions.Logging.Debug* in the search box, select version 9.0.4, and click **Install**.

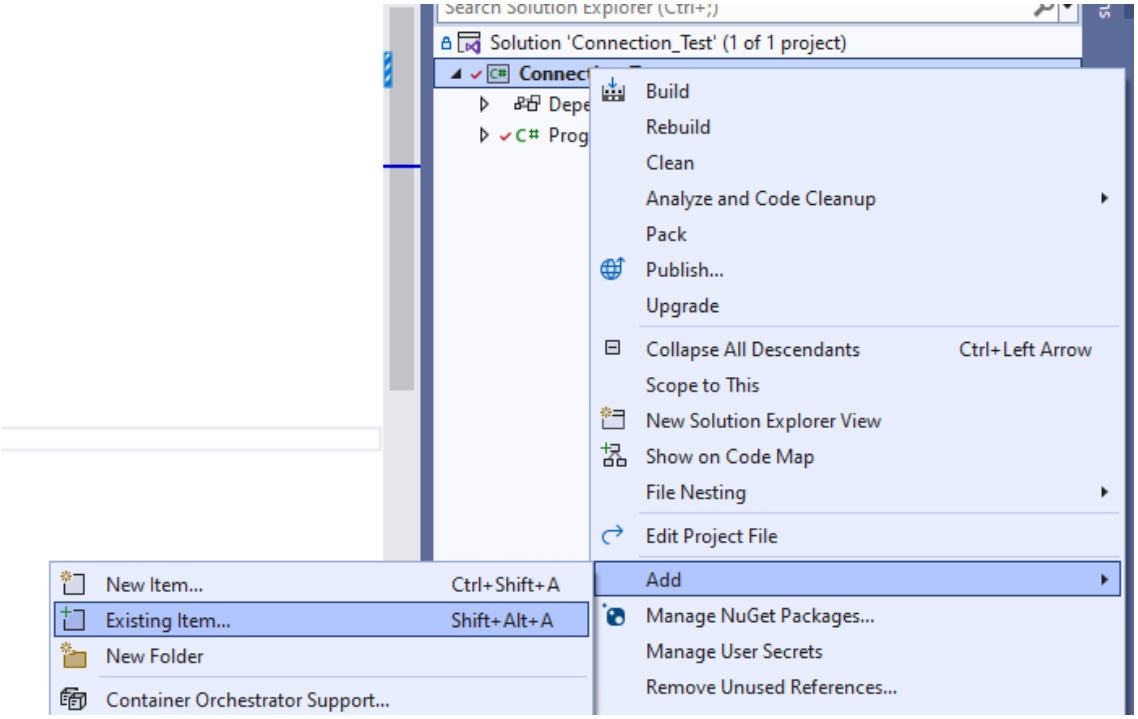
    f. Enter *Microsoft.Identity.Web* in the search box, select version 3.8.4, and click **Install**.

    g. Enter *Serilog* in the search box, select version 4.2.0, and click **Install**.

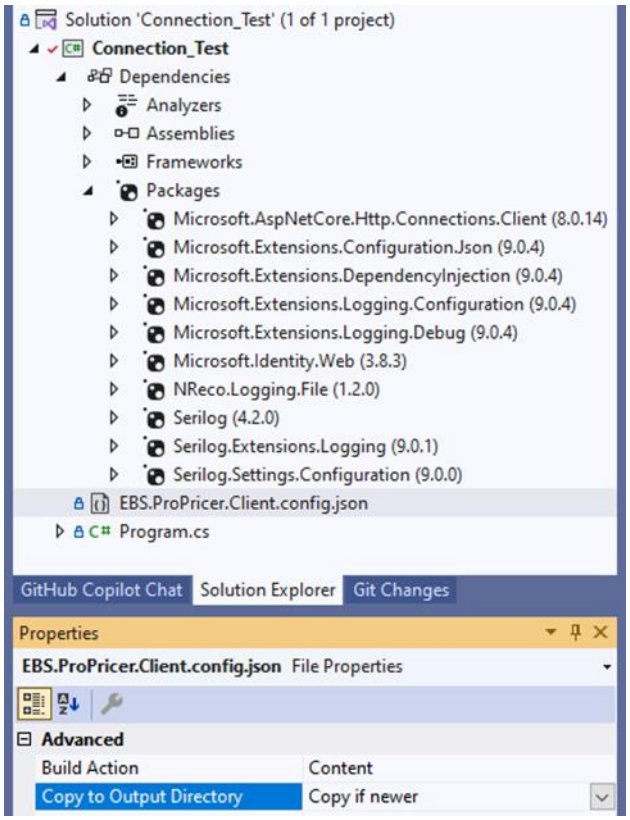
    h. Enter *Serilog.Extensions.Logging* in the search box, select version 9.0.1, and click **Install**.

    i. Enter *Serilog.Settings.Configuration* in the search box, select version 9.0.0, and click **Install**.

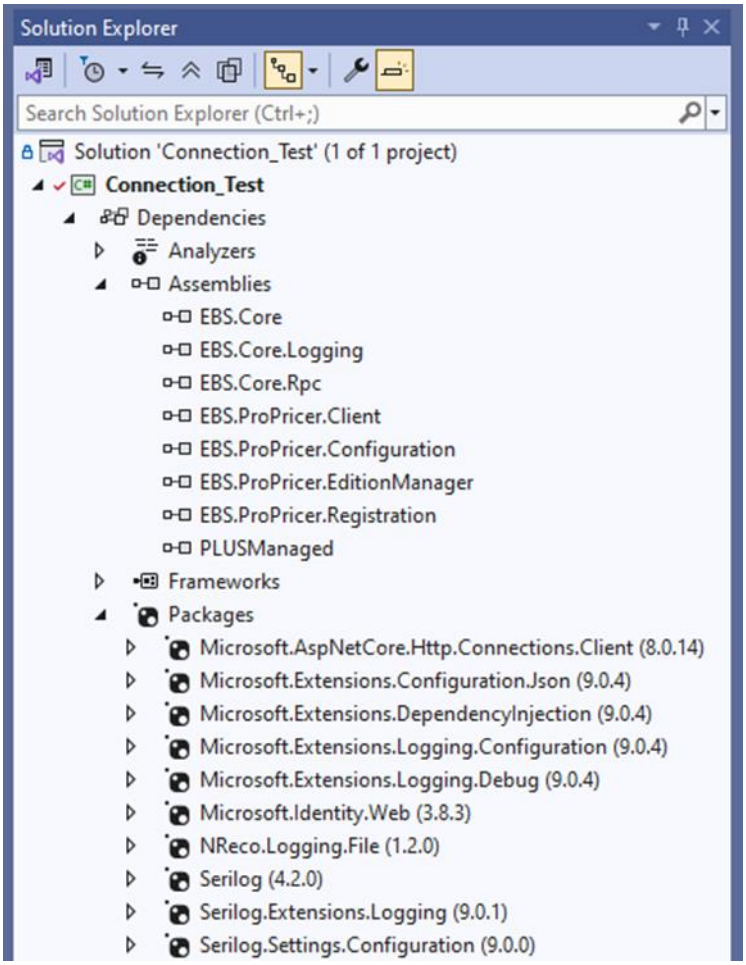
12. In some cases, you can have issues with the verification of the certificate used to connect to the server, to avoid this we must add a configuration file to the project. Right-click **Main .csproj file** > **Add** > **Existing Item** and search **EBS.ProPricer.Client.config.json** in the same path where the .dll files used in the project are located**:**



13. Click on the configuration file and in the properties change **Build Action** to **Content** and **Copy to Output Directory** to **Copy if newer**:



14. Verify that the Solution Explorer shows the following under **Packages**.



15. To use the references and connect to the application server, you must add the appropriate ProPricer namespaces to your solution. Add the following namespaces.

```
using EBS.ProPricer.Client;
using EBS.ProPricer.Data;
using EBS.ProPricer.Model;
using EBS.ProPricer.Model.General;
using EBS.ProPricer.Registration;
```

16. Within the Main method, add a Try/Catch block and the following code.

```
try
{
      Console.WriteLine("Connecting...");
      var dataCenter = await DataCenter.OpenAsync("Your ProPricer Server FQDN",
8092);      Console.WriteLine("Data Center Ready ...");
      var dataServer = dataCenter.DataServers.Find("Your Database Connection
Name");
}
catch (Exception ex)
{
      Console.WriteLine($"Error: {ex.Message}");
      Console.Read();
}
```

Code comments:

- The dataCenter variable invokes the EBS.ProPricer.DataCenter.Client.OpenAsync(string servername, int port) method. The server name and port are both found in ProPricer Server Manager.
- The dataCenter.DataServers.Find(string name) method is assigned to the dataServer variable. The data server contains the connection name that is contained within ProPricer Server Manager. More than one connection can be set up within the manager, so the connection should be verified by finding it first.

17. Now that the dataCenter and dataServer are created, the existence of the server must be verified. Add the following code.

```
      Console.WriteLine("Validating Data Server ...");
      if (dataServer == null)
      {
            Console.WriteLine("Invalid Data Server");
            return;
      }
      Console.WriteLine($"Valid Data Server {dataServer.Info.Name}...");
```

18. Once it is verified that the server is not null, the workspace can be opened. The workspace is the base entity that is needed before accessing any ProPricer data. Add the following code.

```
      Console.WriteLine("Opening Workspace ...");
      using (var workspace = await dataServer.OpenWorkspaceAsync(GetLoginInfo,
GetRegistration, GetActivation))
      {
            Console.WriteLine("Workspace Ready/Open ...");
            workspace.Proposals.Open();
            try
            {
                  Console.WriteLine("Proposal List...");
                  foreach (var proposal in workspace.Proposals.Items())
                  {
                        Console.WriteLine($"{proposal.Name}, Ver.
{proposal.Version}");
                  }
            }
            finally
            {
                  workspace.Proposals.Close();
            }
      }
      Console.WriteLine("Workspace Closed ...");
      Console.Read();
```

Code comments:

- The dataServer.OpenWorkspaceAsync((LogonInfoProvider logonInfoProvider, RegistrationProvider registrationProvider, ActivationProvider activationProvider) opens the workspace. The GetLoginInfo, GetRegistration, and GetActivation methods handle the user login information, and the registration and activation of the API license. These methods will be created in later steps.

- The workspace.Proposals.Open() method loads all proposals from the server to the client, and the try finally ensures the workspace.Proposals.Close() is called. Note: the Proposals collection is opened by default after login. This code is not strictly necessary, but it's a good practice when working with ProPricer collections.

- The foreach statement loops through all proposals within the current workspace. Additional detail will be provided later in this guide.

19. Now that the main connection assignments are made, the login information, registration method, and activation method need to be set up. Add the following code.

```
static bool GetLoginInfo(LogonInfo info)
{
	info.UserName = "sysadmin";
	info.Password = "sysadmin";
	return true;
}
```

Code comments:

- The LogonInfo provider handles the username, password, and logon type. The username and password relate to the user's setup in ProPricer. The logon type refers to the type in the System Configuration of ProPricer.
  - ProPricer: LogonType.Regular
  - Windows (Interactive): LogonType.Windows
  - Windows (Integrated): LogonType.Integrated
  - Microsoft Entra (Azure AD): LogonType.AzureAd
  - Not defining the LogonType defaults to Regular.

20. Add the GetRegistration method to validate the registration code. Add the following code.

```
static bool GetRegistration(RegistrationInformation registrationInfo)
{
      registrationInfo.Key = "Your ProPricer API Registration Key";
      if (!string.IsNullOrEmpty(registrationInfo.LastError))
            Console.WriteLine(registrationInfo.LastError);
      return true;
}
```

Code comments:

- Instead of hard coding a value, you can set up a registration input method that allows the user to add the registration code.
- The RegistrationInformation class contains specific variables to help you with the registration process. Here are the most used.
  - IsRegistered: Determine if the product is registered.
  - Key: Assign the registration key.
  - LastError: Return the last error when attempting to register.
  - IsEvaluationkey: Determine if the key is an evaluation key.

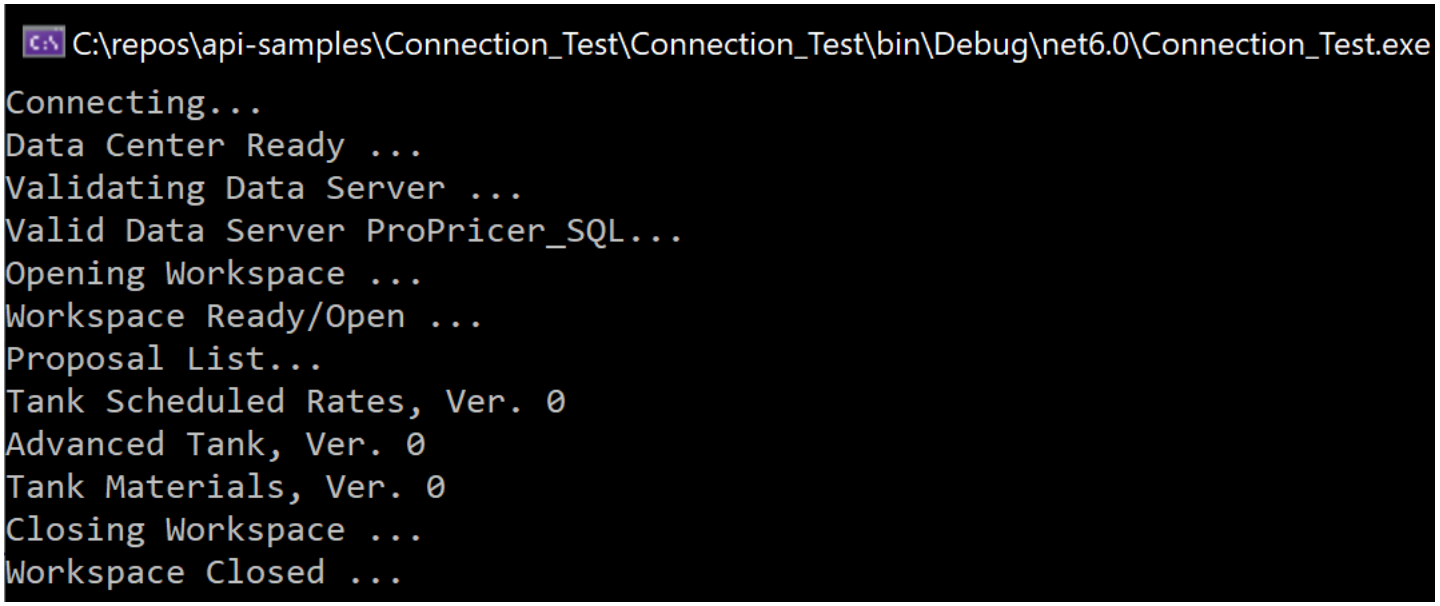
21. Add the GetActivation method to handle the activation of the API product. Add the following code.

```
static bool GetActivation(ProPricerActivation activation, ref bool changeKey)
{
	try
	{
		changeKey = true;
		if (!activation.Status.IsActivated())
		{
			Console.WriteLine("Activating key online...");
			activation.Activate();
		}
	}
	catch (Exception e)
	{
		Console.WriteLine(e.Message);
		Console.WriteLine(e.StackTrace);
	}
	return true;
}
```

Code comments:

- You can also capture if the activation is valid, and offer the option to activate offline (By Code).
- The ProPricerActivation class contains specific methods related to the activation process. Here are the most used.
  - Activate: Activate the product through Deltek servers.
  - CheckActivationResponse: Get the activation response message.
  - CheckDeactivationResponse: Get the deactivation response.
  - Deactivate: Deactivate the product.
  - IsActivated: Check for activation.
  - lastErrorDescription: Get the error if activation or deactivation fails.

22. On the Connection_Test solution, click **Start** to verify the data server connection and the opening of the workspace.



Complete solution

```
using EBS.ProPricer.Client;
using EBS.ProPricer.Data;
using EBS.ProPricer.Model;
using EBS.ProPricer.Model.General;
using EBS.ProPricer.Registration;

try
{
	Console.WriteLine("Connecting...");
	var dataCenter = await DataCenter.OpenAsync("Your ProPricer Server FQDN", 8092);
	Console.WriteLine("Data Center Ready ...");
	var dataServer = dataCenter.DataServers.Find("Your Database Connection Name");
	Console.WriteLine("Validating Data Server ...");
	if (dataServer == null)
	{
		Console.WriteLine("Invalid Data Server");
		return;
	}
	Console.WriteLine($"Valid Data Server {dataServer.Info.Name}...");
	Console.WriteLine("Opening Workspace ...");
	using (var workspace = await dataServer.OpenWorkspaceAsync(GetLoginInfo,
GetRegistration, GetActivation))
	{
		Console.WriteLine("Workspace Ready/Open ...");
		workspace.Proposals.Open();
		try
		{
			Console.WriteLine("Proposal List...");
			foreach (var proposal in workspace.Proposals.Items())
			{
				Console.WriteLine($"{proposal.Name}, Ver. {proposal.Version}");
			}
		}
		finally
		{
			workspace.Proposals.Close();
		}
	}
	Console.WriteLine("Workspace Closed ...");
	Console.Read();
}
catch (Exception ex)
{
	Console.WriteLine($"Error: {ex.Message}");
	Console.Read();
}

static bool GetLoginInfo(LogonInfo info)
{
	info.UserName = "sysadmin";
	info.Password = "sysadmin";
	return true;
}

static bool GetRegistration(RegistrationInformation registrationInfo)
{
	registrationInfo.Key = "Your ProPricer API Registration Key";
```

```
	if (!string.IsNullOrEmpty(registrationInfo.LastError))
		Console.WriteLine(registrationInfo.LastError);
	return true;
}

static bool GetActivation(ProPricerActivation activation, ref bool changeKey)
{
	try
	{
		changeKey = true;
		if (!activation.Status.IsActivated())
		{
			Console.WriteLine("Activating key online...");
			activation.Activate();
		}
	}
	catch (Exception e)
	{
		Console.WriteLine(e.Message);
		Console.WriteLine(e.StackTrace);
	}
	return true;
}
```

How to create an application server connection through a connection file

The code is almost identical to the previous exercise except the connection parameters are missing. Those parameters will now be defined with the EBS.ProPricer.Configuration.Client.ClientConfiguration class.

1. Add or replace the following code to create a new connection and retrieve the connection parameters.

```
      Console.WriteLine("Get ppc Information...");

      var conPP = new EBS.ProPricer.Configuration.Client.ClientConfiguration();
      conPP.LoadConfiguration(@"Path to your PPC file");

      var dataCenter = await DataCenter.OpenAsync(conPP.ServerName,
conPP.ServerPort);

      Console.WriteLine("Data Center Ready ...");
      var dataServer = dataCenter.DataServers.Find(conPP.ConnectionName);
```

Code comments:

- The new conPP ClientConfiguration instance allows you to load the connection file using the LoadConfiguration(string filePath) method.
- Once the .ppc file is loaded, the ServerName, ServerPort, and ConnectionName can be used in the DataCenter.OpenAsync and DataServers.Find methods.

2. Click **Start** to test the solution and verify the new server connection parameters.

```
C:\repos\api-docs\examples\Connection_Test_PPC\Connection_Test_PPC\bin\Debug\net6.0\Connection_Test_PPC.exe
Get ppc Information...
Data Center Ready ...
Validating Data Server ...
Valid Data Server ProPricer_SQL...
Opening Workspace ...
Workspace Ready/Open ...
Proposal List...
Advanced Tank, Ver. 0
Tank Materials, Ver. 0
Tank Scheduled Rates, Ver. 0
Closing Workspace ...
Workspace Closed ...
```

Complete solution

```
using EBS.ProPricer.Client;
using EBS.ProPricer.Data;
using EBS.ProPricer.Model;
using EBS.ProPricer.Model.General;
using EBS.ProPricer.Registration;

try
{
	Console.WriteLine("Get ppc Information...");

	var conPP = new EBS.ProPricer.Configuration.Client.ClientConfiguration();
	conPP.LoadConfiguration(@"Path to your PPC file");

	var dataCenter = await DataCenter.OpenAsync(conPP.ServerName, conPP.ServerPort);

	Console.WriteLine("Data Center Ready ...");
	var dataServer = dataCenter.DataServers.Find(conPP.ConnectionName);
	Console.WriteLine("Validating Data Server ...");
	if (dataServer == null)
	{
		Console.WriteLine("Invalid Data Server");
		return;
	}
	Console.WriteLine($"Valid Data Server {dataServer.Info.Name}...");
	Console.WriteLine("Opening Workspace ...");
	using (var workspace = await dataServer.OpenWorkspaceAsync(GetLoginInfo,
GetRegistration, GetActivation))
	{
		Console.WriteLine("Workspace Ready/Open ...");
		workspace.Proposals.Open();
		try
		{
			Console.WriteLine("Proposal List...");
			foreach (var proposal in workspace.Proposals.Items())
			{
				Console.WriteLine($"{proposal.Name}, Ver. {proposal.Version}");
			}
		}
		finally
		{
			workspace.Proposals.Close();
		}
	}
	Console.WriteLine("Workspace Closed ...");
	Console.Read();
}
catch (Exception ex)
{
	Console.WriteLine("Error: ");
	Console.WriteLine(ex.Message);
	Console.WriteLine(ex.StackTrace);
	Console.Read();
}

static bool GetLoginInfo(LogonInfo info)
{
	info.UserName = "sysadmin";
```

```
        info.Password = "sysadmin";
        return true;
}

static bool GetRegistration(RegistrationInformation registrationInfo)
{
        registrationInfo.Key = "Your ProPricer API Registration Key";
        if (!string.IsNullOrEmpty(registrationInfo.LastError))
                Console.WriteLine(registrationInfo.LastError);
        return true;
}

static bool GetActivation(ProPricerActivation activation, ref bool changeKey)
{
        try
        {
                changeKey = true;
                if (!activation.Status.IsActivated())
                {
                        Console.WriteLine("Activating key online...");
                        activation.Activate();
                }
        }
        catch (Exception e)
        {
                Console.WriteLine(e.Message);
                Console.WriteLine(e.StackTrace);
        }
        return true;
}
```

Global libraries

Global libraries contain a variety of non-proposal specific data. For example, the Resource Library contains a list of system-wide resources that can be used within other libraries or the proposals themselves. The API has access to all libraries through the established Workspace entity.

The global libraries house the core data needed for all related proposals, rate tables, CERs, and other miscellaneous information.

Establishing a connection to these libraries allows you to read, edit, add, and delete base information within ProPricer. These tables can also be used as validating parameters when attempting to add data in other related tables.

How to access the Resource Library

Workspace.GlobalLibrary provides access to all Global libraries.

Workspace.GlobalLibrary.Resources will access the workspace and retrieve the resources located in the Resources Library.

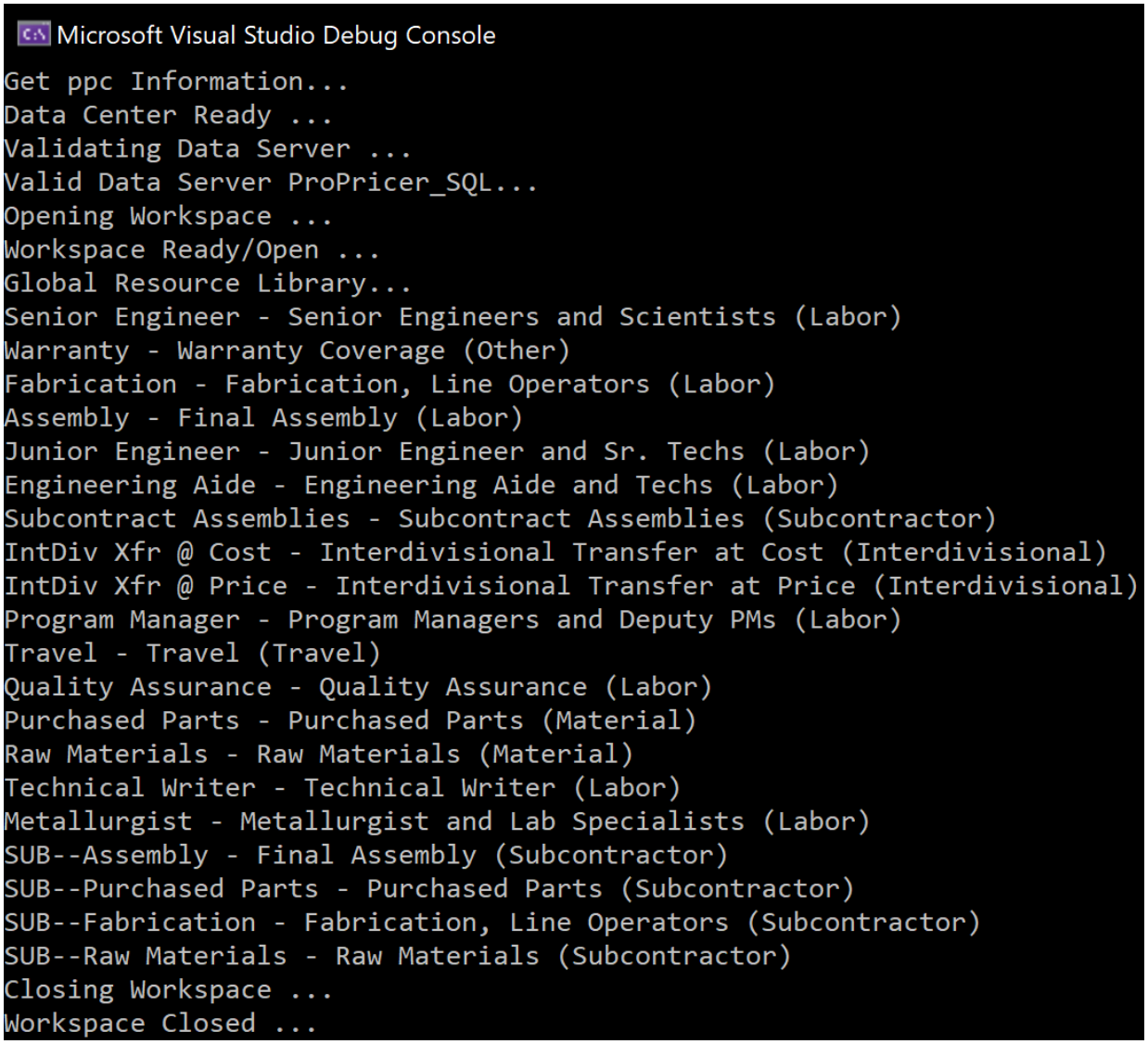
1. Use previous examples in this guide as a base.
2. Add or replace the following code to retrieve the resources from the library, and display their Name, Description, and Resource Type.

```
workspace.GlobalLibrary.Resources.Open();
try
{
    Console.WriteLine("Global Resource Library...");
    foreach (var resource in workspace.GlobalLibrary.Resources.Items())
    {
        Console.WriteLine($"{resource.Name} - {resource.Description} ({resource.Type})");
    }
}
finally
{
    workspace.GlobalLibrary.Resources.Close();
}
```

Code comments:

- The workspace.GlobalLibrary.Resources.Open() method loads all resources in the global library from the server to the client. The try finally ensures the workspace.GlobalLibrary.Resources.Close() is called.
- The workspace.GlobalLibrary.Resources.Items() collection in the foreach statement is accessing all the items in the Resource Library. Since this is an enumerable list, you can loop through all resources.

3. Click **Start** to run the solution.
4. Once the solution connects, the console will be populated with a list of resources from the Resource Library.



Complete solution

```
using EBS.ProPricer.Client;
using EBS.ProPricer.Data;
using EBS.ProPricer.Model;
using EBS.ProPricer.Model.General;
using EBS.ProPricer.Registration;

try
{
	Console.WriteLine("Get ppc Information...");

	var conPP = new EBS.ProPricer.Configuration.Client.ClientConfiguration();
	conPP.LoadConfiguration(@"Path to your PPC file");

	var dataCenter = await DataCenter.OpenAsync(conPP.ServerName, conPP.ServerPort);

	Console.WriteLine("Data Center Ready ...");
	var dataServer = dataCenter.DataServers.Find(conPP.ConnectionName);
	Console.WriteLine("Validating Data Server ...");
	if (dataServer == null)
	{
		Console.WriteLine("Invalid Data Server");
		return;
	}
	Console.WriteLine($"Valid Data Server {dataServer.Info.Name}...");
	Console.WriteLine("Opening Workspace ...");
	using (var workspace = await dataServer.OpenWorkspaceAsync(GetLoginInfo,
GetRegistration, GetActivation))
	{
		Console.WriteLine("Workspace Ready/Open ...");
		workspace.GlobalLibrary.Resources.Open();
		try
		{
			Console.WriteLine("Global Resource Library...");
			foreach (var resource in workspace.GlobalLibrary.Resources.Items())
			{
				Console.WriteLine($"{resource.Name} - {resource.Description}
({resource.Type})");
			}
		}
		finally
		{
			workspace.GlobalLibrary.Resources.Close();
		}
		Console.WriteLine("Closing Workspace ...");
	}
	Console.WriteLine("Workspace Closed ...");
	Console.Read();
}
catch (Exception ex)
{
	Console.WriteLine("Error:");
	Console.WriteLine(ex.Message);
	Console.WriteLine(ex.StackTrace);
	Console.Read();
}

static bool GetLoginInfo(LogonInfo info)
```

```
{
      info.UserName = "sysadmin";
      info.Password = "sysadmin";
      return true;
}

static bool GetRegistration(RegistrationInformation registrationInfo)
{
      registrationInfo.Key = "Your ProPricer API Registration Key";
      if (!string.IsNullOrEmpty(registrationInfo.LastError))
            Console.WriteLine(registrationInfo.LastError);

      return true;
}

static bool GetActivation(ProPricerActivation activation, ref bool changeKey)
{
      try
      {
            changeKey = true;
            if (!activation.Status.IsActivated())
            {
                  Console.WriteLine("Activating key online...");
                  activation.Activate();        }
      }
      catch (Exception e)
      {
            Console.WriteLine(e.Message);
            Console.WriteLine(e.StackTrace);
      }
      return true;
}
```

How to access the Travel Library

1. Add the following code to retrieve the travel items from the library.

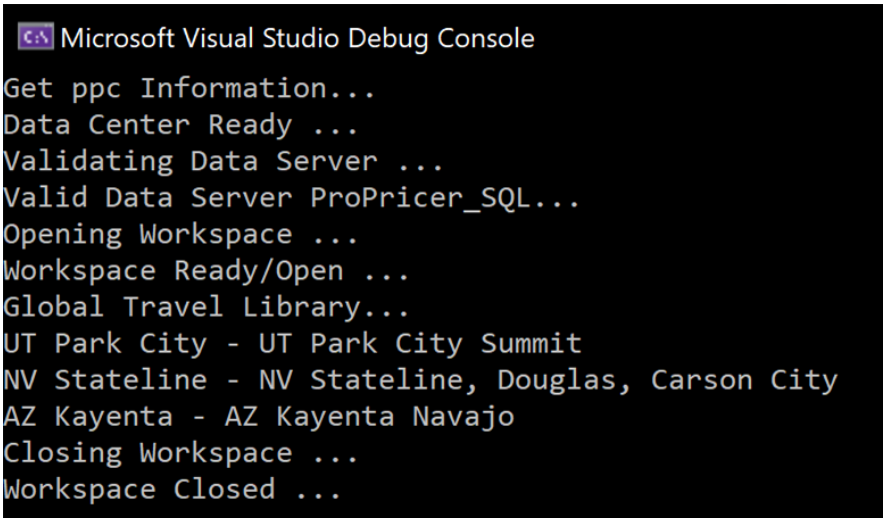
```
workspace.GlobalLibrary.Travels.Open();
try
{
    Console.WriteLine("Global Travel Library...");
    foreach (var travel in workspace.GlobalLibrary.Travels.Items())
    {
        Console.WriteLine($"{travel.Name} - {travel.Description}");
    }
}
finally
{
    workspace.GlobalLibrary.Travels.Close();
}
```

Code comments:

- The workspace.GlobalLibrary.Travels.Open() method loads all resources in the global library from the server to the client and the try finally ensures the workspace.GlobalLibrary.Travels.Close() is called.
- The workspace.GlobalLibrary.Travels.Items() collection in the foreach statement is accessing all the items within the Travel Library. Since this is an enumerable list, you can loop through all travel items.

2. Click Start to run the solution.

3. Once the solution connects, the console will be populated with all travel items from the Travel Library.



How to access the Resource Field Library

1. Add the following code to retrieve the resource field definitions from the Resource Fields Library.

```
workspace.GlobalLibrary.ResourceFieldDefinitions.Open();
try
{
    Console.WriteLine("Global Resource Fields Library...");
    foreach (var definition in workspace.GlobalLibrary.ResourceFieldDefinitions.Items())
    {
        Console.WriteLine($"{definition.Name} ({definition.DataType})");
    }
}
finally
{
    workspace.GlobalLibrary.ResourceFieldDefinitions.Close();
}
```

Code comments:

- The workspace.GlobalLibrary.ResourceFieldDefinitions collection in the foreach statement accesses all the resource field definitions within the Resource Field Library. You can gather the list of names, data types, and maximum length (for Text data type) or number of decimals (for Numeric data type).

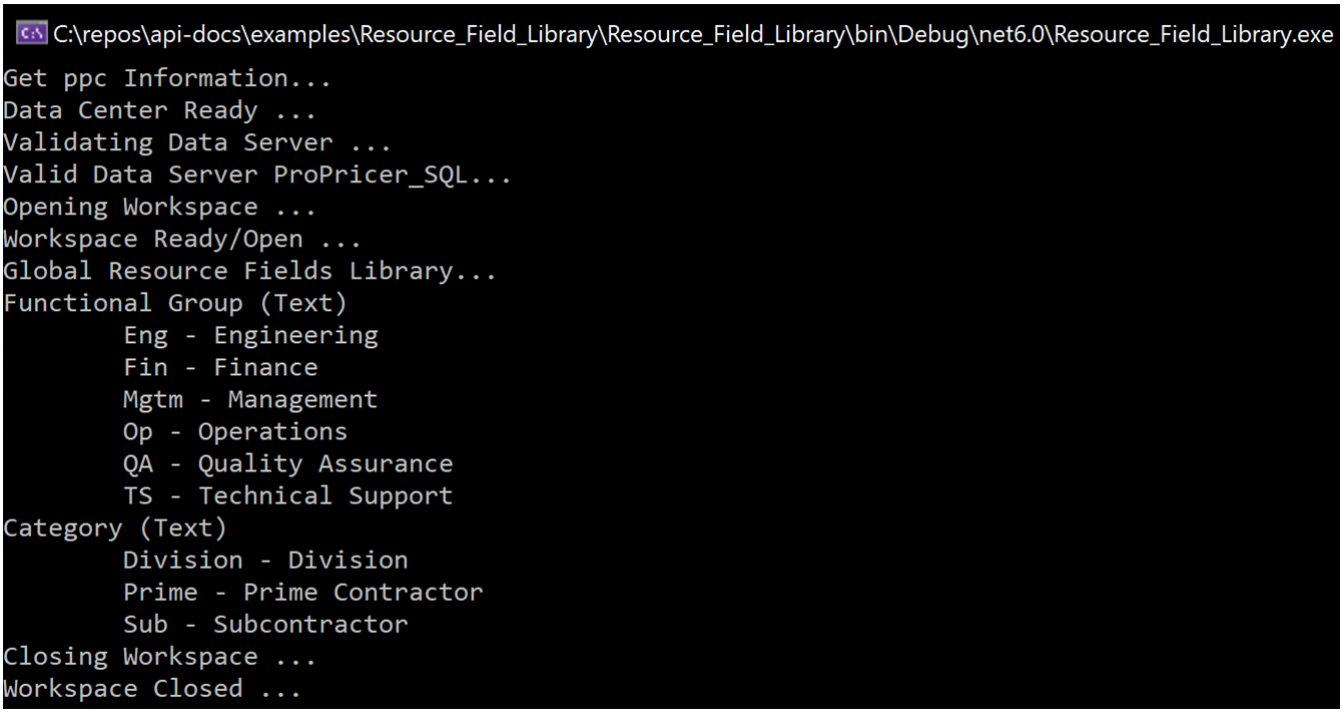
2. Add the following code to retrieve the value list for each Resource Field.

```
definition.ValueList.Open();
try
{
    foreach (var value in definition.ValueList.Items())
    {
        Console.WriteLine($"\t{value.Value} - {value.Description}");
    }
}
finally
{
    definition.ValueList.Close();
}
```

Code comments:

- Each Resource Field Definition has an associated value list that is presented to the user within ProPricer. After retrieving the field, you can then retrieve the value list.
    - First, show all Resource Field definitions in workspace.GlobalLibrary.ResourceFieldDefinitions collection. The .ValueList property provides access to the values.
    - The second loop iterates through the value list associated with the definition and returns the value and its description.

3. Click **Start** to run the solution.
4. Once the solution connects, the console will be populated with a list of resource fields and their values from the Resource Field Library.



How to add a new resource to the Resource Library

1. After opening Workspace and after displaying all resources in the global library, add the following code to prompt for the name and description of the new resource from the console.

```
Console.WriteLine();
Console.Write("Enter New Resource Name: ");
var name = Console.ReadLine();
Console.Write("Enter New Resource Description: ");
var description = Console.ReadLine();
```

2. Add the following code to add the resource to the Resource Library.

```
var newResource = workspace.GlobalLibrary.Resources.AddNew();
try
{
    newResource.Name = name;
    newResource.Description = description;
    newResource.EndEdit();
}
catch
{
    newResource.CancelEdit();
    throw;
}
```

Code comments:

- The workspace.GlobalLibrary.Resources.AddNew() method begins the add process by assigning the addition to the newResource variable. Any additions will begin with the AddNew() method.
- Some properties have default values, for example newResource.Type is automatically set to ResourceType.Labor. You do not need to set it to create a Labor resource.
- After the AddNew() method is invoked and the parameters and definitions are assigned, the EndEdit() must be invoked to release the record locking and commit the record, or CancelEdit() if there were any errors.
- Ensure the new item is saved with EndEdit() or disposed with CancelEdit following this try { } catch { } pattern for all new item creations:

```
var newItem = Collection.AddNew();
try
{
   //set newItem properties.
   newItem.EndEdit();
}
catch
{
   newItem.CancelEdit();
   throw;
}
```

3. To have better error messages ProPricer API provides the BrokenRules property. Add the following code to ensure you display any errors occurred during the EndEdit().

```
catch (Exception ex)
{
    Console.WriteLine("Error creating Resource: ");
    if (newResource.BrokenRules.Count > 0)
    {
        foreach (var brokenRule in newResource.BrokenRules)
        {
            Console.WriteLine(brokenRule.ErrorMessage);
        }
    }
    else
    {
        Console.WriteLine(ex.Message);
    }
    newResource.CancelEdit();
}
```

Code comments:

- BrokenRules are ProPricer specific error messages that are returned and are easier to diagnose than generic exceptions. The error messages are specific to ProPricer and the operation you are attempting.

4. Click **Start** to run the solution.

5. Once the solution connects, it will display existing resources. Input a value for the Resource Name and Description. The resource will be added.

Complete solution

```
using EBS.ProPricer.Client;
using EBS.ProPricer.Data;
using EBS.ProPricer.Model;
using EBS.ProPricer.Model.General;
using EBS.ProPricer.Registration;

try
{
	Console.WriteLine("Get ppc Information...");

	var conPP = new EBS.ProPricer.Configuration.Client.ClientConfiguration();
	conPP.LoadConfiguration(@"Path to your PPC file");

	var dataCenter = await DataCenter.OpenAsync(conPP.ServerName, conPP.ServerPort);

	Console.WriteLine("Data Center Ready ...");
	var dataServer = dataCenter.DataServers.Find(conPP.ConnectionName);
	Console.WriteLine("Validating Data Server ...");
	if (dataServer == null)
	{
		Console.WriteLine("Invalid Data Server");
		return;
	}
	Console.WriteLine($"Valid Data Server {dataServer.Info.Name}...");
	Console.WriteLine("Opening Workspace ...");
	using (var workspace = await dataServer.OpenWorkspaceAsync(GetLoginInfo,
GetRegistration, GetActivation))
	{
		Console.WriteLine("Workspace Ready/Open ...");
		workspace.GlobalLibrary.Resources.Open();
		try
		{
			Console.WriteLine("Global Resource Library...");
			foreach (var resource in workspace.GlobalLibrary.Resources.Items())
			{
				Console.WriteLine($"{resource.Name} - {resource.Description}
({resource.Type})");
			}

			Console.WriteLine();
			Console.Write("Enter New Resource Name: ");
			var name = Console.ReadLine();
			Console.Write("Enter New Resource Description: ");
			var description = Console.ReadLine();

			var newResource = workspace.GlobalLibrary.Resources.AddNew();
			try
			{
				newResource.Name = name;
				newResource.Description = description;
				newResource.EndEdit();
				Console.WriteLine("Resource successfully created.");
			}
			catch (Exception ex)
			{
				Console.WriteLine("Error creating Resource: ");
				if (newResource.BrokenRules.Count > 0)
```

```
                        {
                              foreach (var brokenRule in newResource.BrokenRules)
                              {
                                    Console.WriteLine(brokenRule.ErrorMessage);
                              }
                        }
                        else
                        {
                              Console.WriteLine(ex.Message);
                        }
                        newResource.CancelEdit();
                  }
            }
            finally
            {
                  workspace.GlobalLibrary.Resources.Close();
            }

            Console.WriteLine("Closing Workspace ...");
      }
      Console.WriteLine("Workspace Closed ...");
      Console.Read();
}
catch (Exception ex)
{
      Console.WriteLine("Error:");
      Console.WriteLine(ex.Message);
      Console.WriteLine(ex.StackTrace);
      Console.Read();
}

static bool GetLoginInfo(LogonInfo info)
{
      info.UserName = "sysadmin";
      info.Password = "sysadmin";
      return true;
}

static bool GetRegistration(RegistrationInformation registrationInfo)
{
      registrationInfo.Key = "Your ProPricer API Registration Key";
      if (!string.IsNullOrEmpty(registrationInfo.LastError))
            Console.WriteLine(registrationInfo.LastError);

      return true;
}

static bool GetActivation(ProPricerActivation activation, ref bool changeKey)
{
      try
      {
            changeKey = true;
            if (!activation.Status.IsActivated())
            {
                  Console.WriteLine("Activating key online...");
                  activation.Activate();
            }
      }
      catch (Exception e)
```

```
        {
                Console.WriteLine(e.Message);
                Console.WriteLine(e.StackTrace);
        }
        return true;
}
```

Rate tables

Rate tables can contain some of the global library information with additional details needed in the proposal. The tables must also be accessed through the workspace. The API has access to all rate tables and the associated details.

The rate tables contain the core data needed for all related proposals. These rates tables are used within proposals to build up the proposal's cost values.

Establishing a connection to these rate tables allows you to read, edit, add, and delete important rate information within ProPricer. These rates directly affect proposals that are connected to the table. With the ability in ProPricer to automatically price a proposal, most changes will immediately affect the proposal.

How to access an indirect rate table

1. Open the console app from previous examples in this guide.
2. Once the Workspace is opened, enter the following code to list all Indirect Rate Tables in your global library:

```
try
{
    Console.WriteLine("Indirect Rate Tables...");
    foreach (var table in workspace.GlobalLibrary.BurdenRateTables.Items())
    {
        Console.WriteLine($"{table.Name} - {table.Description}");
    }
}
finally
{
    workspace.GlobalLibrary.BurdenRateTables.Close();
```

Code comments:

- Indirect Rate Table is named Burden Rate Table in the API.
- The first step in accessing the table is using Open() to open the collection.
    - Workspace.GlobalLibrary.BurdenRateTables.Open()
- Next, you can access all tables in the collection using the .Items() in foreach loop.
- Print the Name and Description properties to the console.
- Close the library by calling the Close() method in a finally clause.

3. Insert the following code to prompt the user for the name of the Indirect Rate Table desired, use the Find() method to search for it, then display the Indirect Elements:

```
Console.WriteLine();
Console.Write("Enter the Indirect Rate Table name: ");
var name = Console.ReadLine();

var indirectTable = workspace.GlobalLibrary.BurdenRateTables.Find(name);
if (indirectTable == null)
{
    Console.WriteLine($"Indirect Rate Table '{name}' not found.");
    return;
}

indirectTable.Open();
try
{
    Console.WriteLine($"Indirect Pools in {indirectTable.Name}...");
    indirectTable.Elements.Open();
    try
    {
        foreach (var element in indirectTable.Elements.Items())
        {
            Console.WriteLine($"{element.BurdenPool.Name} - {element.BurdenPool.Description}");
        }
    }
    finally
    {
        indirectTable.Elements.Close();
    }
}
finally
{
    indirectTable.Close();
}
```

Code comments:

- Indirect Pool is named Burden Pool in the API.
- The first step in accessing the rates within the table is to find the table and use the Open() method.
    - var indirectTable = …
    - indirectTable.Open();
- Next, iterate through the Indirect Elements using Elements.Items() in a foreach loop.

4. Insert the following code to prompt the user for the name of the Indirect Pool desired, use the Find() method to search for it, then display the Rates:

```
Console.WriteLine();
Console.Write("Enter the Indirect Pool name: ");
var indirectPoolName = Console.ReadLine();
var indirectPool = workspace.GlobalLibrary.BurdenPools.Find(indirectPoolName);
if (indirectPool == null)
{
    Console.WriteLine($"Indirect Pool '{indirectPoolName}' not found in the library.");
    return;
}
var indirectElement = indirectTable.Elements.Find(indirectPool);
if (indirectElement == null)
{
    Console.WriteLine($"Indirect Pool '{indirectPoolName}' not found in table '{indirectTable.Name}'.");
    return;
}

indirectElement.Open();
try
{
    foreach (var rate in indirectElement.Rates)
    {
        Console.WriteLine($"{rate.Key.Element.Name} - {rate.Key.Period}: {rate.Value}");
    }
}
finally
{
    indirectElement.Close();
}
```

Code comments:

- First, use Find to obtain the Indirect Pool from the library.
    - var indirectPool = …
- Then use the Elements.Find(indirectPool) to find the indirect element for that indirect pool, then invoke the Open() method on the element.
    - var indirectElement =
    - indirectElement.Open();
- After the element is open, iterate through the rates dictionary. The Key contains the Indirect Cost Element (from the algorithm) and Period. The Period is the time period and will adjust if the rate table is created using monthly, quarterly, or annual rates. The value is the rate added within the pool.

5. Click **Start** to run the solution.

```
C:\repos\api-docs\examples\Indirect_Rate_Tables\Indirect_Rate_Tables\bin\Debug\net6.0\Indirect_Rate_Tables.exe
Get ppc Information...
Data Center Ready ...
Validating Data Server ...
Valid Data Server ProPricer_SQL...
Opening Workspace ...
Workspace Ready/Open ...
Indirect Rate Tables...
Tank Burdens - Advanced Tank Burden Rates

Enter the Indirect Rate Table name: Tank Burdens
Indirect Pools in Tank Burdens...
Eng Labor - Engineering Labor
Subcontracts - Subcontracts
ODCs - Other Direct Costs
Mnfg Labor - Manufacturing Labor
Materials - Materials

Enter the Indirect Pool name: Eng Labor
Labor Ovhd - 2020: 0.51
G&A IRD BP - 2020: 0.08
Profit - 2020: 0.1
G&A COM - 2020: 0.003
Labor Ovhd - 2021: 0.511
G&A IRD BP - 2021: 0.08
Profit - 2021: 0.1
G&A COM - 2021: 0.003
Labor Ovhd - 2022: 0.512
G&A IRD BP - 2022: 0.08
Profit - 2022: 0.1
G&A COM - 2022: 0.003
Labor Ovhd - 2023: 0.513
G&A IRD BP - 2023: 0.08
Profit - 2023: 0.1
G&A COM - 2023: 0.003
Labor Ovhd - 2024: 0.514
G&A IRD BP - 2024: 0.08
Profit - 2024: 0.1
G&A COM - 2024: 0.003
Labor Ovhd - 2025: 0.515
G&A IRD BP - 2025: 0.08
Profit - 2025: 0.1
G&A COM - 2025: 0.003
Closing Workspace ...
Workspace Closed ...
```

How to add a rate to an indirect element

1. Add the following code to prompt for the cost element, year, and rate to be added, and add the rate to the selected indirect element.

```
Console.WriteLine();
Console.Write($"Enter the Cost Element: ");
var costElementName = Console.ReadLine();
var costElement = indirectTable.Algorithm.Elements.FindName(costElementName);
if (costElement == null)
{
    Console.WriteLine($"Cost Element {costElementName} not found in the algorithm.");
    return;
}
Console.Write($"Enter the Year: ");
var yearText = Console.ReadLine();
int year;
if (!int.TryParse(yearText, out year))
{
    Console.WriteLine("Year must be an integer.");
    return;
}
Console.Write($"Enter the Rate: ");
var rateText = Console.ReadLine();
double rateToAdd;
if (!double.TryParse(rateText, out rateToAdd))
{
    Console.WriteLine("Rate must be a double number.");
    return;
}
var period = new TimeFrame(new DateTime(year, 1, 1));

indirectElement.BeginEdit();
try
{
    indirectElement.Rates[period, costElement] = rateToAdd;
    indirectElement.EndEdit();
    Console.WriteLine("Rate added.");
}
catch (Exception ex)
{
    Console.WriteLine("Error adding rate: ");
    if (indirectElement.BrokenRules.Count > 0)
    {
        foreach (var brokenRule in indirectElement.BrokenRules)
            Console.WriteLine(brokenRule.ErrorMessage);
    }
    else
        Console.WriteLine(ex.Message);
    indirectElement.CancelEdit();
}
```

Code comments:

- Since you are adding or editing the rate of an indirect element, the BeginEdit() and EndEdit() methods must be used.
- If there is an error, the CancelEdit() method should be invoked.
- The costElement variable finds the name of the BurdenCostElement based on the text entered in the Algorithm associated with the Indirect Rate Table.
- The indirectElement.Rates[TimeFrame period, BurdenCostElement element] specifies the year and cost element where the associated rate should be assigned.

2. Click **Start** to run the solution.

```
C:\repos\api-docs\examples\Indirect_Rate_Tables\Indirect_Rate_Tables\bin\Debug\net6.0\Indirect_Rate_Tables.exe
Get ppc Information...
Data Center Ready ...
Validating Data Server ...
Valid Data Server ProPricer_SQL...
Opening Workspace ...
Workspace Ready/Open ...
Indirect Rate Tables...
Tank Burdens - Advanced Tank Burden Rates

Enter the Indirect Rate Table name: Tank Burdens
Indirect Pools in Tank Burdens...
Eng Labor - Engineering Labor
Subcontracts - Subcontracts
ODCs - Other Direct Costs
Mnfg Labor - Manufacturing Labor
Materials - Materials

Enter the Indirect Pool name: Eng Labor
Labor Ovhd - 2020: 0.51
G&A IRD BP - 2020: 0.08
Profit - 2020: 0.1
G&A COM - 2020: 0.003
Labor Ovhd - 2021: 0.511
G&A IRD BP - 2021: 0.08
Profit - 2021: 0.1
G&A COM - 2021: 0.003
Labor Ovhd - 2022: 0.512
G&A IRD BP - 2022: 0.08
Profit - 2022: 0.1
G&A COM - 2022: 0.003
Labor Ovhd - 2023: 0.513
G&A IRD BP - 2023: 0.08
Profit - 2023: 0.1
G&A COM - 2023: 0.003
Labor Ovhd - 2024: 0.514
G&A IRD BP - 2024: 0.08
Profit - 2024: 0.1
G&A COM - 2024: 0.003
Labor Ovhd - 2025: 0.515
G&A IRD BP - 2025: 0.08
Profit - 2025: 0.1
G&A COM - 2025: 0.003

Enter the Cost Element: Labor Ovhd
Enter the Year: 2026
Enter the Rate: 0.516
Rate added.
Closing Workspace ...
Workspace Closed ...
```

Complete solution

```
using EBS.ProPricer.Client;
using EBS.ProPricer.Data;
using EBS.ProPricer.Model;
using EBS.ProPricer.Model.General;
using EBS.ProPricer.Registration;

try
{
	Console.WriteLine("Get ppc Information...");
	var conPP = new EBS.ProPricer.Configuration.Client.ClientConfiguration();
	conPP.LoadConfiguration(@"Path to your PPC file");

	var dataCenter = await DataCenter.OpenAsync(conPP.ServerName, conPP.ServerPort);

	Console.WriteLine("Data Center Ready ...");
	var dataServer = dataCenter.DataServers.Find(conPP.ConnectionName);
	Console.WriteLine("Validating Data Server ...");
	if (dataServer == null)
	{
		Console.WriteLine("Invalid Data Server");
		return;
	}
	Console.WriteLine($"Valid Data Server {dataServer.Info.Name}...");
	Console.WriteLine("Opening Workspace ...");
	using (var workspace = await dataServer.OpenWorkspaceAsync(GetLoginInfo,
GetRegistration, GetActivation))
	{
		Console.WriteLine("Workspace Ready/Open ...");
		workspace.GlobalLibrary.BurdenRateTables.Open();
		try
		{
			Console.WriteLine("Indirect Rate Tables...");
			foreach (var table in workspace.GlobalLibrary.BurdenRateTables.Items())
			{
				Console.WriteLine($"{table.Name} - {table.Description}");
			}

			Console.WriteLine();
			Console.Write("Enter the Indirect Rate Table name: ");
			var name = Console.ReadLine();

			var indirectTable =
workspace.GlobalLibrary.BurdenRateTables.Find(name);
			if (indirectTable == null)
			{
				Console.WriteLine($"Indirect Rate Table '{name}' not found.");
				return;
			}

			indirectTable.Open();
			try
			{
				Console.WriteLine($"Indirect Pools in {indirectTable.Name}...");
				indirectTable.Elements.Open();
				try
				{
					foreach (var element in indirectTable.Elements.Items())
```

```
                                {
                                      Console.WriteLine($"{element.BurdenPool.Name} -
{element.BurdenPool.Description}");
                                }

                                Console.WriteLine();
                                Console.Write("Enter the Indirect Pool name: ");
                                var indirectPoolName = Console.ReadLine();
                                var indirectPool =
workspace.GlobalLibrary.BurdenPools.Find(indirectPoolName);
                                if (indirectPool == null)
                                {
                                      Console.WriteLine($"Indirect Pool
'{indirectPoolName}' not found in the library.");
                                      return;
                                }
                                var indirectElement =
indirectTable.Elements.Find(indirectPool);
                                if (indirectElement == null)
                                {
                                      Console.WriteLine($"Indirect Pool
'{indirectPoolName}' not found in table '{indirectTable.Name}'.");
                                      return;
                                }

                                indirectElement.Open();
                                try
                                {
                                      foreach (var rate in indirectElement.Rates)
                                      {
                                            Console.WriteLine($"{rate.Key.Element.Name} -
{rate.Key.Period}: {rate.Value}");
                                      }

                                      Console.WriteLine();
                                      Console.Write($"Enter the Cost Element: ");
                                      var costElementName = Console.ReadLine();
                                      var costElement =
indirectTable.Algorithm.Elements.FindName(costElementName);
                                      if (costElement == null)
                                      {
                                            Console.WriteLine($"Cost Element
{costElementName} not found in the algorithm.");
                                            return;
                                      }
                                      Console.Write($"Enter the Year: ");
                                      var yearText = Console.ReadLine();
                                      int year;
                                      if (!int.TryParse(yearText, out year))
                                      {
                                            Console.WriteLine("Year must be an integer.");
                                            return;
                                      }
                                      Console.Write($"Enter the Rate: ");
                                      var rateText = Console.ReadLine();
                                      double rateToAdd;
                                      if (!double.TryParse(rateText, out rateToAdd))
                                      {
```

```
                                    Console.WriteLine("Rate must be a double
number.");
                                    return;
                                }
                                var period = new TimeFrame(new DateTime(year, 1, 1));

                                indirectElement.BeginEdit();
                                try
                                {
                                    indirectElement.Rates[period, costElement] =
rateToAdd;
                                    indirectElement.EndEdit();
                                    Console.WriteLine("Rate added.");
                                }
                                catch (Exception ex)
                                {
                                    Console.WriteLine("Error adding rate: ");
                                    if (indirectElement.BrokenRules.Count > 0)
                                    {
                                        foreach (var brokenRule in
indirectElement.BrokenRules)
                                        {

        Console.WriteLine(brokenRule.ErrorMessage);
                                        }
                                    }
                                    else
                                    {
                                        Console.WriteLine(ex.Message);
                                    }
                                    indirectElement.CancelEdit();
                                }
                            }
                            finally
                            {
                                indirectElement.Close();
                            }
                        }
                        finally
                        {
                            indirectTable.Elements.Close();
                        }
                    }
                    finally
                    {
                        indirectTable.Close();
                    }
                }
                finally
                {
                    workspace.GlobalLibrary.BurdenRateTables.Close();
                }

                Console.WriteLine("Closing Workspace ...");
            }
            Console.WriteLine("Workspace Closed ...");
            Console.Read();
        }
        catch (Exception ex)
```

```
{
      Console.WriteLine("Error:");
      Console.WriteLine(ex.Message);
      Console.WriteLine(ex.StackTrace);
      Console.Read();
}

static bool GetLoginInfo(LogonInfo info)
{
      info.UserName = "sysadmin";
      info.Password = "sysadmin";
      return true;
}

static bool GetRegistration(RegistrationInformation registrationInfo)
{
      registrationInfo.Key = "Your ProPricer API Registration Key";
      if (!string.IsNullOrEmpty(registrationInfo.LastError))
            Console.WriteLine(registrationInfo.LastError);

      return true;
}

static bool GetActivation(ProPricerActivation activation, ref bool changeKey)
{
      try
      {
            changeKey = true;
            if (!activation.Status.IsActivated())
            {
                  Console.WriteLine("Activating key online...");
                  activation.Activate();
            }
      }
      catch (Exception e)
      {
            Console.WriteLine(e.Message);
            Console.WriteLine(e.StackTrace);
      }
      return true;
}
```

Proposals

Proposals stores the tasks, resources, hours, and cost information. They also use a majority of the library items as well as the rate tables. The proposals can be accessed through the workspace entity, and have additional methods available for capturing data.

The proposal contains the core data needed for pricing. The proposal is organized through unique tasks which contain resources. Resources contain hours, units, and cost estimates.

Establishing a connection to the proposal allows you to read, edit, add, and delete important task and resource information. All data within the proposal are exposed to the API. The API is also required to access any calculated data since it is not stored within any database tables.

How to access the proposal properties and task list

1. Open the Connection_Test_PPC console app from the previous example in this guide.
2. Once the Workspace is opened, notice the code to list all proposals.

```
workspace.Proposals.Open();
try
{
    Console.WriteLine("Proposal List...");
    foreach (var proposal in workspace.Proposals.Items())
    {
        Console.WriteLine($"{proposal.Name}, Ver. {proposal.Version}");
    }
}
finally
{
    workspace.Proposals.Close();
}
```

3. Enter the following code to prompt the user for a proposal name and version, then open the proposal and list all tasks.

```
Console.WriteLine();
Console.Write("Enter the Proposal name: ");
var name = Console.ReadLine();
Console.Write("Enter the Proposal version: ");
var ver = Console.ReadLine();

var myProposal = workspace.Proposals.Find(name, ver);
if (myProposal == null)
{
    Console.WriteLine("Proposal not found");
    return;
}
myProposal.Open();
try
{
    Console.WriteLine($"Proposal '{myProposal.Name}, Ver. {myProposal.Version}' found.");
    Console.WriteLine($"\tDescription: {myProposal.Description}");
    Console.WriteLine($"\tCreator: {myProposal.Creator?.Name}");
    Console.WriteLine($"\tCreation Date: {myProposal.CreationDate}");
    Console.WriteLine($"\tDirect Rate Table: {myProposal.DirectRateTable?.Name}");
    Console.WriteLine($"\tIndirect Rate Table: {myProposal.ActualBurdenRateTable?.Name}");
    Console.WriteLine($"\tPOP: [{myProposal.StartDate} - {myProposal.EndDate}]");
    Console.WriteLine($"Tasks:");
    foreach (var task in myProposal.Tasks.Items())
    {
        Console.WriteLine($"\t{task.Name} - {task.Description} [{task.StartDate} - {task.EndDate}]");
    }
}
finally
{
    myProposal.Close();
}
```

Code comments:

- The myProposal variable is assigned using the Find(string name, string version) method. Once selected, the Open() method is used to access any proposal related data.
- Once opened, you can access a variety of general proposal data properties.
    - Null-conditional operator is used because Creator can be null when the user that created the proposal has been deleted.
    - Null-conditional operator is used because Direct Rate Table can be null when the proposal doesn't have one assigned.
    - The property ActualBurdenRateTable provides the reference to the Indirect Rate Table used by the proposal. An example is when the Indirect Rate Table associated with the Direct Rate Table has been overridden in the proposal.
- The myProposal.Tasks.Items() collection can be accessed to iterate through the tasks and gather information.

4. Click **Start** to run the solution.

```
C:\repos\api-docs\examples\Proposal\Proposal\bin\Debug\net6.0\Proposal.exe
Get ppc Information...
Data Center Ready ...
Validating Data Server ...
Valid Data Server ProPricer_SQL...
Opening Workspace ...
Workspace Ready/Open ...
Proposal List...
Advanced Tank, Ver. 0
Tank Materials, Ver. 0
Tank Scheduled Rates, Ver. 0
multi currency, Ver. 0
PROPOSAL, Ver. 0

Enter the Proposal name: advanced tank
Enter the Proposal version: 0
Proposal 'Advanced Tank, Ver. 0' found.
        Description: Advanced Tank
        Creator: System Administrator
        Creation Date: 3/10/2023 7:01:29 PM
        Direct Rate Table: Tank Directs
        Indirect Rate Table: Tank Burdens
        POP: [01/2020 - 03/2022]
Tasks:
        TANK 001 01 - Program Management [01/2020 - 03/2022]
        TANK 002 01 - Software Planning [01/2020 - 03/2020]
        TANK 003 01 - Ergonomics [01/2021 - 03/2022]
        TANK 008 02 - Wind Tunnel Testing [01/2020 - 12/2020]
        TANK 008 03 - Sun Tunnel Testing [01/2021 - 12/2021]
        TANK 001 02 - Financial Mangement [01/2020 - 09/2021]
        TANK 001 03 - Planning & Scheduling [01/2020 - 03/2022]
        TANK 001 04 - Contracts & Pricing [04/2020 - 06/2020]
        TANK 001 TR - Program Management Travel [01/2020 - 03/2022]
        TANK 001 WR - Program Warranty [01/2020 - 03/2022]
        TANK 002 02 - SW Implementation, Test and Doc [04/2020 - 06/2020]
        TANK 002 03 - SW Deployment & Maintenance [10/2020 - 10/2021]
        TANK 004 01 - Prototype Destructive Test [07/2021 - 12/2021]
        TANK 004 02 - Tabletop Development [01/2020 - 03/2022]
        TANK 005 01 - Technical Publications [01/2020 - 03/2022]
        TANK 005 02 - Engineering Data [01/2020 - 03/2022]
        TANK 005 03 - Management Data [01/2020 - 03/2022]
        TANK 005 04 - Support Data [01/2020 - 03/2022]
        TANK 005 05 - User Manual [10/2021 - 03/2022]
        TANK 006 01 - Cardboard Mockup [01/2020 - 03/2022]
        TANK 006 02 - Clay Mockup [01/2020 - 12/2020]
        TANK 006 03 - Scale Mockup [01/2021 - 12/2021]
        TANK 007 01 - Destructive Mockup Test [01/2020 - 03/2022]
        TANK 008 01 - Accelerated Aging Test & Evaluation [01/2020 - 03/2022]
Closing Workspace ...
Workspace Closed ...
```

How to add a task to a proposal

1. Add the following code to prompt the user for a task name, then add it to the proposal.

```
Console.WriteLine();
Console.Write($"Enter the new {myProposal.TaskIdLabel}: ");
var taskName = Console.ReadLine();
if (!string.IsNullOrEmpty(taskName))
{
    var newTask = myProposal.Tasks.AddNew();
    try
    {
        newTask.Name = taskName;
        newTask.Description = "API added task";
        newTask.EndEdit();
        Console.WriteLine($"Task '{newTask.Name}' created successfully.");
        Console.WriteLine($"\tPOP: [{newTask.StartDate} - {newTask.EndDate}]");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error adding task: ");
        if (newTask.BrokenRules.Count > 0)
        {
            foreach (var brokenRule in newTask.BrokenRules)
            {
                Console.WriteLine(brokenRule.ErrorMessage);
            }
        }
        else
        {
            Console.WriteLine(ex.Message);
        }
        newTask.CancelEdit();
    }
}
```

Code comments:

- Use Tasks.AddNew() to add a task to the proposal, then set the properties.
- The minimum information required for the tasks is the Name.
- The description is set to a hardcoded value.
- If a start and end date are not provided, the dates from the proposal will be used. This example shows that you don’t need to assign the dates.
- Save the task with EndEdit(), then obtain the tasks dates assigned by default.
- If there are errors, call CancelEdit() to release locks.

2. Click **Start** to run the solution.

```
C:\repos\api-docs\examples\Proposal\Proposal\bin\Debug\net6.0\Proposal.exe
Get ppc Information...
Data Center Ready ...
Validating Data Server ...
Valid Data Server ProPricer_SQL...
Opening Workspace ...
Workspace Ready/Open ...
Proposal List...
Advanced Tank, Ver. 0
Tank Materials, Ver. 0
Tank Scheduled Rates, Ver. 0
multi currency, Ver. 0
PROPOSAL, Ver. 0

Enter the Proposal name: advanced tank
Enter the Proposal version: 0
Proposal 'Advanced Tank, Ver. 0' found.
        Description: Advanced Tank
        Creator: System Administrator
        Creation Date: 3/10/2023 7:01:29 PM
        Direct Rate Table: Tank Directs
        Indirect Rate Table: Tank Burdens
        POP: [01/2020 - 03/2022]
Tasks:
        TANK 001 01 - Program Management [01/2020 - 03/2022]
        TANK 002 01 - Software Planning [01/2020 - 03/2020]
        TANK 003 01 - Ergonomics [01/2021 - 03/2022]
        TANK 008 02 - Wind Tunnel Testing [01/2020 - 12/2020]
        TANK 008 03 - Sun Tunnel Testing [01/2021 - 12/2021]
        TANK 001 02 - Financial Mangement [01/2020 - 09/2021]
        TANK 001 03 - Planning & Scheduling [01/2020 - 03/2022]
        TANK 001 04 - Contracts & Pricing [04/2020 - 06/2020]
        TANK 001 TR - Program Management Travel [01/2020 - 03/2022]
        TANK 001 WR - Program Warranty [01/2020 - 03/2022]
        TANK 002 02 - SW Implementation, Test and Doc [04/2020 - 06/2020]
        TANK 002 03 - SW Deployment & Maintenance [10/2020 - 10/2021]
        TANK 004 01 - Prototype Destructive Test [07/2021 - 12/2021]
        TANK 004 02 - Tabletop Development [01/2020 - 03/2022]
        TANK 005 01 - Technical Publications [01/2020 - 03/2022]
        TANK 005 02 - Engineering Data [01/2020 - 03/2022]
        TANK 005 03 - Management Data [01/2020 - 03/2022]
        TANK 005 04 - Support Data [01/2020 - 03/2022]
        TANK 005 05 - User Manual [10/2021 - 03/2022]
        TANK 006 01 - Cardboard Mockup [01/2020 - 03/2022]
        TANK 006 02 - Clay Mockup [01/2020 - 12/2020]
        TANK 006 03 - Scale Mockup [01/2021 - 12/2021]
        TANK 007 01 - Destructive Mockup Test [01/2020 - 03/2022]
        TANK 008 01 - Accelerated Aging Test & Evaluation [01/2020 - 03/2022]

Enter the new TASK ID: NEW TASK
Task 'NEW TASK' created successfully.
        POP: [01/2020 - 03/2022]
Closing Workspace ...
Workspace Closed ...
```

Resources

Resources are assigned hour, unit, and cost information. They can be used in a variety of ways within the Resources, Activities, Travel, Material, CERs, Resource Sets, and Formulas tabs. The API can access all the related data at the spread amount or monthly level.

Individual resources (labor and non-labor) are assigned to various tasks depending on the role that they play in the accomplishment of the task.

Accurately assigning and spreading the hours or costs of a particular resource directly influences the grand total price of a proposal. Any hours, units, or costs input for a resource using the spread amount must be accessed through the API. All burden costs follow this rule as well since they are calculated fields.

How to access the resources in a proposal

1. Use the following code to prompt the user for a task name, open the task and show the direct resources in the task.

```
Console.WriteLine();
Console.Write($"Enter an existing {myProposal.TaskIdLabel}: ");
var myTaskName = Console.ReadLine();
if (!string.IsNullOrEmpty(myTaskName))
{
    var myTask = myProposal.Tasks.Find(myTaskName);
    if (myTask == null)
    {
        Console.WriteLine($"Task '{myTaskName}' not found.");
        return;
    }
    myTask.Open();
    try
    {
        var gtpPosition = myProposal.Algorithm.TotalPriceElement.Position;
        Console.WriteLine($"Direct resources in task '{myTask.Name}':");
        foreach (var resourceAssignment in myTask.ResourceAssignments.Items())
        {
            Console.WriteLine($"\t{resourceAssignment.Info.Resource.Name} - {resourceAssignment.Info.Resource.Description}");
            Console.Write($"\t\tTotal Estimate: {resourceAssignment.Spread.TotalAmount:n}");
            if (resourceAssignment.Spread.AmountType == ResourceAmountType.EquivalentHeads)
            {
                Console.Write($" Equivalent Heads.");
            }
            else
            {
                Console.Write($" {resourceAssignment.Info.RateType}.");
            }
            var resourceCost = resourceAssignment.GetCost();
            Console.Write($" Direct Cost: {resourceCost?.DirectCost ?? 0:C}.");
            Console.WriteLine($" Total Price: {resourceCost?.BurdenCost(gtpPosition) ?? 0:C}.");
        }
    }
    finally
    {
        myTask.Close();
    }
}
```

Code comments:

- The task.ResourceAssignments.Items() collection contains the information for all resources on all input tabs.
- There can be duplicate resources within a single task.
- The gtpPosition integer is assigned the TotalPriceElement.Position to retrieve the Grand Total Price by index number. Each algorithm has six common positions (TotalCAS, TotalCost, TotalFee, TotalG&A, TotalOverhead, and TotalPrice).
- While looping through the ResourceAssignments, we obtain total amount information and total direct cost and total price. You can also use GetCost() to obtain the cost information for a specific period.
- Null-conditional operator is used because zero values will not have cost information (GetCost() will return null).

2. Click **Start** to run the solution.

```
C:\repos\api-docs\examples\Proposal\Proposal\bin\Debug\net6.0\Proposal.exe
Proposal List...
Advanced Tank, Ver. 0
Tank Materials, Ver. 0
Tank Scheduled Rates, Ver. 0
multi currency, Ver. 0
PROPOSAL, Ver. 0

Enter the Proposal name: advanced tank
Enter the Proposal version: 0
Proposal 'Advanced Tank, Ver. 0' found.
        Description: Advanced Tank
        Creator: System Administrator
        Creation Date: 3/10/2023 7:01:29 PM
        Direct Rate Table: Tank Directs
        Indirect Rate Table: Tank Burdens
        POP: [01/2020 - 03/2022]
Tasks:
        TANK 001 01 - Program Management [01/2020 - 03/2022]
        TANK 002 01 - Software Planning [01/2020 - 03/2020]
        TANK 003 01 - Ergonomics [01/2021 - 03/2022]
        TANK 008 02 - Wind Tunnel Testing [01/2020 - 12/2020]
        TANK 008 03 - Sun Tunnel Testing [01/2021 - 12/2021]
        TANK 001 02 - Financial Mangement [01/2020 - 09/2021]
        TANK 001 03 - Planning & Scheduling [01/2020 - 03/2022]
        TANK 001 04 - Contracts & Pricing [04/2020 - 06/2020]
        TANK 001 TR - Program Management Travel [01/2020 - 03/2022]
        TANK 001 WR - Program Warranty [01/2020 - 03/2022]
        TANK 002 02 - SW Implementation, Test and Doc [04/2020 - 06/2020]
        TANK 002 03 - SW Deployment & Maintenance [10/2020 - 10/2021]
        TANK 004 01 - Prototype Destructive Test [07/2021 - 12/2021]
        TANK 004 02 - Tabletop Development [01/2020 - 03/2022]
        TANK 005 01 - Technical Publications [01/2020 - 03/2022]
        TANK 005 02 - Engineering Data [01/2020 - 03/2022]
        TANK 005 03 - Management Data [01/2020 - 03/2022]
        TANK 005 04 - Support Data [01/2020 - 03/2022]
        TANK 005 05 - User Manual [10/2021 - 03/2022]
        TANK 006 01 - Cardboard Mockup [01/2020 - 03/2022]
        TANK 006 02 - Clay Mockup [01/2020 - 12/2020]
        TANK 006 03 - Scale Mockup [01/2021 - 12/2021]
        TANK 007 01 - Destructive Mockup Test [01/2020 - 03/2022]
        TANK 008 01 - Accelerated Aging Test & Evaluation [01/2020 - 03/2022]
        NEW TASK - API added task [01/2020 - 03/2022]

Enter the new TASK ID:

Enter the existing TASK ID: tank 001 01
Resources in task 'TANK 001 01':
        Engineering Aide - Engineering Aide and Techs
                Total Estimate: 48,000.00 Hours. Direct Cost: $1,091,493.32 Total Price: $1,963,847.55
        IntDiv Xfr @ Cost - Interdivisional Transfer at Cost
                Total Estimate: 50,000.00 Cost. Direct Cost: $50,000.00 Total Price: $59,600.00
        IntDiv Xfr @ Price - Interdivisional Transfer at Price
                Total Estimate: 35,000.00 Cost. Direct Cost: $0.00 Total Price: $35,000.00
        Program Manager - Program Managers and Deputy PMs
                Total Estimate: 16,000.00 Hours. Direct Cost: $652,938.20 Total Price: $1,174,574.54
Closing Workspace ...
Workspace Closed ...
```

How to add a resource to the proposal's Resources tab

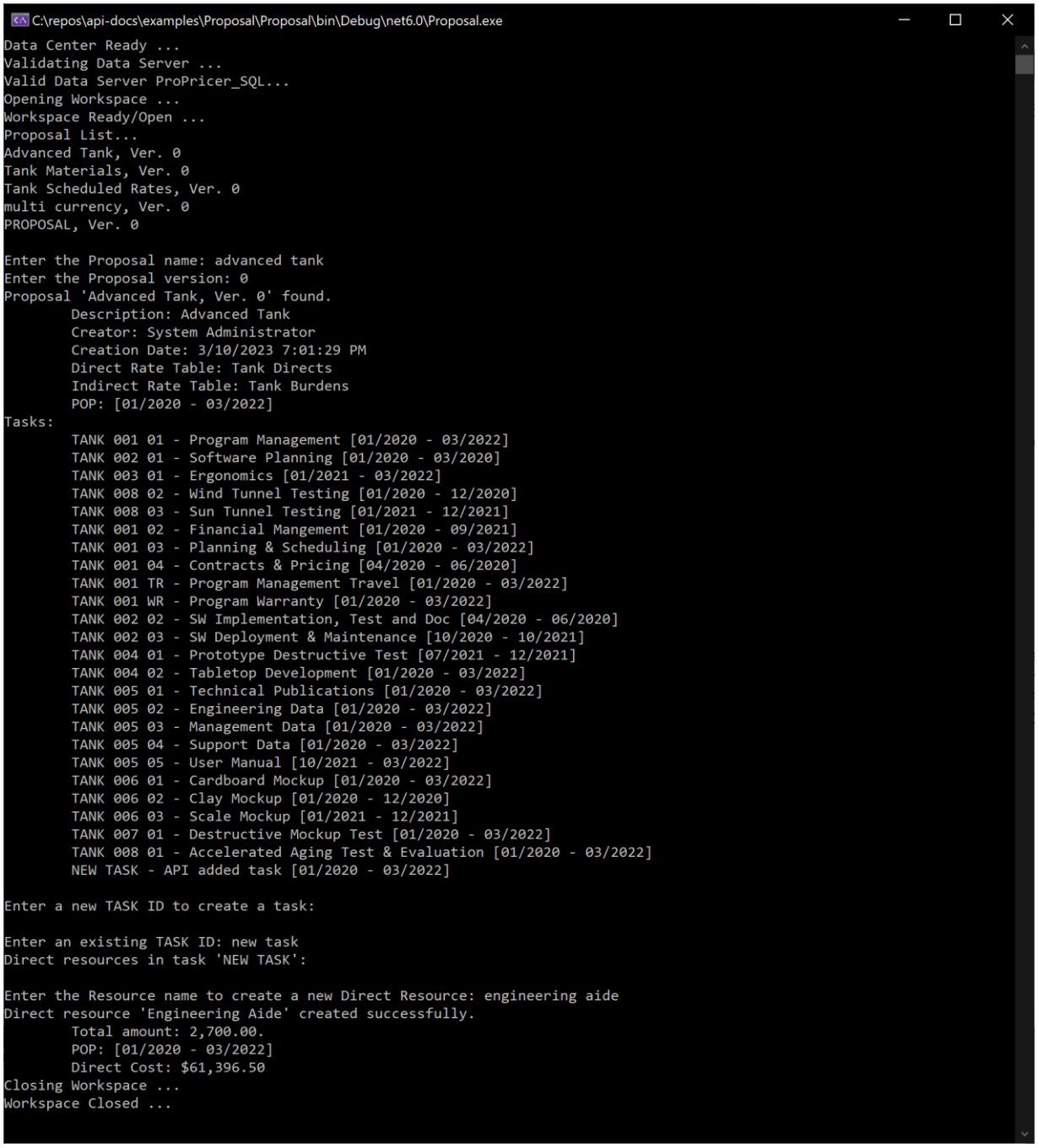
1. Add the following code to prompt the user for a resource name to be added as direct resource.

```
Console.WriteLine();
Console.Write($"Enter the Resource name to create a new Direct Resource: ");
var newResourceName = Console.ReadLine();
if (!string.IsNullOrEmpty(newResourceName))
{
    var resource = myProposal.Library.Resources.Find(newResourceName);
    if (resource == null)
    {
        Console.WriteLine($"'{newResourceName}' not found in the Resource Library.");
        return;
    }
    if (myProposal.DirectRateTable?.Elements?.Find(resource) == null)
    {
        Console.WriteLine($"'{newResourceName}' not found in the proposal's Direct Rate Table.");
        return;
    }
    var newResource = myTask.Resources.AddNew();
    try
    {
        newResource.Info.Resource = resource;
        newResource.Spread.Amount = 100;
        newResource.Spread.Method = SpreadMethod.Load;
        newResource.EndEdit();
        Console.WriteLine($"Direct resource '{newResource.Info.Resource.Name}' created successfully.");
        Console.WriteLine($"\tTotal amount: {newResource.Spread.TotalAmount:n}.");
        Console.WriteLine($"\tPOP: [{newResource.Spread.StartDate} - {newResource.Spread.EndDate}]");
        Console.WriteLine($"\tDirect Cost: {newResource.GetCost()?.DirectCost ?? 0:c}");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error creating the direct resource: ");
        if (newResource.BrokenRules.Count > 0)
        {
            foreach (var brokenRule in newResource.BrokenRules)
            {
                Console.WriteLine(brokenRule.ErrorMessage);
            }
        }
        else
        {
            Console.WriteLine(ex.Message);
        }
        newResource.CancelEdit();
    }
}
```

Code comments:

- The addition of the new resource is verified against the resource library and the assigned direct rate table. Only resources within the rate table can be added to a task.
- The only required element to add a resource is the Info.Resource.
- Hardcoded values are set for Amount, and Spread Method.
- Resource Field definitions could be added if required.
- After resource is created with EndEdit() you can access cost information.

2. Click **Start** to run the solution.



Complete solution

```
using EBS.ProPricer.Client;
using EBS.ProPricer.Data;
using EBS.ProPricer.Model;
using EBS.ProPricer.Model.General;
using EBS.ProPricer.Registration;

try
{
	Console.WriteLine("Get ppc Information...");

	var conPP = new EBS.ProPricer.Configuration.Client.ClientConfiguration();
	conPP.LoadConfiguration(@"Path to your PPC file");

	var dataCenter = await DataCenter.OpenAsync(conPP.ServerName, conPP.ServerPort);

	Console.WriteLine("Data Center Ready ...");
	var dataServer = dataCenter.DataServers.Find(conPP.ConnectionName);
	Console.WriteLine("Validating Data Server ...");
	if (dataServer == null)
	{
		Console.WriteLine("Invalid Data Server");
		return;
	}
	Console.WriteLine($"Valid Data Server {dataServer.Info.Name}...");
	Console.WriteLine("Opening Workspace ...");
	using (var workspace = await dataServer.OpenWorkspaceAsync(GetLoginInfo,
GetRegistration, GetActivation))
	{
		Console.WriteLine("Workspace Ready/Open ...");
		workspace.Proposals.Open();
		try
		{
			Console.WriteLine("Proposal List...");
			foreach (var proposal in workspace.Proposals.Items())
			{
				Console.WriteLine($"{proposal.Name}, Ver. {proposal.Version}");
			}

			Console.WriteLine();
			Console.Write("Enter the Proposal name: ");
			var name = Console.ReadLine();
			Console.Write("Enter the Proposal version: ");
			var ver = Console.ReadLine();

			var myProposal = workspace.Proposals.Find(name, ver);
			if (myProposal == null)
			{
				Console.WriteLine("Proposal not found");
				return;
			}
			myProposal.Open();
			try
			{
				Console.WriteLine($"Proposal '{myProposal.Name}, Ver.
{myProposal.Version}' found.");
				Console.WriteLine($"\tDescription: {myProposal.Description}");
				Console.WriteLine($"\tCreator: {myProposal.Creator?.Name}");
```

```
                        Console.WriteLine($"\tCreation Date: {myProposal.CreationDate}");
                        Console.WriteLine($"\tDirect Rate Table:
{myProposal.DirectRateTable?.Name}");
                        Console.WriteLine($"\tIndirect Rate Table:
{myProposal.ActualBurdenRateTable?.Name}");
                        Console.WriteLine($"\tPOP: [{myProposal.StartDate} -
{myProposal.EndDate}]");
                        Console.WriteLine($"Tasks:");
                        foreach (var task in myProposal.Tasks.Items())
                        {
                              Console.WriteLine($"\t{task.Name} - {task.Description}
[{task.StartDate} - {task.EndDate}]");
                        }

                        Console.WriteLine();
                        Console.Write($"Enter a new {myProposal.TaskIdLabel} to create a
task: ");
                        var taskName = Console.ReadLine();
                        if (!string.IsNullOrEmpty(taskName))
                        {
                              var newTask = myProposal.Tasks.AddNew();
                              try
                              {
                                    newTask.Name = taskName;
                                    newTask.Description = "API added task";
                                    newTask.EndEdit();
                                    Console.WriteLine($"Task '{newTask.Name}' created
successfully.");
                                    Console.WriteLine($"\tPOP: [{newTask.StartDate} -
{newTask.EndDate}]");
                              }
                              catch (Exception ex)
                              {
                                    Console.WriteLine("Error adding task: ");
                                    if (newTask.BrokenRules.Count > 0)
                                    {
                                          foreach (var brokenRule in newTask.BrokenRules)
                                          {

      Console.WriteLine(brokenRule.ErrorMessage);
                                          }
                                    }
                                    else
                                    {
                                          Console.WriteLine(ex.Message);
                                    }
                                    newTask.CancelEdit();
                              }
                        }

                        Console.WriteLine();
                        Console.Write($"Enter an existing {myProposal.TaskIdLabel}: ");
                        var myTaskName = Console.ReadLine();
                        if (!string.IsNullOrEmpty(myTaskName))
                        {
                              var myTask = myProposal.Tasks.Find(myTaskName);
                              if (myTask == null)
                              {
                                    Console.WriteLine($"Task '{myTaskName}' not found.");
```

```
                                    return;
                                }
                                myTask.Open();
                                try
                                {
                                    var gtpPosition =
myProposal.Algorithm.TotalPriceElement.Position;
                                    Console.WriteLine($"Direct resources in task
'{myTask.Name}':");
                                    foreach (var resourceAssignment in
myTask.ResourceAssignments.Items())
                                    {

      Console.WriteLine($"\t{resourceAssignment.Info.Resource.Name} -
{resourceAssignment.Info.Resource.Description}");
                                        Console.Write($"\t\tTotal Estimate:
{resourceAssignment.Spread.TotalAmount:n}");
                                        if (resourceAssignment.Spread.AmountType ==
ResourceAmountType.EquivalentHeads)
                                        {
                                            Console.Write($" Equivalent Heads.");
                                        }
                                        else
                                        {
                                            Console.Write($"
{resourceAssignment.Info.RateType}.");
                                        }
                                        var resourceCost =
resourceAssignment.GetCost();
                                        Console.Write($" Direct Cost:
{resourceCost?.DirectCost ?? 0:C}.");
                                        Console.WriteLine($" Total Price:
{resourceCost?.BurdenCost(gtpPosition) ?? 0:C}.");
                                    }

                                    Console.WriteLine();
                                    Console.Write($"Enter the Resource name to create a
new Direct Resource: ");
                                    var newResourceName = Console.ReadLine();
                                    if (!string.IsNullOrEmpty(newResourceName))
                                    {
                                        var resource =
myProposal.Library.Resources.Find(newResourceName);
                                        if (resource == null)
                                        {
                                            Console.WriteLine($"'{newResourceName}'
not found in the Resource Library.");
                                            return;
                                        }
                                        if
(myProposal.DirectRateTable?.Elements?.Find(resource) == null)
                                        {
                                            Console.WriteLine($"'{newResourceName}'
not found in the proposal's Direct Rate Table.");
                                            return;
                                        }
                                        var newResource = myTask.Resources.AddNew();
                                        try
                                        {
```

```
                                                newResource.Info.Resource = resource;
                                                newResource.Spread.Amount = 100;
                                                newResource.Spread.Method =
SpreadMethod.Load;
                                                newResource.EndEdit();
                                                Console.WriteLine($"Direct resource
'{newResource.Info.Resource.Name}' created successfully.");
                                                Console.WriteLine($"\tTotal amount:
{newResource.Spread.TotalAmount:n}.");
                                                Console.WriteLine($"\tPOP:
[{newResource.Spread.StartDate} - {newResource.Spread.EndDate}]");
                                                Console.WriteLine($"\tDirect Cost:
{newResource.GetCost()?.DirectCost ?? 0:c}");
                                          }
                                          catch (Exception ex)
                                          {
                                                Console.WriteLine("Error creating the
direct resource: ");
                                                if (newResource.BrokenRules.Count > 0)
                                                {
                                                      foreach (var brokenRule in
newResource.BrokenRules)
                                                      {

      Console.WriteLine(brokenRule.ErrorMessage);
                                                      }
                                                }
                                                else
                                                {
                                                      Console.WriteLine(ex.Message);
                                                }
                                                newResource.CancelEdit();
                                          }
                                    }
                              }
                              finally
                              {
                                    myTask.Close();
                              }
                        }
                  }
                  finally
                  {
                        myProposal.Close();
                  }
            }
            finally
            {
                  workspace.Proposals.Close();
            }
            Console.WriteLine("Closing Workspace ...");
      }
      Console.WriteLine("Workspace Closed ...");
      Console.Read();
}
catch (Exception ex)
{
      Console.WriteLine("Error: ");
      Console.WriteLine(ex.Message);
```

```
        Console.WriteLine(ex.StackTrace);
        Console.Read();
}

static bool GetLoginInfo(LogonInfo info)
{
        info.UserName = "sysadmin";
        info.Password = "sysadmin";
        return true;
}

static bool GetRegistration(RegistrationInformation registrationInfo)
{
        registrationInfo.Key = "Your ProPricer API Registration Key";
        if (!string.IsNullOrEmpty(registrationInfo.LastError))
                Console.WriteLine(registrationInfo.LastError);

        return true;
}

static bool GetActivation(ProPricerActivation activation, ref bool changeKey)
{
        try
        {
                changeKey = true;
                if (!activation.Status.IsActivated())
                {
                        Console.WriteLine("Activating key online...");
                        activation.Activate();
                }
        }
        catch (Exception e)
        {
                Console.WriteLine(e.Message);
                Console.WriteLine(e.StackTrace);
        }
        return true;
}
```

Proposal operations

Proposal operations are simple functions that people will use on a daily basis. These functions include Add, Copy, and Delete. Adding, copying, and deleting proposals are common functions within the ProPricer client.

The operations are important since a programmer can automate these functions for maintenance or application interconnectivity.

How to add a new proposal

1. Open the Connection_Test_PPC console app from the previous example in this guide.
2. Once the Workspace is opened, find the code to list all proposals.

```
workspace.Proposals.Open();
try
{
    Console.WriteLine("Proposal List...");
    foreach (var proposal in workspace.Proposals.Items())
    {
        Console.WriteLine($"{proposal.Name}, Ver. {proposal.Version}");
    }
}
finally
{
    workspace.Proposals.Close();
}
```

3. After the preceding code, add the following code.

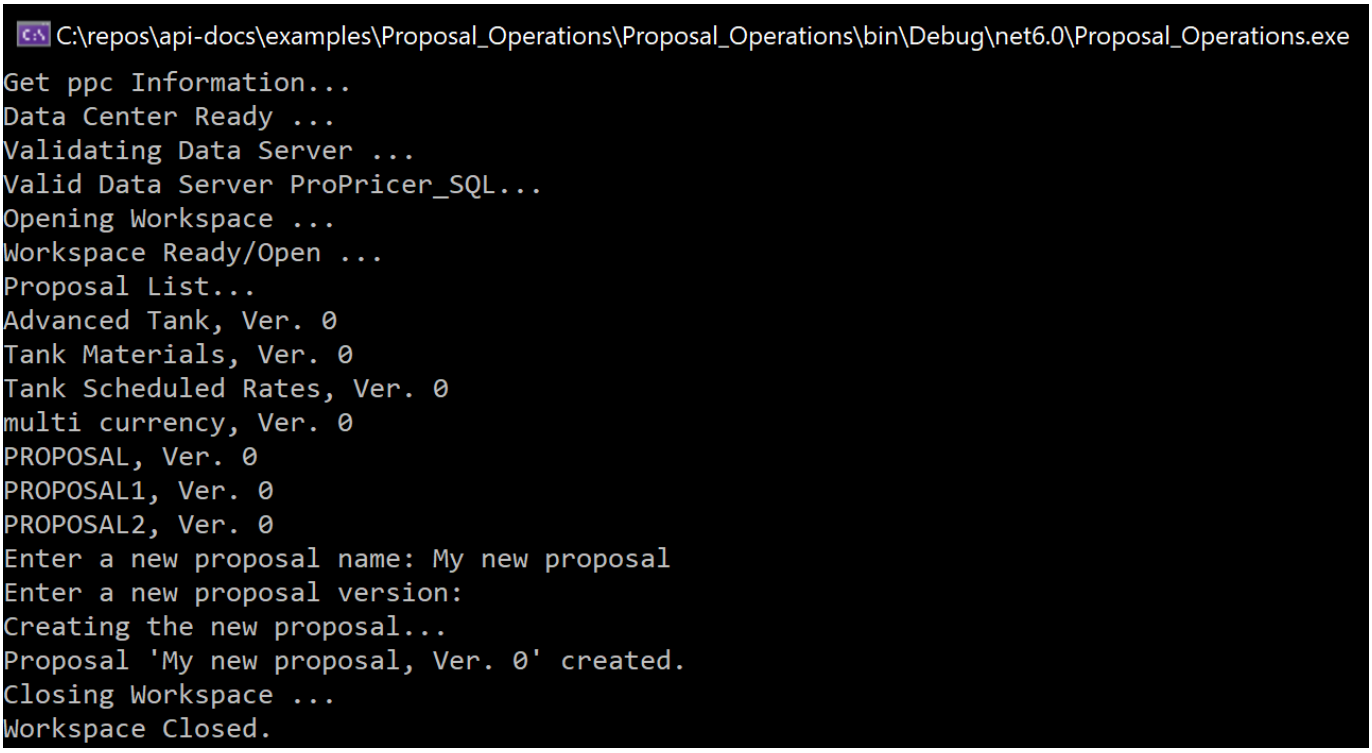
```
Console.Write("Enter a new proposal name: ");
var proposalName = Console.ReadLine();
Console.Write("Enter a new proposal version: ");
var proposalVer = Console.ReadLine();

Console.WriteLine("Creating the new proposal...");
var newProposal = workspace.Proposals.AddNew();
try
{
    if (!string.IsNullOrWhiteSpace(proposalName)) newProposal.Name = proposalName;
    if (!string.IsNullOrWhiteSpace(proposalVer)) newProposal.Version = proposalVer;
    newProposal.Description = "New Proposal Description";
    newProposal.StartDate = DateTime.Now;
    newProposal.EndDate = DateTime.Now.AddMonths(12);
    newProposal.EndEdit();
    Console.WriteLine($"Proposal '{newProposal.Name}, Ver. {newProposal.Version}' created.");
}
catch (Exception ex)
{
    Console.WriteLine("Error adding the proposal: ");
    if (newProposal.BrokenRules.Count > 0)
    {
        foreach (var brokenRule in newProposal.BrokenRules)
        {
            Console.WriteLine(brokenRule.ErrorMessage);
        }
    }
    else
    {
        Console.WriteLine(ex.Message);
    }
    newProposal.CancelEdit();
    return;
}
```

Code comments:

- The AddNew() method is called and is assigned to the newProposal variable.
- The properties Name, Version, Description, Start Date, and End Date have default values. Name and Version automatically generate a non-existing name and version combination. Direct Rate Table is automatically assigned if there is one set as Current.
- You can change any property including Name, Description, Start Date, End Date, Direct Rate Table, and Burden Rate Table.
- The code sets Name and Version only if the user entered a value. Otherwise, the default value is used.

4. Click **Start** to run the solution.

5. Once the solution connects, input a proposal name and version. The new proposal will be added to ProPricer.



How to copy a proposal

1. After creating the new proposal, add the following code.

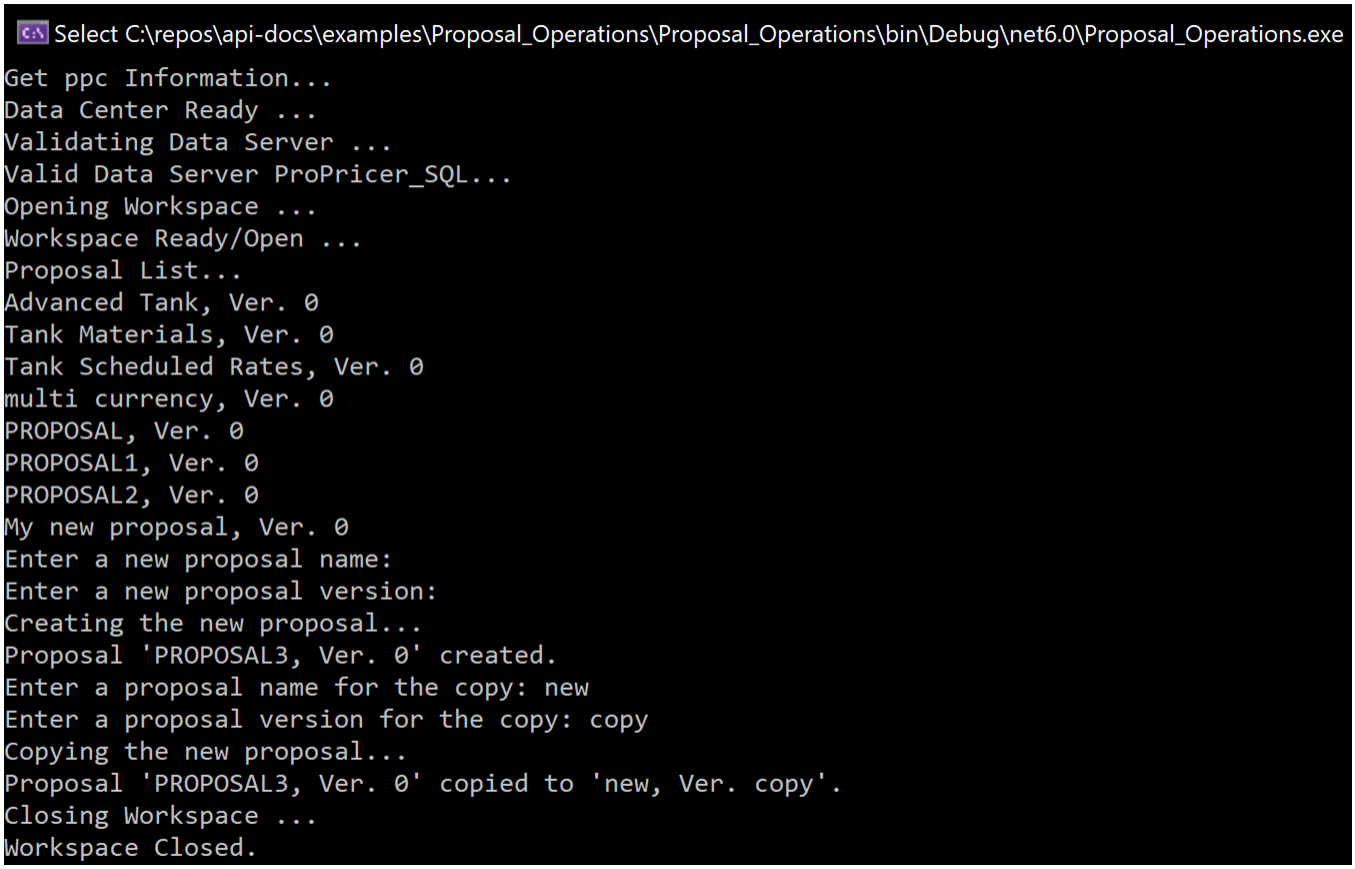
```
Console.Write("Enter a proposal name for the copy: ");
var copyName = Console.ReadLine();
Console.Write("Enter a proposal version for the copy: ");
var copyVer = Console.ReadLine();

Console.WriteLine("Copying the new proposal...");
try
{
    var copyProposal = newProposal.Copy(copyName, copyVer, "API copy test proposal", false);
    Console.WriteLine($"Proposal copied to '{copyProposal.Name}, Ver. {copyProposal.Version}'.");
}
catch (Exception ex)
{
    Console.WriteLine("Error copying the proposal: ");
    Console.WriteLine(ex.Message);
}
```

Code comments:

- The Copy(string name, string version, string description, bool resetWfData) will copy the proposal just created, and assign it the designated name, version, and description.
- Proposal Copy is a special function done by the database for performance reasons.

2. Click **Start** to run the solution.

3. Once the solution connects, enter the information to copy a proposal. The copied proposal will be added to ProPricer.



How to delete a proposal

1. Add the following code.

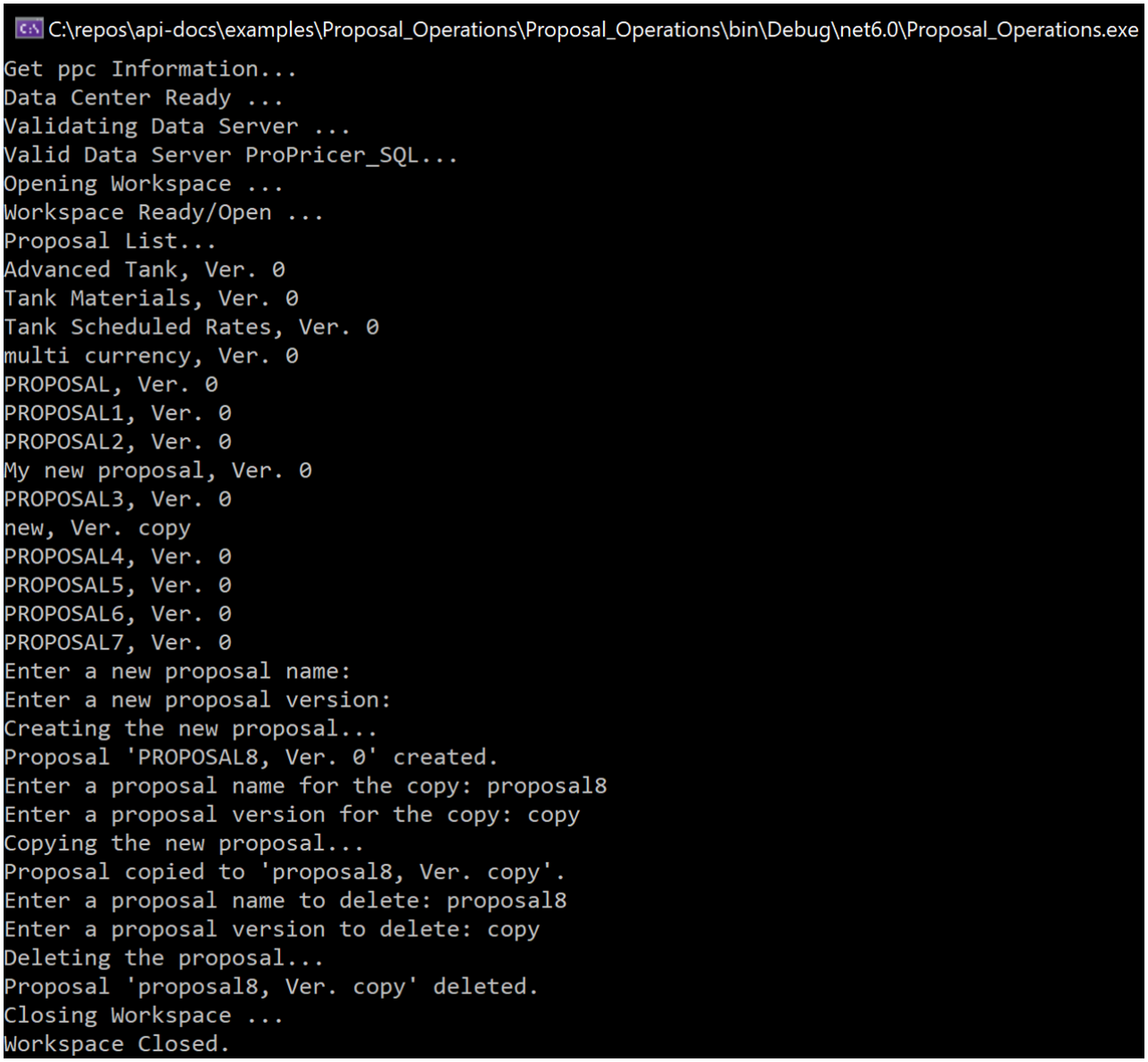
```
Console.Write("Enter a proposal name to delete: ");
var deleteName = Console.ReadLine();
Console.Write("Enter a proposal version to delete: ");
var deleteVer = Console.ReadLine();

var delProposal = workspace.Proposals.Find(deleteName, deleteVer);
if (delProposal != null)
{
    Console.WriteLine("Deleting the proposal...");
    try
    {
        delProposal.Delete();
        Console.WriteLine($"Proposal '{deleteName}, Ver. {deleteVer}' deleted.");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error deleting the proposal: ");
        Console.WriteLine(ex.Message);
    }
}
else
{
    Console.WriteLine($"Proposal '{deleteName}, Ver. {deleteVer}' not found.");
}
```

Code comments:

- The Delete() method removes the proposal from ProPricer.

2. Click **Start** to run the solution.

3. Once the solution connects, enter the proposal name and version. The proposal will be removed from ProPricer.



Complete solution

```
using EBS.ProPricer.Client;
using EBS.ProPricer.Data;
using EBS.ProPricer.Model;
using EBS.ProPricer.Model.General;
using EBS.ProPricer.Registration;

try
{
	Console.WriteLine("Get ppc Information...");

	var conPP = new EBS.ProPricer.Configuration.Client.ClientConfiguration();
	conPP.LoadConfiguration(@"Path to your PPC file");

	var dataCenter = await DataCenter.OpenAsync(conPP.ServerName, conPP.ServerPort);

	Console.WriteLine("Data Center Ready ...");
	var dataServer = dataCenter.DataServers.Find(conPP.ConnectionName);
	Console.WriteLine("Validating Data Server ...");
	if (dataServer == null)
	{
		Console.WriteLine("Invalid Data Server");
		return;
	}
	Console.WriteLine($"Valid Data Server {dataServer.Info.Name}...");
	Console.WriteLine("Opening Workspace ...");
	using (var workspace = await dataServer.OpenWorkspaceAsync(GetLoginInfo,
GetRegistration, GetActivation))
	{
		Console.WriteLine("Workspace Ready/Open ...");

		workspace.Proposals.Open();
		try
		{
			Console.WriteLine("Proposal List...");
			foreach (var proposal in workspace.Proposals.Items())
			{
				Console.WriteLine($"{proposal.Name}, Ver. {proposal.Version}");
			}
		}
		finally
		{
			workspace.Proposals.Close();
		}

		Console.Write("Enter a new proposal name: ");
		var proposalName = Console.ReadLine();
		Console.Write("Enter a new proposal version: ");
		var proposalVer = Console.ReadLine();

		Console.WriteLine("Creating the new proposal...");
		var newProposal = workspace.Proposals.AddNew();
		try
		{
			if (!string.IsNullOrWhiteSpace(proposalName)) newProposal.Name =
proposalName;
			if (!string.IsNullOrWhiteSpace(proposalVer)) newProposal.Version =
proposalVer;
```

```
                newProposal.Description = "New Proposal Description";
                newProposal.StartDate = DateTime.Now;
                newProposal.EndDate = DateTime.Now.AddMonths(12);
                newProposal.EndEdit();
                Console.WriteLine($"Proposal '{newProposal.Name}, Ver.
{newProposal.Version}' created.");
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error adding the proposal: ");
                if (newProposal.BrokenRules.Count > 0)
                {
                    foreach (var brokenRule in newProposal.BrokenRules)
                    {
                        Console.WriteLine(brokenRule.ErrorMessage);
                    }
                }
                else
                {
                    Console.WriteLine(ex.Message);
                }
                newProposal.CancelEdit();
                return;
            }

            Console.Write("Enter a proposal name for the copy: ");
            var copyName = Console.ReadLine();
            Console.Write("Enter a proposal version for the copy: ");
            var copyVer = Console.ReadLine();

            Console.WriteLine("Copying the new proposal...");
            try
            {
                var copyProposal = newProposal.Copy(copyName, copyVer, "API copy test
proposal", false);
                Console.WriteLine($"Proposal copied to '{copyProposal.Name}, Ver.
{copyProposal.Version}'.");
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error copying the proposal: ");
                Console.WriteLine(ex.Message);
            }

            Console.Write("Enter a proposal name to delete: ");
            var deleteName = Console.ReadLine();
            Console.Write("Enter a proposal version to delete: ");
            var deleteVer = Console.ReadLine();

            var delProposal = workspace.Proposals.Find(deleteName, deleteVer);
            if (delProposal != null)
            {
                Console.WriteLine("Deleting the proposal...");
                try
                {
                    delProposal.Delete();
                    Console.WriteLine($"Proposal '{deleteName}, Ver. {deleteVer}'
deleted.");
                }
```

```
                    catch (Exception ex)
                    {
                          Console.WriteLine("Error deleting the proposal: ");
                          Console.WriteLine(ex.Message);
                    }
              }
              else
              {
                    Console.WriteLine($"Proposal '{deleteName}, Ver. {deleteVer}' not
found.");
              }

              Console.WriteLine("Closing Workspace ...");
        }
        Console.WriteLine("Workspace Closed.");
        Console.Read();
}
catch (Exception ex)
{
        Console.WriteLine("Error: ");
        Console.WriteLine(ex.Message);
        Console.WriteLine(ex.StackTrace);
        Console.Read();
}

static bool GetLoginInfo(LogonInfo info)
{
        info.UserName = "sysadmin";
        info.Password = "sysadmin";
        return true;
}

static bool GetRegistration(RegistrationInformation registrationInfo)
{
        registrationInfo.Key = "Your ProPricer API Registration Key";
        if (!string.IsNullOrEmpty(registrationInfo.LastError))
              Console.WriteLine(registrationInfo.LastError);

        return true;
}

static bool GetActivation(ProPricerActivation activation, ref bool changeKey)
{
        try
        {
              changeKey = true;
              if (!activation.Status.IsActivated())
              {
                    Console.WriteLine("Activating key online...");
                    activation.Activate();
              }
        }
        catch (Exception e)
        {
              Console.WriteLine(e.Message);
              Console.WriteLine(e.StackTrace);
        }
        return true;
}
```

Audit Log entries

Audit Log is capable of tracking any event that happens in your ProPricer system. It includes everything in the Change Log, as well as the security-related events selected on the System Logs page of the System Options. After selecting at least one event to record in the System Options, you will enable the Audit Log. The API has access to all entries through the established Workspace.AuditLog collection entity.

How to enable the Audit Log in ProPricer

1. Open ProPricer.
2. On the **Setup** ribbon, in the **Preferences** group, click **System Options**.
3. On the **System Logs** page, select the events the log will record. At least one must be selected to enable the Audit Log.
4. Click **OK** or **Apply**.

The AuditLog collection is read-only, and you access all the entries using the AuditLog.GetEntries(filter) method with an AuditLogEntryFilter instance. The AuditLogEntryFilter has properties like StartTime to specify the initial time you want the entries from.

How to access the Audit Log entries

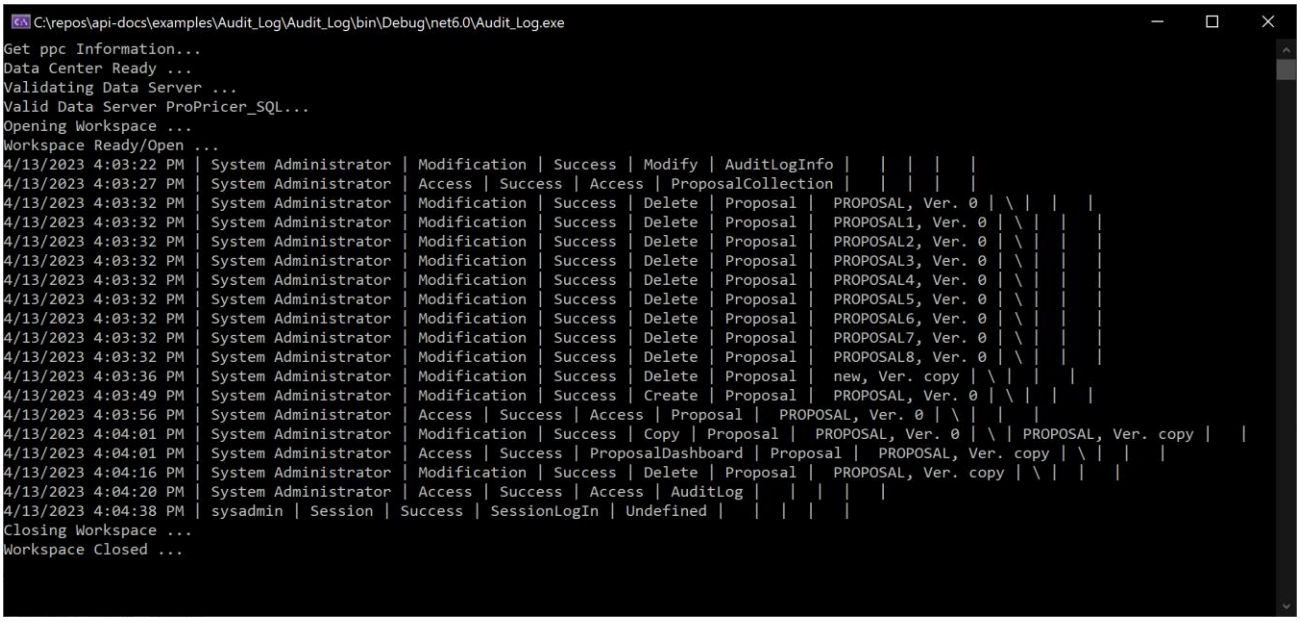
1. Use the console application solution created in the section Establishing a Connection.
2. Inside the using statement, add the following code to access the entries since yesterday.

```
using (var workspace = dataServer.OpenWorkspace(GetLoginInfo, GetRegistration, GetActivation))
{
    Console.WriteLine("Workspace Ready/Open ...");

    var filter = new AuditLogEntryFilter()
    {
        StartTime = DateTime.Today.AddDays(-1)
    };
    foreach (var logEntry in workspace.AuditLog.GetEntries(filter))
    {
        Console.Write($"{logEntry.Time} | {logEntry.UserName} | {logEntry.Category} | ");
        Console.Write($"{logEntry.Result} | {logEntry.Process} | {logEntry.Target} | ");
        Console.Write($"{logEntry.Name} | {logEntry.Location} | {logEntry.NewName} | ");
        Console.WriteLine($"{logEntry.NewLocation} | {logEntry.Details}");
    }
    Console.WriteLine("Closing Workspace ...");
}
```

Code comments:

- Create an instance of the AuditLogEntryFilter class, and set the properties to create your filter.
- The properties available to filter by are: StartTime, EndTime, UserName, UserDescription, Categories, Processes, Results, Targets, Name, Location, NewName, and NewLocation.
- Use the Workspace.AuditLog.GetEntries() method to access the entries that match the filter. Since this is an enumerable list, you can loop through all entries and send them to other log systems, file, or Console.

3. Click **Start** to run the solution and verify you can access the Audit Log entries.



4. If you don’t see any entry, verify the Audit Log is enabled.

Complete solution

```
using EBS.ProPricer.Client;
using EBS.ProPricer.Data;
using EBS.ProPricer.Model;
using EBS.ProPricer.Model.General;
using EBS.ProPricer.Registration;

try
{
	Console.WriteLine("Get ppc Information...");

	var conPP = new EBS.ProPricer.Configuration.Client.ClientConfiguration();
	conPP.LoadConfiguration(@"Path to your PPC file");

	var dataCenter = await DataCenter.OpenAsync(conPP.ServerName, conPP.ServerPort);

	Console.WriteLine("Data Center Ready ...");
	var dataServer = dataCenter.DataServers.Find(conPP.ConnectionName);
	Console.WriteLine("Validating Data Server ...");
	if (dataServer == null)
	{
		Console.WriteLine("Invalid Data Server");
		return;
	}
	Console.WriteLine($"Valid Data Server {dataServer.Info.Name}...");
	Console.WriteLine("Opening Workspace ...");
	using (var workspace = await dataServer.OpenWorkspaceAsync(GetLoginInfo,
GetRegistration, GetActivation))
	{
		Console.WriteLine("Workspace Ready/Open ...");

		var filter = new AuditLogEntryFilter()
		{
			StartTime = DateTime.Today.AddDays(-1)
		};
		foreach (var logEntry in workspace.AuditLog.GetEntries(filter))
		{
			Console.Write($"{logEntry.Time} | {logEntry.UserName} |
{logEntry.Category} | ");
			Console.Write($"{logEntry.Result} | {logEntry.Process} |
{logEntry.Target} | ");
			Console.Write($" {logEntry.Name} | {logEntry.Location} |
{logEntry.NewName} | ");
			Console.WriteLine($" {logEntry.NewLocation} | {logEntry.Details}");
		}

		Console.WriteLine("Closing Workspace ...");
	}
	Console.WriteLine("Workspace Closed ...");
	Console.Read();
}
catch (Exception ex)
{
	Console.WriteLine("Error: ");
	Console.WriteLine(ex.Message);
	Console.WriteLine(ex.StackTrace);
	Console.Read();
}
```

```
static bool GetLoginInfo(LogonInfo info)
{
	info.UserName = "sysadmin";
	info.Password = "sysadmin";
	return true;
}
static bool GetRegistration(RegistrationInformation registrationInfo)
{
	registrationInfo.Key = "Your ProPricer API Registration Key";
	if (!string.IsNullOrEmpty(registrationInfo.LastError))
		Console.WriteLine(registrationInfo.LastError);
	return true;
}

static bool GetActivation(ProPricerActivation activation, ref bool changeKey)
{
	try
	{
		changeKey = true;
		if (!activation.Status.IsActivated())
		{
			Console.WriteLine("Activating key online...");
			activation.Activate();
		}
	}
	catch (Exception e)
	{
		Console.WriteLine(e.Message);
		Console.WriteLine(e.StackTrace);
	}
	return true;
}
```

Commonly used items

This is a list of most commonly used items in the ProPricer API.

Workspace

Workspace.Open(): open a workspace to connect to the proposals, libraries, and system information.

Workspace.Close(): close the workspace.

Global Library

Workspace.GlobalLibrary.BurdenRateTables: access to the Indirect Rate Tables and the associated elements within that collection.

Workspace.GlobalLibrary.EscalationRateTables: access to the Escalation Rate Tables and the associated elements within that collection. These tables are specific to Materials.

Workspace.GlobalLibrary.ExchangeRateTables: access to the Exchange Rate Tables and the associated elements within that collection. These tables are specific to Currency.

Workspace.GlobalLibrary.FactorRateTables: access to the CER Factor Tables and the associated elements within that collection. These tables are specific to CERs.

Workspace.GlobalLibrary.ResourceRateTables: access all the Rate Tables for Resources including Direct Rate Tables, Composite Rate Tables, and T&M Rate Tables and the associated elements within that collection.

Workspace.GlobalLibrary.TravelRateTables: access to the Travel Rate Tables and the associated elements within that collection.

Workspace.GlobalLibrary.Folders: access to the parent and child folder structure in all categories (proposals, rates, etc.).

Workspace.GlobalLibrary.ResourceClasses: access to the Resource Class Library.

Workspace.GlobalLibrary.ResourceFieldDefinitions: access to the Global Resource Fields and the associated value lists.

Workspace.GlobalLibrary.Resources: access to the Resource Library.

Workspace.GlobalLibrary.TitleTables: access to the Title Table Library.

Workspace.GlobalLibrary.Travels: access to the Travel Library.

Proposal

Workspace.Proposals.AddNew(): add a new proposal.

Workspace.Proposals.Open(): open the proposal.

Workspace.Proposals.Close(): close the proposal.

Workspace.Proposals.Find(string name, string version): find a proposal.

Material

Workspace.GlobalLibrary.MaterialLibraries: access to the Product Libraries.

Workspace.GlobalLibrary.MaterialLibraries.CommonLibrary.Materials.Parts: access to the Parts in the (Common) Material Library.

Workspace.GlobalLibrary.MaterialLibraries.CommonLibrary.Materials.Assemblies: access to the Assemblies in the (Common) Material Library.

Workspace.GlobalLibrary.MaterialLibraries.CommonLibrary.Materials: access to the Parts and Assemblies in the (Common) Material Library.

Workspace.GlobalLibrary.MaterialLibraries.CommonLibrary.Materials.OpenCostSources(): required before you access part.CostSources or assembly.CostSources collections.

Part.CostSources: access to the Cost Sources (Quote, P.O., Miscellaneous) in the Part.

Part.AssociatedCosts: access to the Associated Costs in the Part.

Assembly.CostSources: access to the Cost Sources (Quote, P.O., Miscellaneous) in the Assembly.

Assembly.Elements: access to the elements (Parts and Assemblies) in the Assembly.

System

Workspace.AuditLog.GetEntries(AuditLogEntryFilter filter): access to the Audit Log entries that match the filter parameter.

Workspace.Users.AddNew(): add a new user.

Workspace.Users.Find(string LoginName): find a user.

Methods

Most items within the workspace share common methods that can be called once the element is defined.

Open(): open the element to allow for information retrieval or access.

Items(): access the elements in the collection.

Close(): close the element and release the lock.

AddNew(): add to an existing collection.

BeginEdit(): used when modifying an existing element in the collection.

EndEdit(): used when finished with AddNew() or BeginEdit(). This will release the lock on the individual record and process the commit.

CancelEdit(): used to cancel the BeginEdit() when there is an exception and the BeginEdit() cannot be completed.

Delete(): delete the selected record.

DeleteAll(): delete all instances of the selected record.

Copy(): copy the selected record.

IsOpened(): verify the element is still in an opened state.