REST Web Service using APL64 Cross-platform Component

Table of Contents

Overview

Use an APL64 cross-platform component (CPC) to support the algorithmic content of a rest web service example.

A rest web service is created in Visual Studio. For purposes of the example, the web service is run locally using Microsoft IIS web server. The web service exposes CRUD operations on a server-side data source. The simplified data source in the example is a list of Assay records. The server-side operations exposed to a client are:

- Create a new Assay record
- Read all or specified Assay record(s)
- Update an Assay record
- Delete an Assay record

Download Sample Solution

Download the sample solution from:

http://apl2000.com/APL64/UserDocumentation/APL64CpcRestSvc.zip

Extract the zip-format file to c:\.

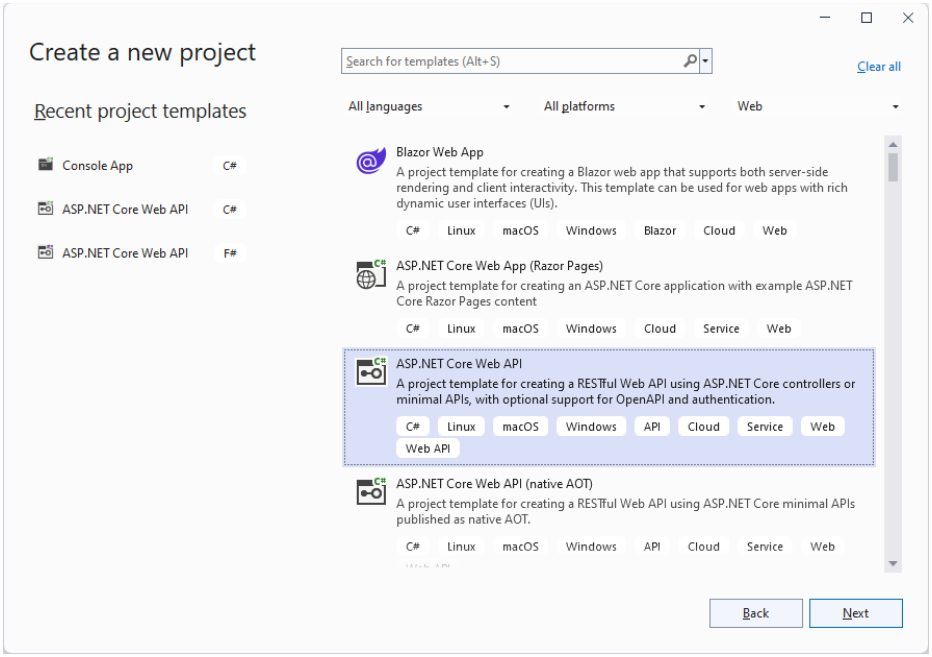
The zip-format file contains a Visual Studio Github repository with:

- APL64 CPC folder with APL64 workspace for the CPC
- APL64CpcRestSvc .Net project
- RestClient .Net project

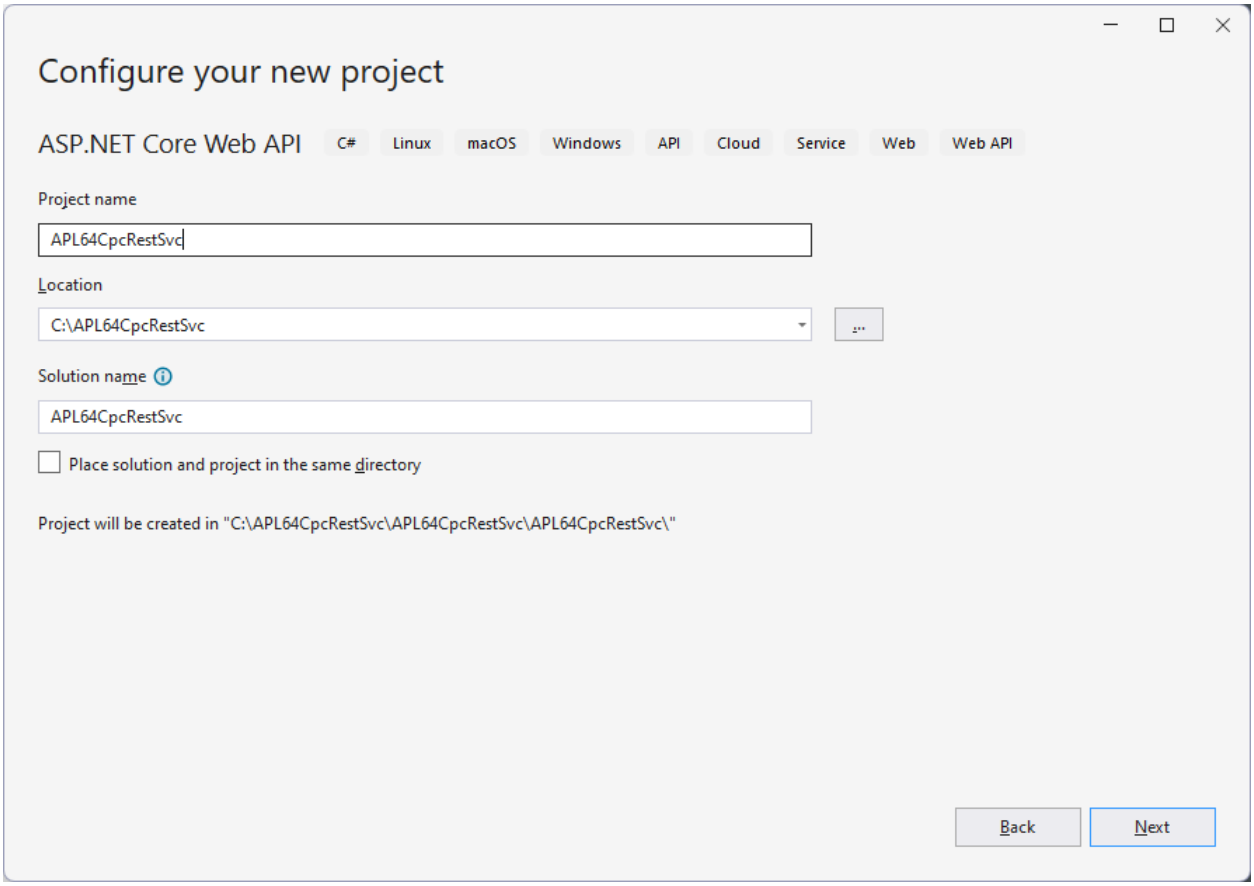
Create Rest Web Service

Create a New Visual Studio Project

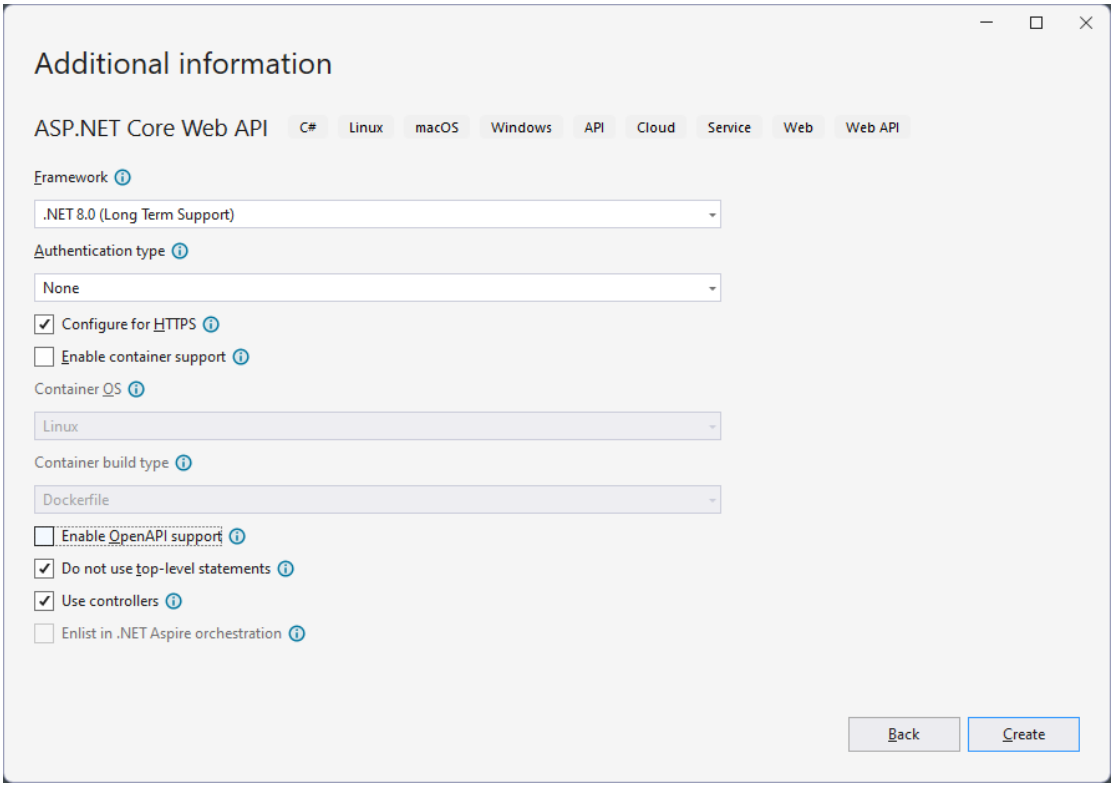
Select the project template



Configure the REST service project



Provide the additional information



Modify the RestWebService project C# code

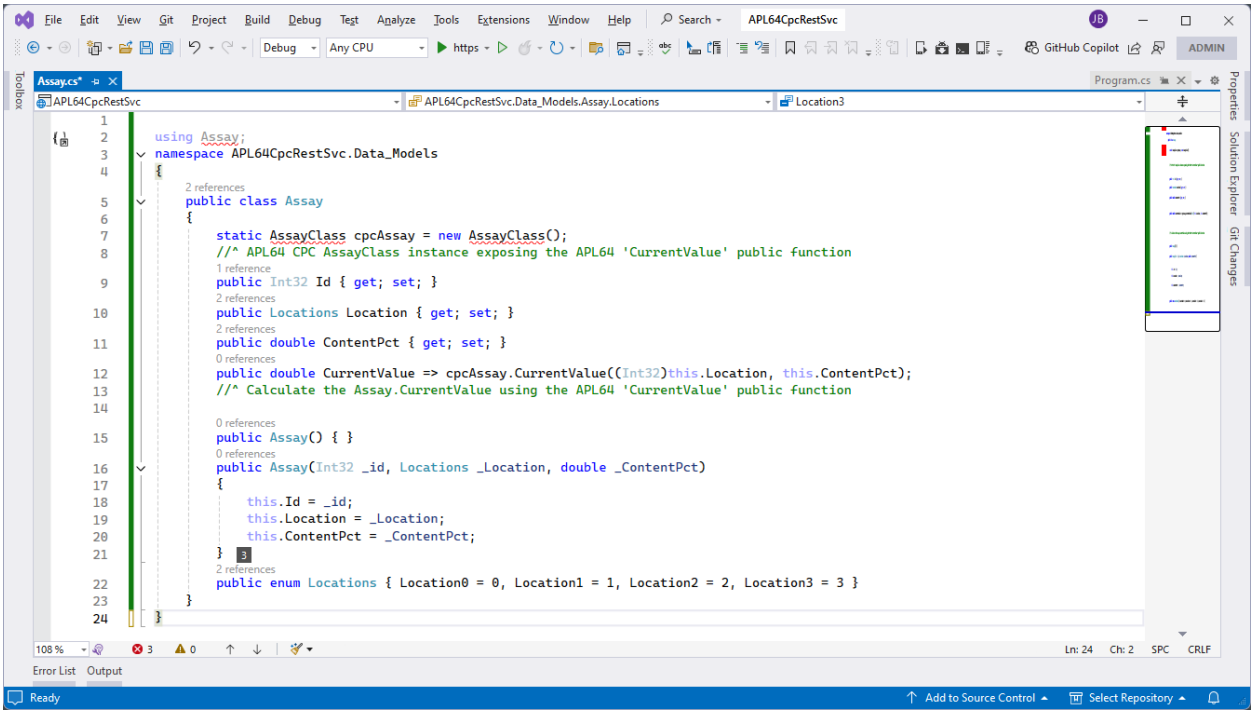
Delete the WeatherForecast.cs code file

Add the 'Data Models' folder to the APL64CpcRestSvc project

Add the Assay.cs class file to the Data Models folder with this code:

```
using Assay;
namespace APL64CpcRestSvc.Data_Models
{
   public class Assay
   {
      static AssayClass cpcAssay = new AssayClass();
      //^ APL64 CPC AssayClass instance exposing the APL64 'CurrentValue' public function
      public Int32 Id { get; set; }
      public Locations Location { get; set; }
      public double ContentPct { get; set; }
      public double CurrentValue => cpcAssay.CurrentValue((Int32)this.Location, this.ContentPct);
      //^ Calculate the Assay.CurrentValue using the APL64 'CurrentValue' public function

      public Assay() { }
      public Assay(Int32 _id, Locations _Location, double _ContentPct)
      {
         this.Id = _id;
         this.Location = _Location;
         this.ContentPct = _ContentPct;
      }
      public enum Locations { Location0 = 0, Location1 = 1, Location2 = 2, Location3 = 3 }
```

```
    }
}
```

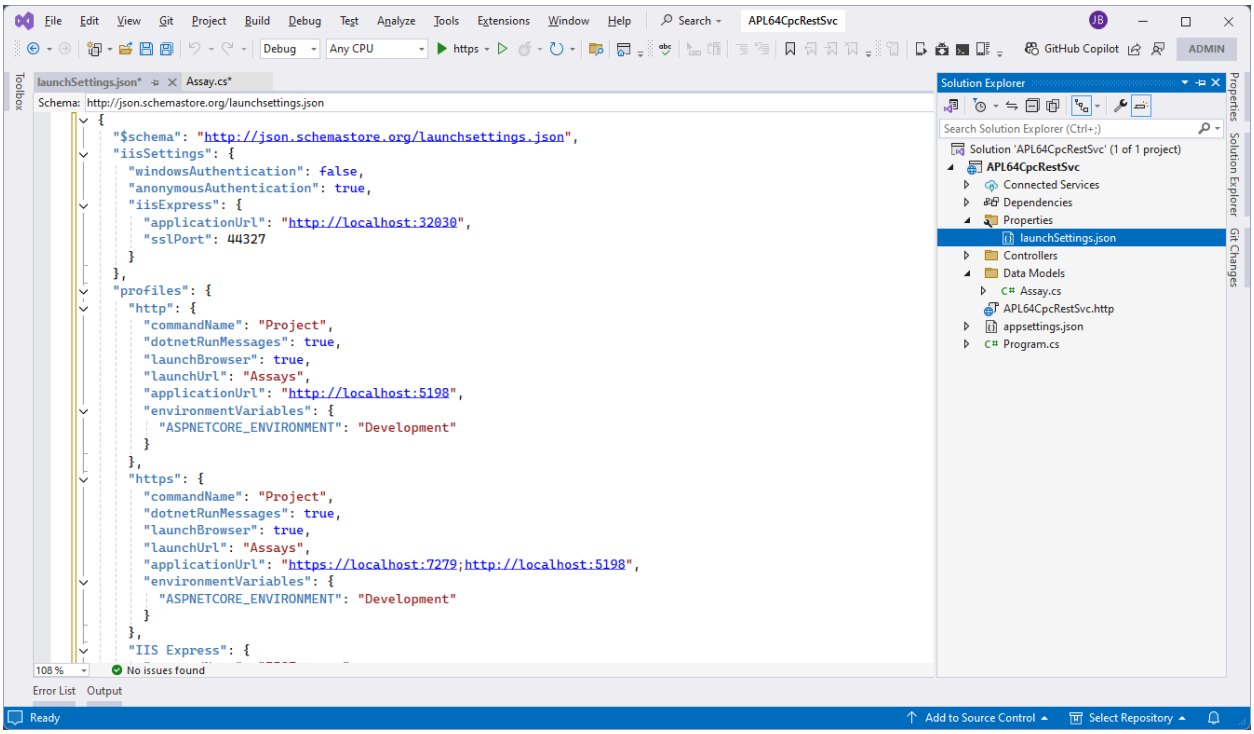


The Assay class definition exposes the applicable properties and methods of an Assay record.

Modify the Properties/LaunchSettings.json file with this content:

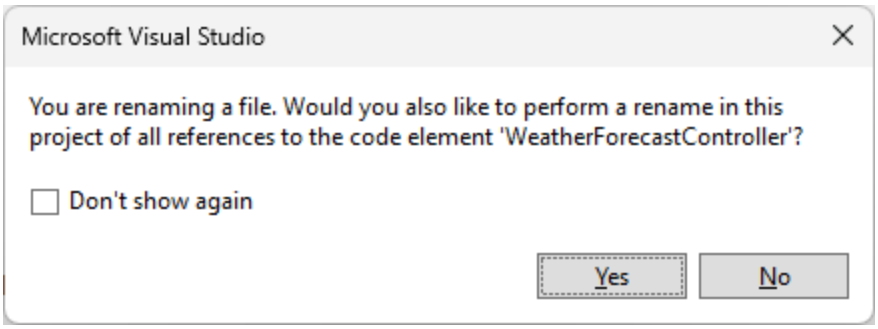
```
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:32030",
      "sslPort": 44327
    }
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "launchUrl": "Assays",
      "applicationUrl": "http://localhost:5198",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "https": {
      "commandName": "Project",
      "dotnetRunMessages": true,
```

```
    "launchBrowser": true,
    "launchUrl": "Assays",
    "applicationUrl": "https://localhost:7279;http://localhost:5198",
    "environmentVariables": {
     "ASPNETCORE_ENVIRONMENT": "Development"
    }
   },
   "IIS Express": {
    "commandName": "IISExpress",
    "launchBrowser": true,
    "launchUrl": "Assays",
    "environmentVariables": {
     "ASPNETCORE_ENVIRONMENT": "Development"
    }
   }
  }
}
```



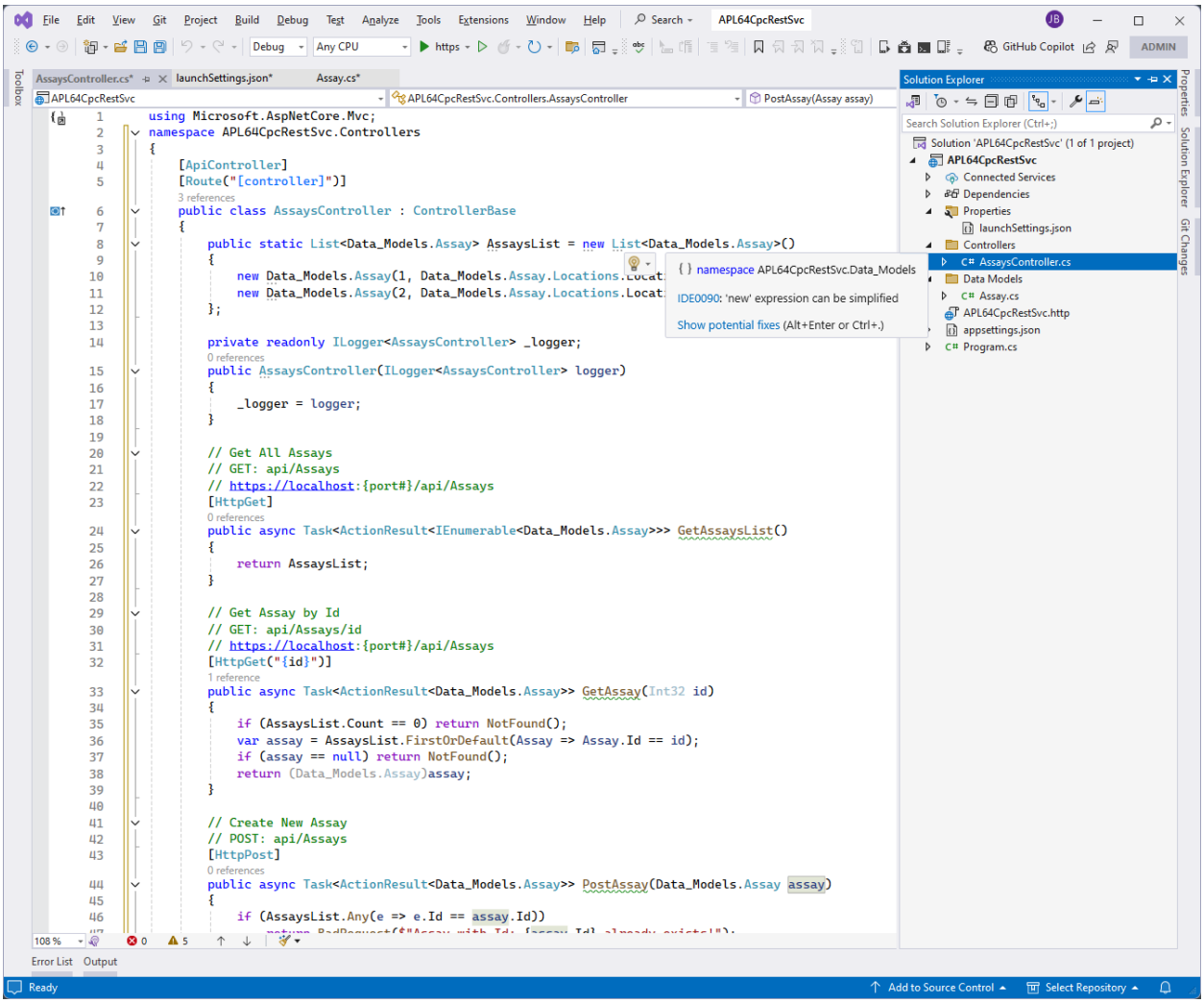
Rename the WeatherForecastController.cs code file to AssaysController.csand replace the existing content

When renaming the file, if Visual Studio presents this dialog, click the ‘Yes’ button:



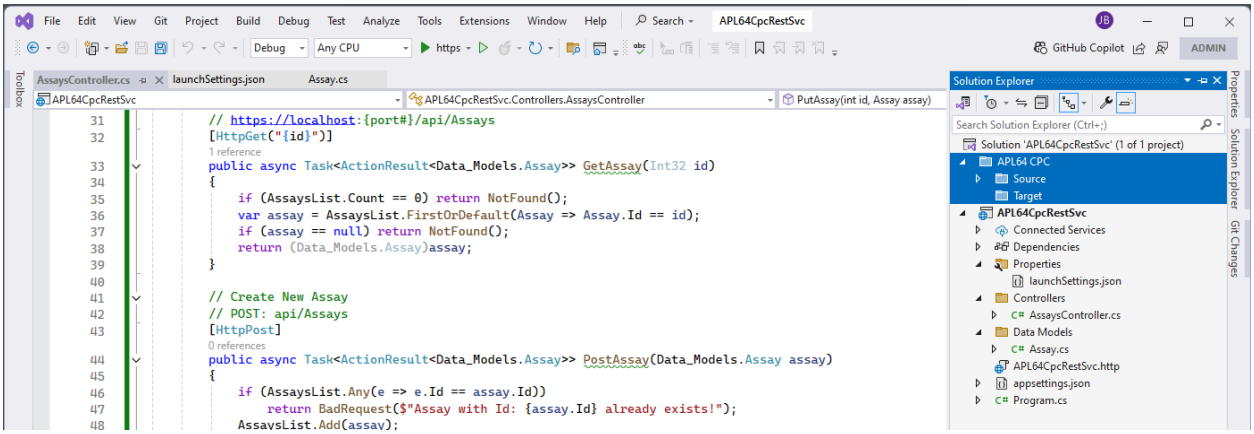

```
using Microsoft.AspNetCore.Mvc;
namespace APL64CpcRestSvc.Controllers
{
  [ApiController]
  [Route("[controller]")]
  public class AssaysController : ControllerBase
  {
    public static List<Data_Models.Assay> AssaysList = new List<Data_Models.Assay>()
    {
      new Data_Models.Assay(1, Data_Models.Assay.Locations.Location1, 2.34),
      new Data_Models.Assay(2, Data_Models.Assay.Locations.Location2, 1.23)
    };

    private readonly ILogger<AssaysController> _logger;
    public AssaysController(ILogger<AssaysController> logger)
    {
      _logger = logger;
    }

    // Get All Assays
    // GET: api/Assays
    // https://localhost:{port#}/api/Assays
    [HttpGet]
    public async Task<ActionResult<IEnumerable<Data_Models.Assay>>> GetAssaysList()
    {
      return AssaysList;
    }

    // Get Assay by Id
    // GET: api/Assays/id
    // https://localhost:{port#}/api/Assays
    [HttpGet("{id}")]
    public async Task<ActionResult<Data_Models.Assay>> GetAssay(Int32 id)
    {
      if (AssaysList.Count == 0) return NotFound();
      var assay = AssaysList.FirstOrDefault(Assay => Assay.Id == id);
```

```
        if (assay == null) return NotFound();
        return (Data_Models.Assay)assay;
    }

    // Create New Assay
    // POST: api/Assays
    [HttpPost]
    public async Task<ActionResult<Data_Models.Assay>> PostAssay(Data_Models.Assay assay)
    {
        if (AssaysList.Any(e => e.Id == assay.Id))
            return BadRequest($"Assay with Id: {assay.Id} already exists!");
        AssaysList.Add(assay);
        return CreatedAtAction(nameof(GetAssay), new { id = assay.Id }, assay);
    }

    // Update Existing Assay
    // PUT: api/Assays
    [HttpPut("{id}")]
    public async Task<IActionResult> PutAssay(Int32 id, Data_Models.Assay assay)
    {
        if (id != assay.Id) return BadRequest($"Assay.Id {assay.Id} does not match id {id}");
        if (AssaysList.Count == 0) return NotFound();
        var index = AssaysList.Select(elt => elt.Id).ToList().IndexOf(id);
        if (index == -1) return NotFound();
        AssaysList[index] = assay;
        return NoContent();
    }

    // Delete Assay by Id
    // DELETE: api/Assays
    [HttpDelete("{id}")]
    public async Task<ActionResult> DeleteAssay(Int32 id)
    {
        var Assay = AssaysList.FirstOrDefault(elt => elt.Id == id);
        if (Assay == null) return NotFound();
        AssaysList.Remove(Assay);
        return NoContent();
    }
  }
}
```



Create the APL64 Cross-platform Component

Add the ‘APL64 CPC’ folder to the REST service solution folder with Source and Target sub-folders



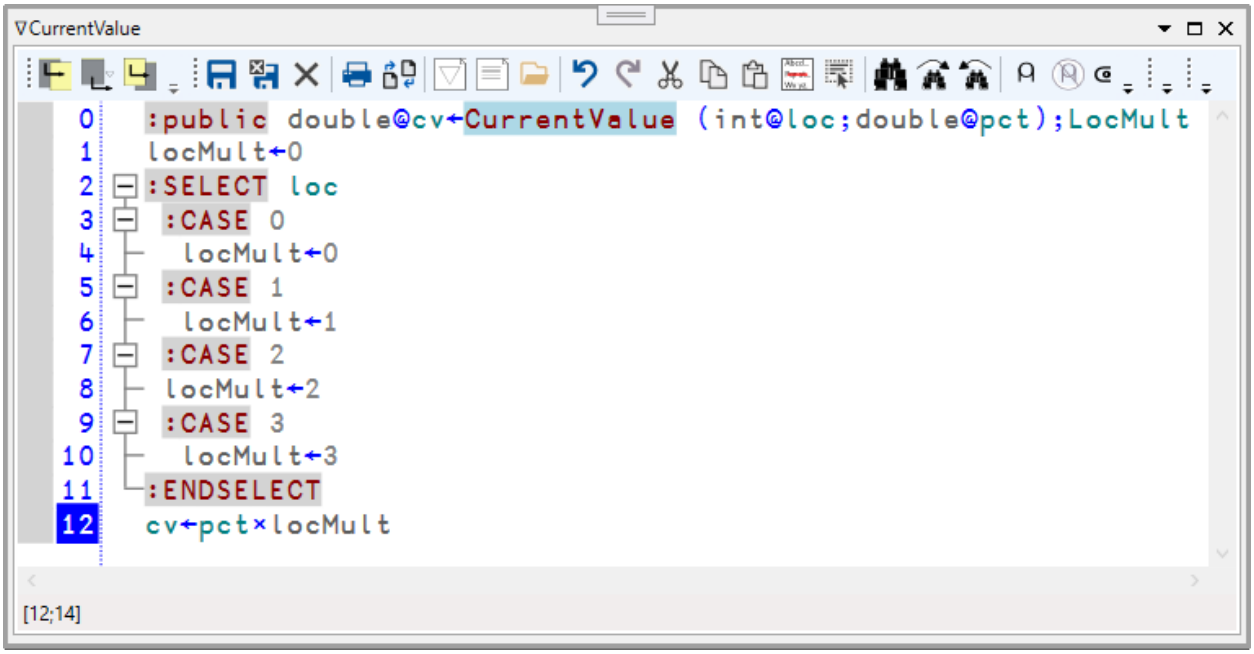
Create the public function ‘CurrentValue’ in the ‘Assay’ workspace

Start the APL64 Developer version with a clear workspace and define the ‘CurrentValue’ public function.

The ‘CurrentValue’ public function

- Arguments:
  -
  - ‘loc’ an integer indicating the ‘Location’ of the assay
  - ‘pct ‘ a double indicating the ‘contentPct’ of the assay
- Result: ‘cv’ a double
- Local variable: LocMult

```
:public double@cv←CurrentValue (int@loc;double@pct);LocMult
locMult←0
:SELECT                                                    loc
 :CASE                                                       0
  locMult←0
 :CASE                                                       1
  locMult←1
 :CASE                                                       2
 locMult←2
 :CASE                                                       3
  locMult←3
:ENDSELECT
cv←pct×locMult
```

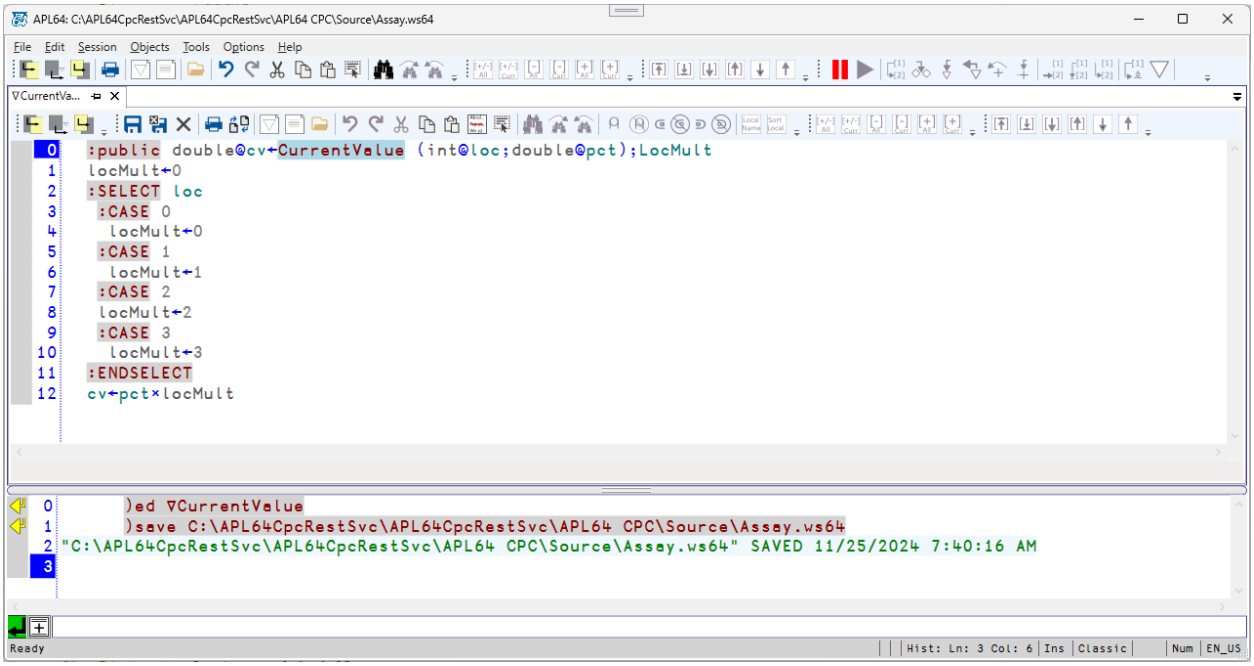


If necessary, create the physical folders:

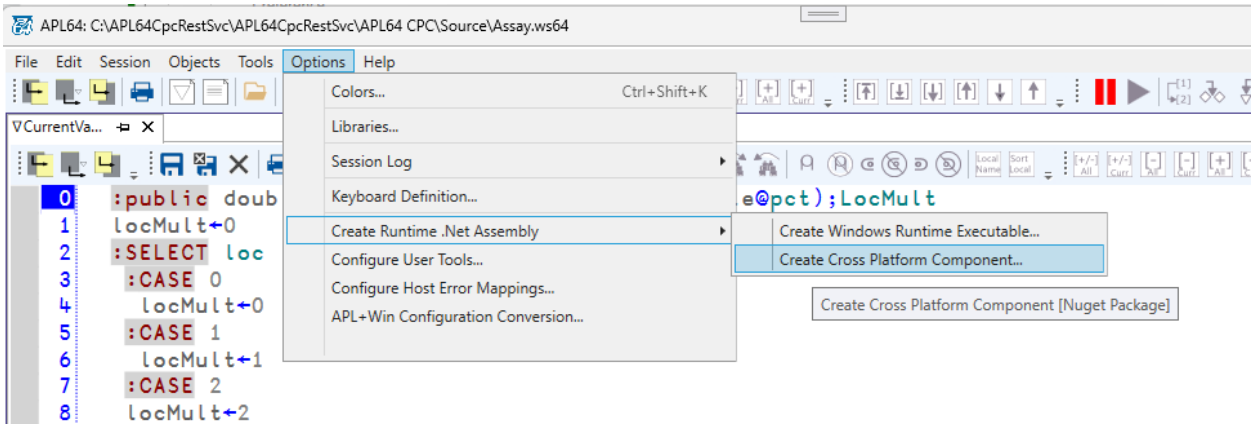
- C:\APL64CpcRestSvc\ APL64CpcRestSvc \APL64 CPC\
- C:\APL64CpcRestSvc\ APL64CpcRestSvc \APL64 CPC\ Source
- C:\APL64CpcRestSvc\ APL64CpcRestSvc \APL64 CPC\ Target

Save the APL64 workspace as 'Assay.ws64' into the folder:

C:\APL64CpcRestSvc\ APL64CpcRestSvc \APL64 CPC\ Source

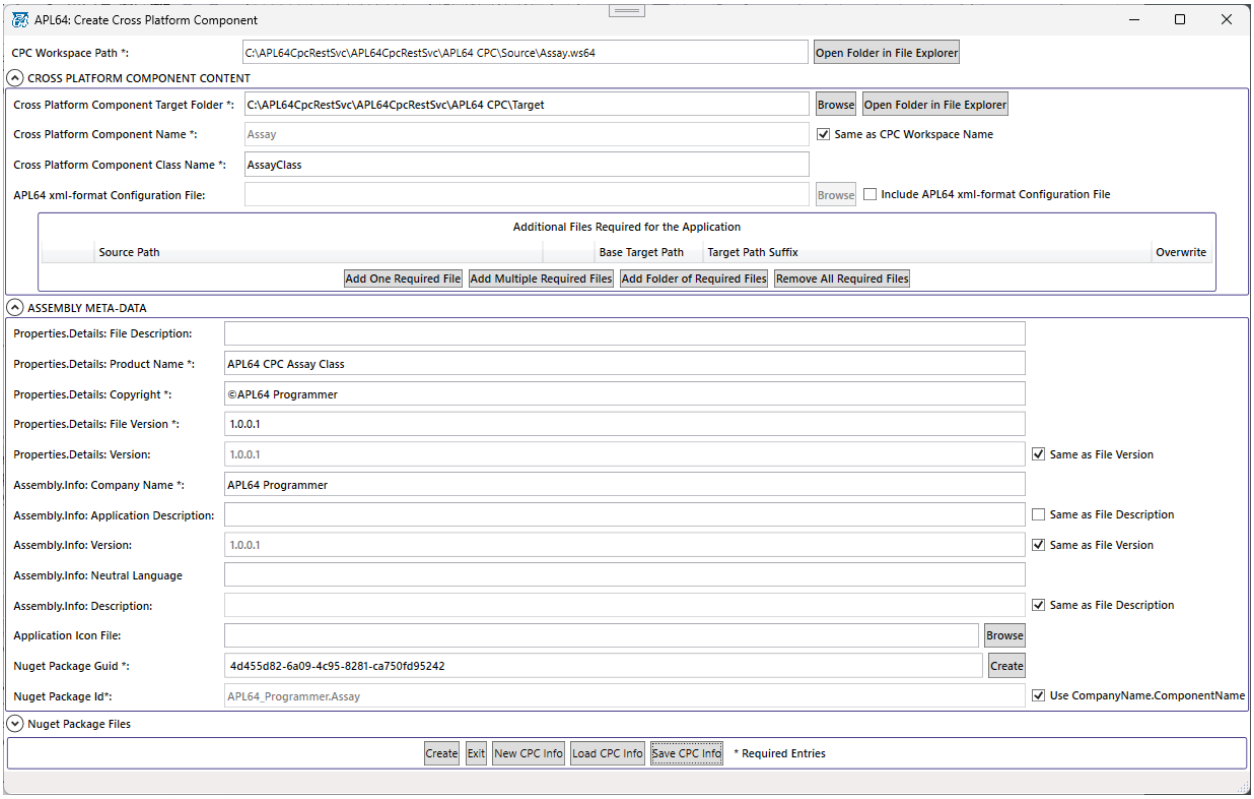


Use the APL64 CPC Utility

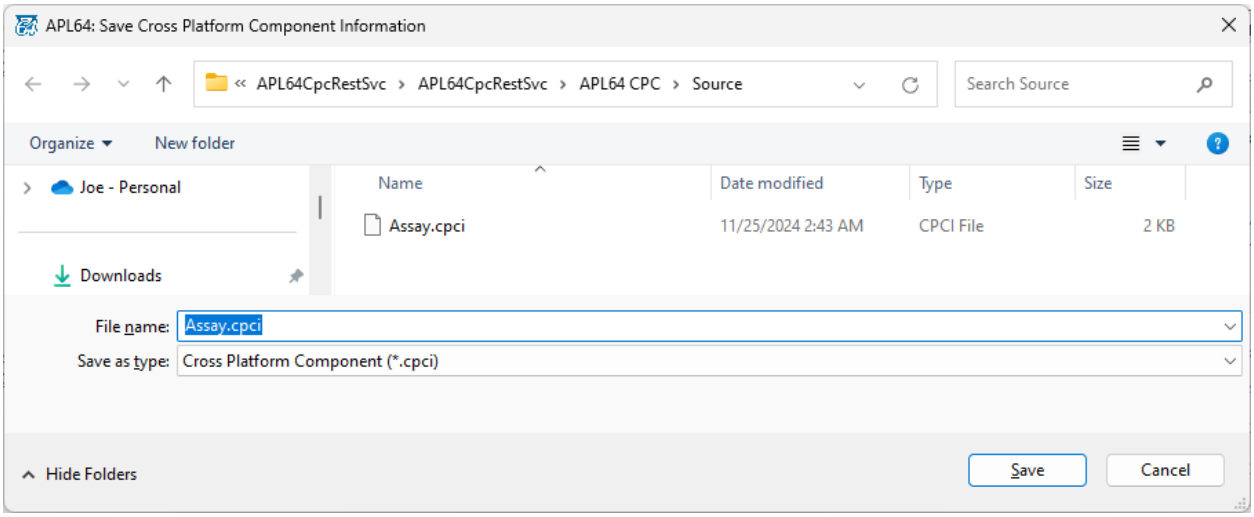


Complete the CPC Utility dialog

All entries marked with an asterisk (*) must be completed. The File Version should be updated whenever a modification is made to the APL64 CPC.

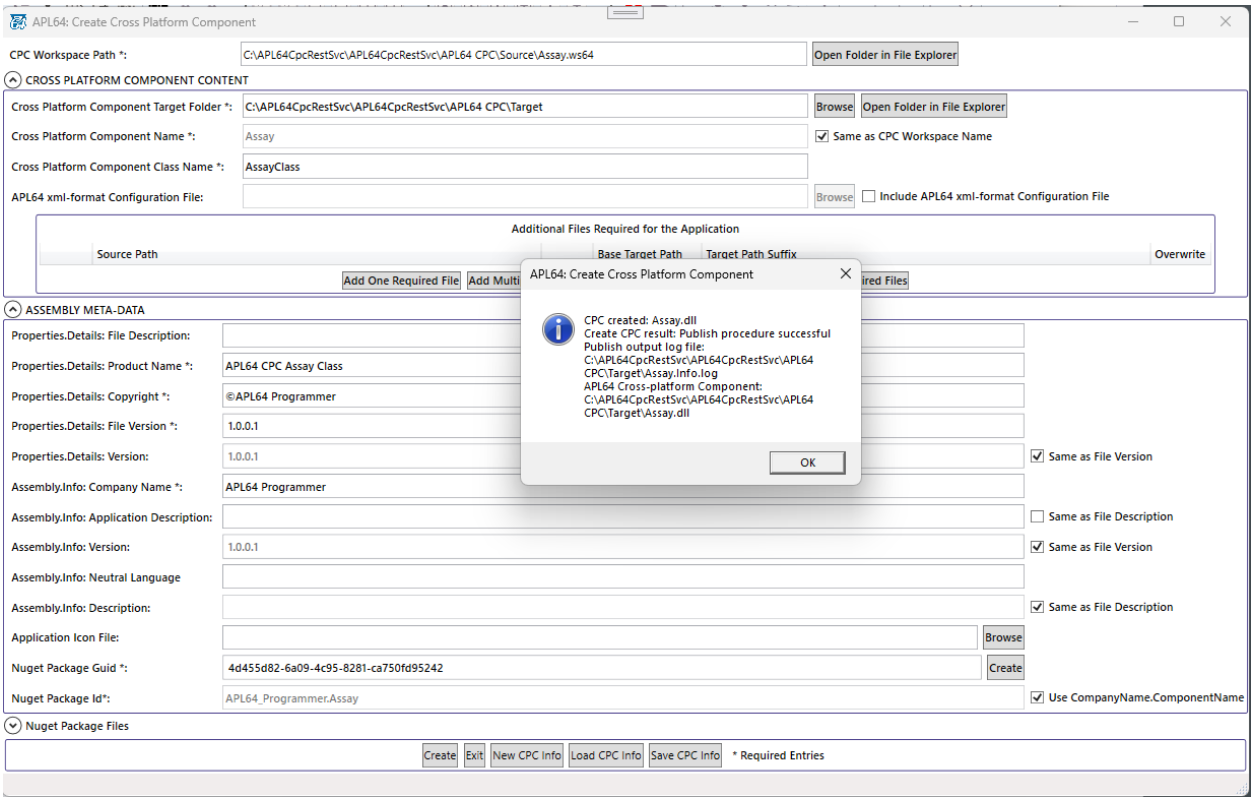


Save the CPC information file to the Source folder



Create the CPC in the Target folder

When the ‘Create’ button is clicked in the CPC utility dialog, the CPC will be created:

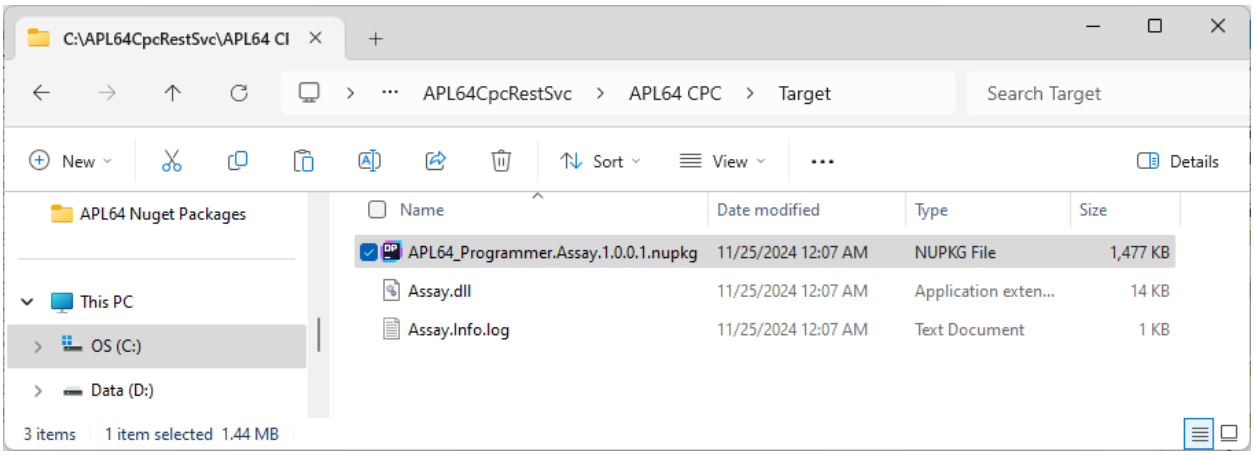


Publish the CPC Nuget Package Locally

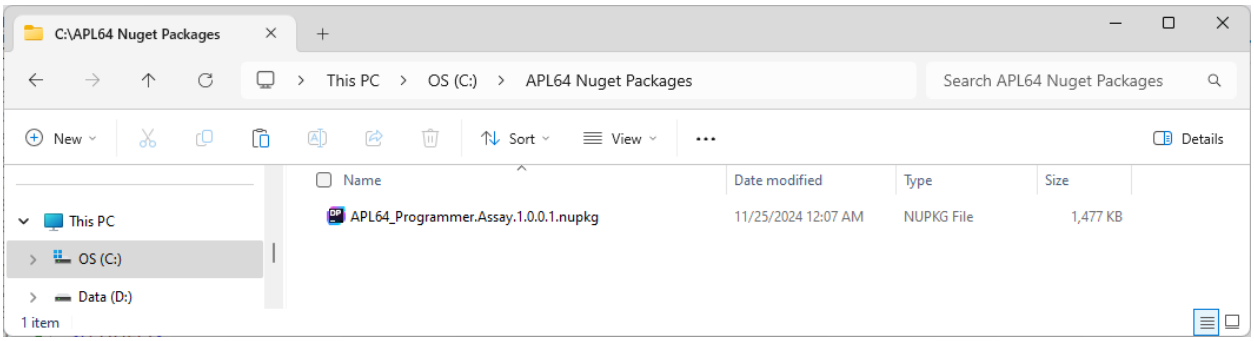
Create the folder: C:\APL64 Nuget Packages\ for testing and debugging purposes. In a production environment the CPC Nuget package would be published in a secure Nuget package repository on the Web.

Do not use the Target folder in the RestWebServiceWithAPL64Cpc project of the solution as the local publish for APL64 Nuget packages under development. This will avoid including spurious components in the project.

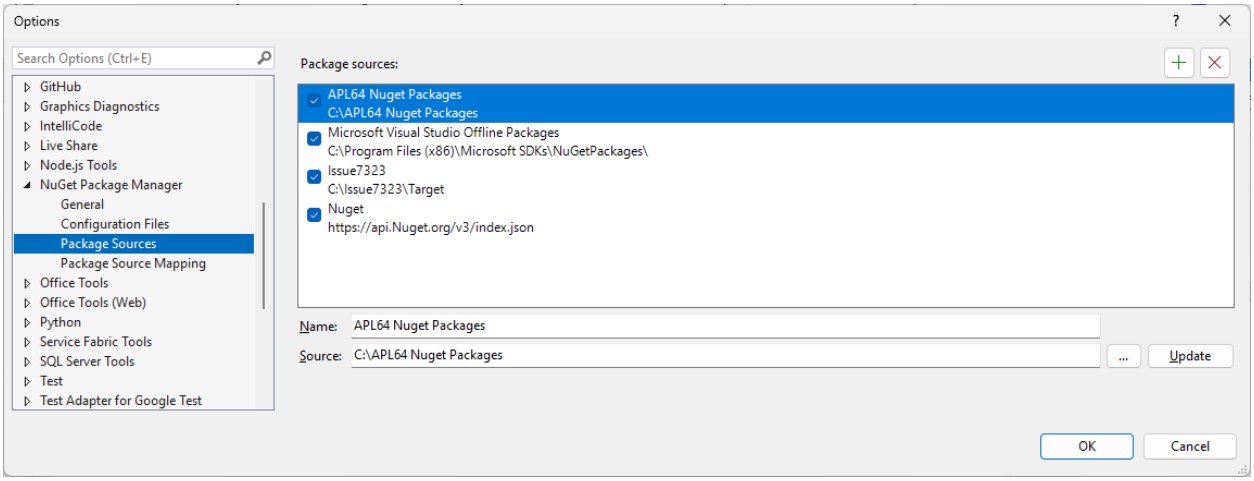
From:



To:

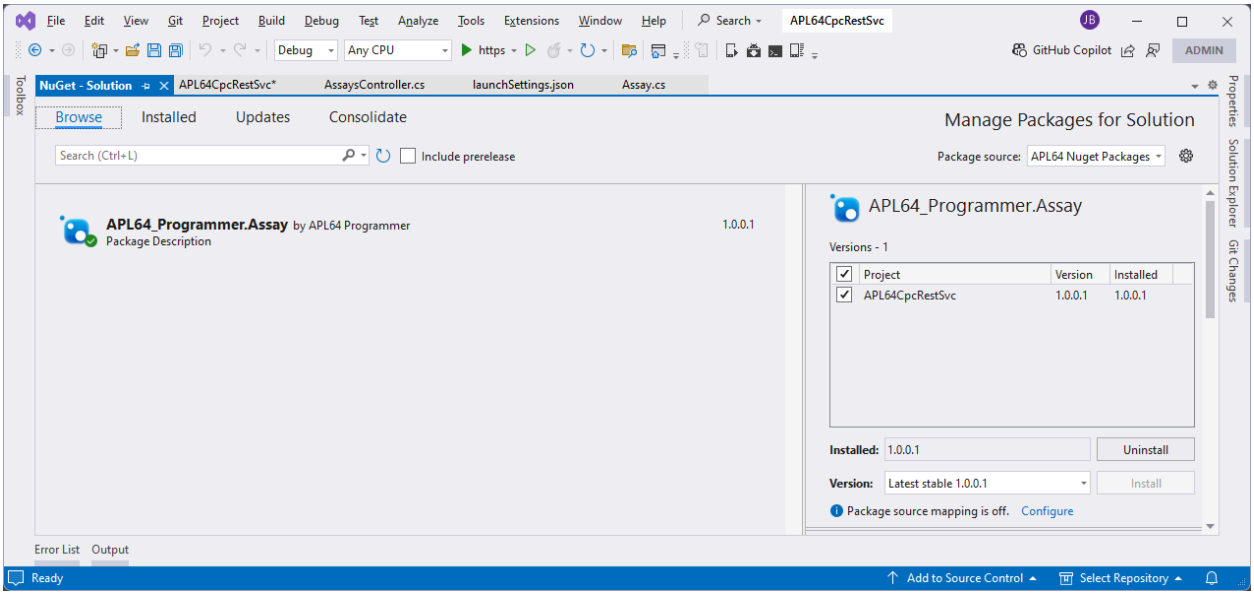


The local Nuget package publish folder must be registered with Visual Studio as a Nuget package source:



Install the CPC 'Assay' Nuget Package into the Project

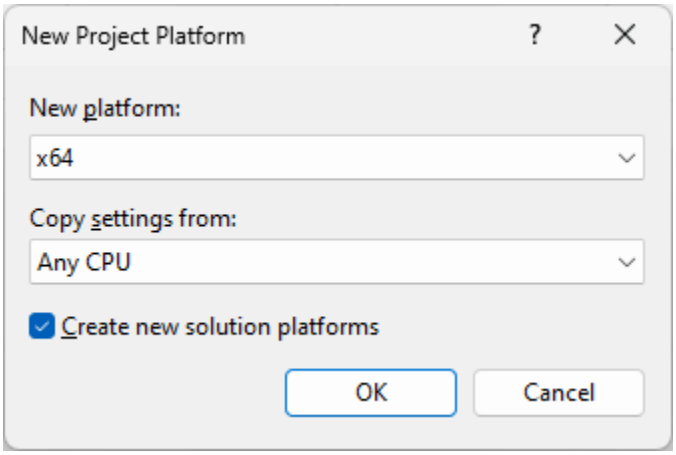
Use the Visual Studio | Tools | Nuget Package Manager | Manage Nuget Packages for the solution... dialog:

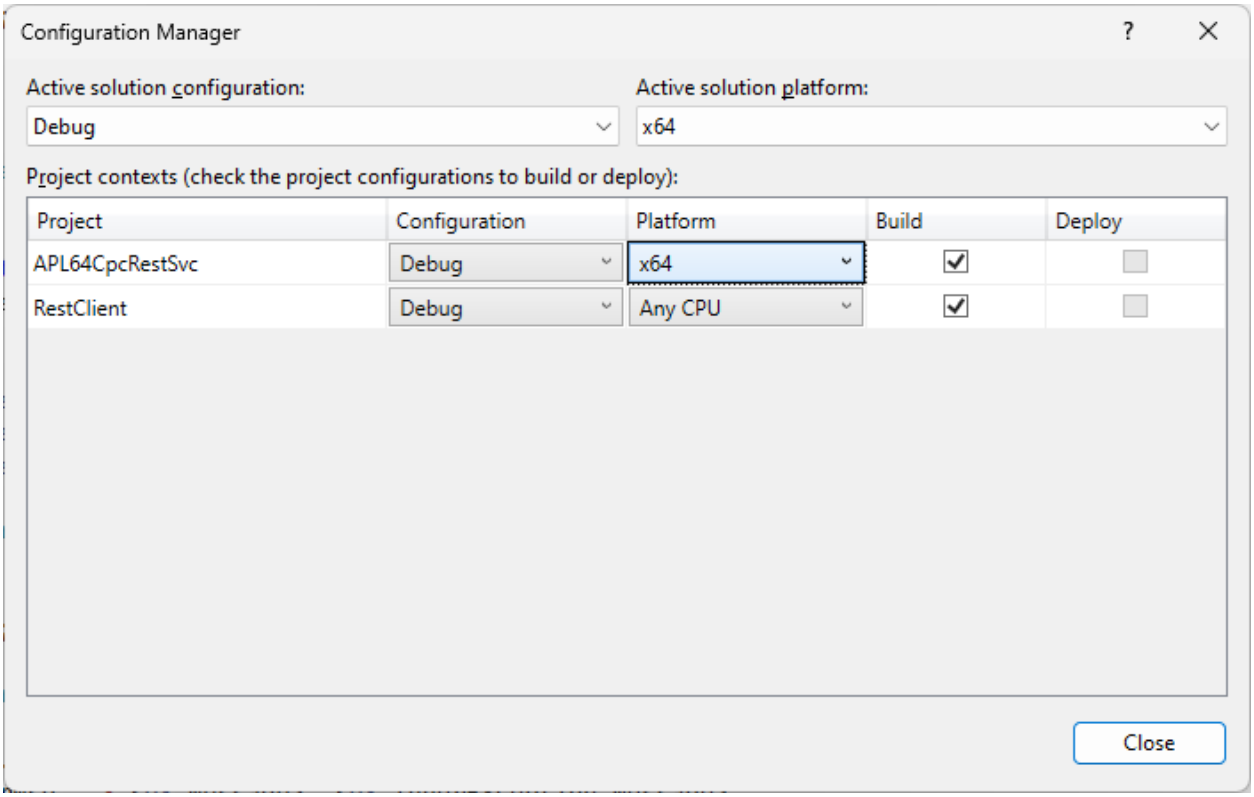


Set x64 Build Configuration for Rest Service Project

Use the Build | Configuration Manager dialog to set the Platform to x64.

It may be necessary to copy the existing Any CPU platform to create the x64 platform:





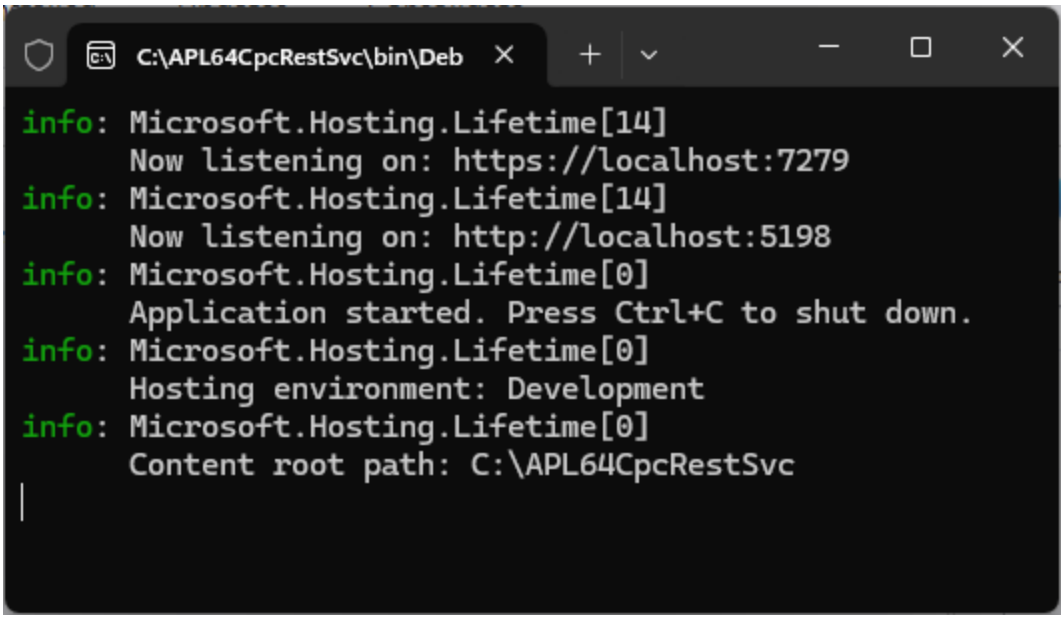
Start the REST Service

Run the Rest service project in Visual Studio to start the service.

Visual Studio may issue warnings when starting the service for the first time, such as 'Trust self-signed certificate'. In a development environment it is generally appropriate to continue testing. The conditions revealed by such warnings must be properly resolved in a production environment.

When started, the REST web service initializes the AssaysList, which is a .Net Generic List of Assay class instances (records). The running REST web service exposes methods which support the CRUD operations on the AssaysList. In a production environment, the simple AssaysList would be replaced by a robust and secure database.

CRUD Operation	**RestWebServiceMethod**
Read	GetAssays() and GetAssay()
Create	PostAssay()
Update	PutAssay()
Delete	DeleteAssay()

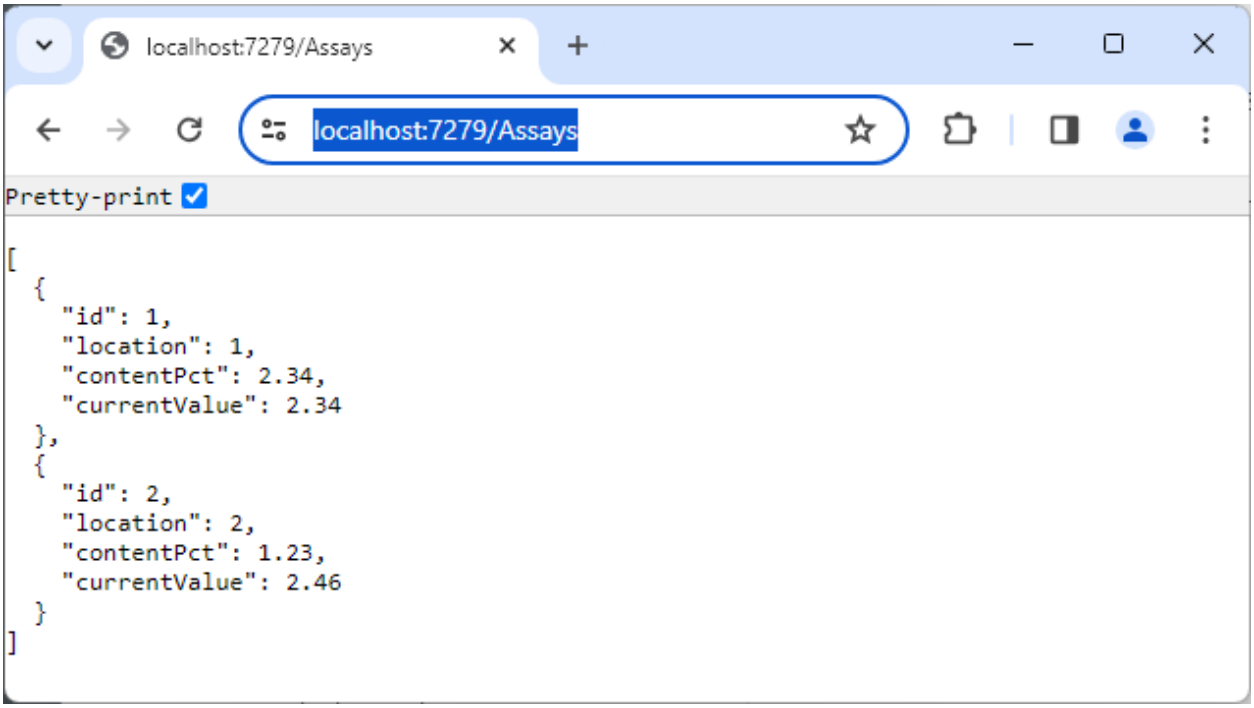


The launchSettings.json in the .Net project cause the default browser to illustrate the results of the REST web service query: localhost:7279/Assays to display all existing Assay records on the server.

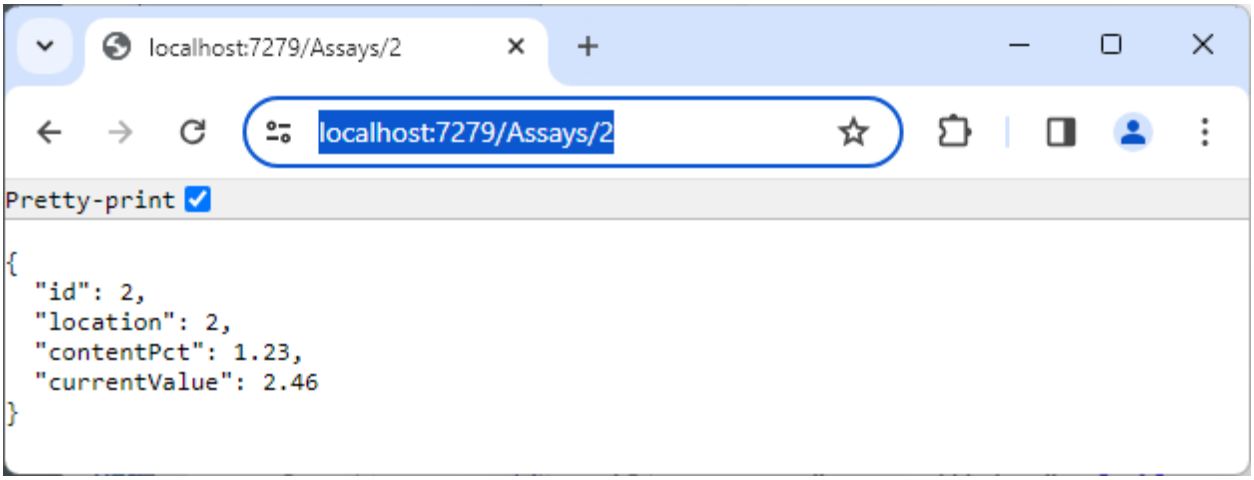
If the default browser indicates that the 'localhost:7279/Assays' site is insecure, select the 'http' option instead of the 'https' option in Visual Studio to start the server and see the 'localhost:5198/Assay' web page in the browser.

Access the REST Service from a Browser

localhost:7279/Assays



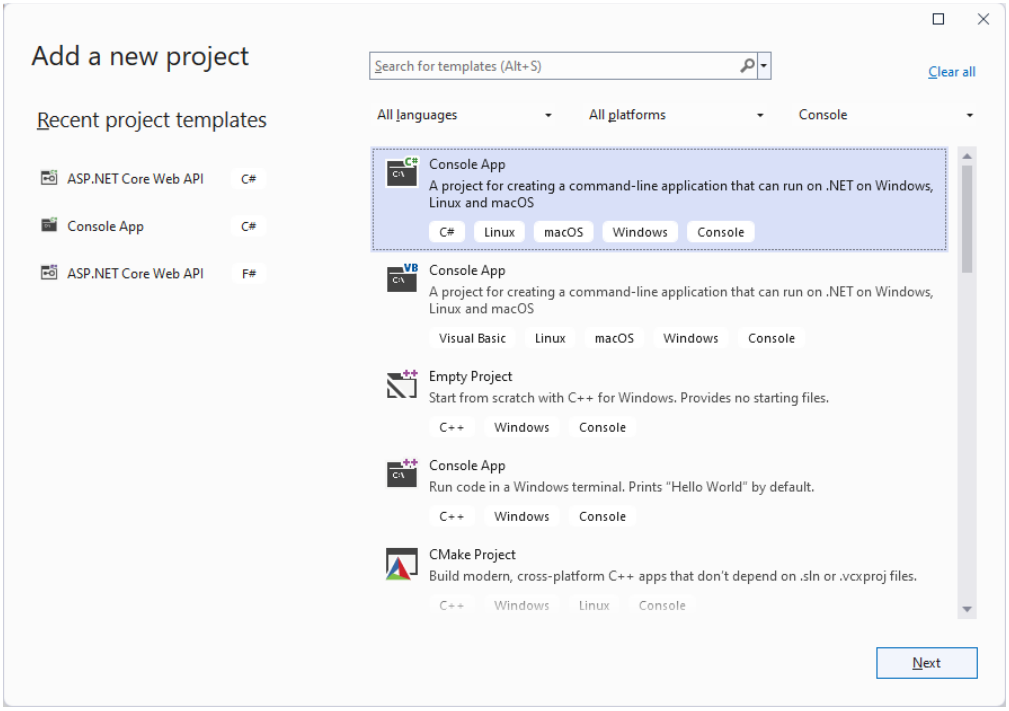
localhost:7279/Assays/2



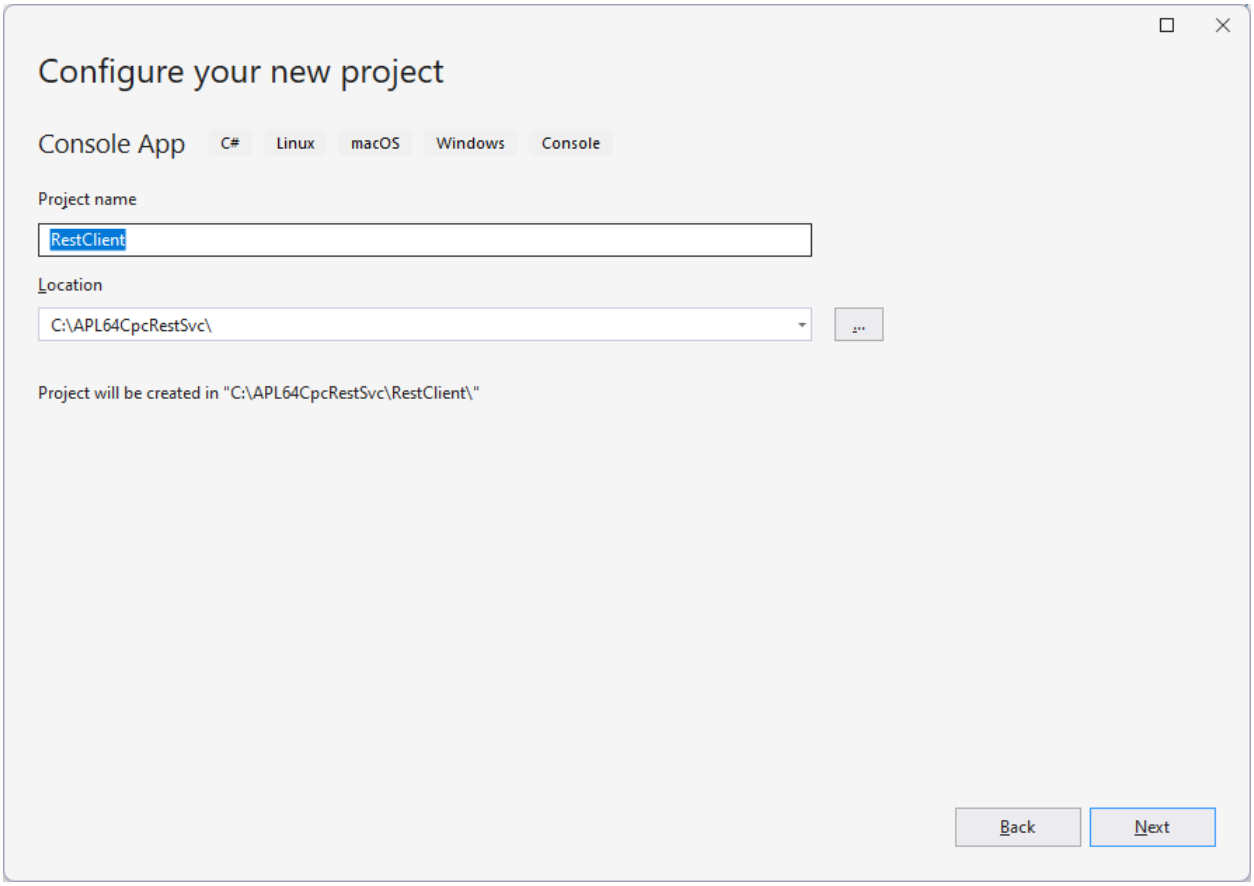
Create a .Net Client for the Rest Web Service

Add a new Project to the Solution called ‘RestClient’

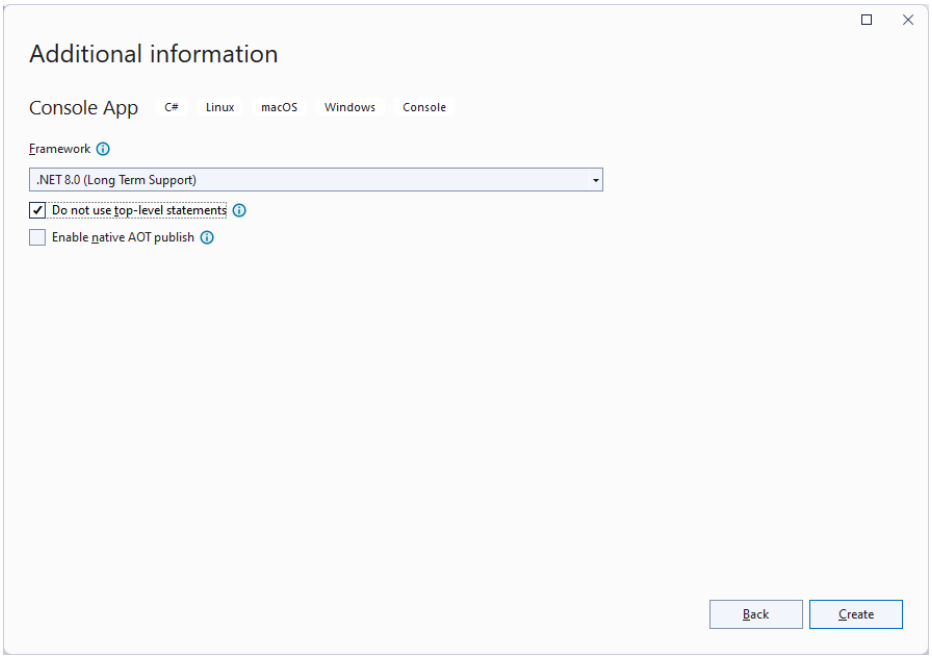
Select the Visual Studio template:



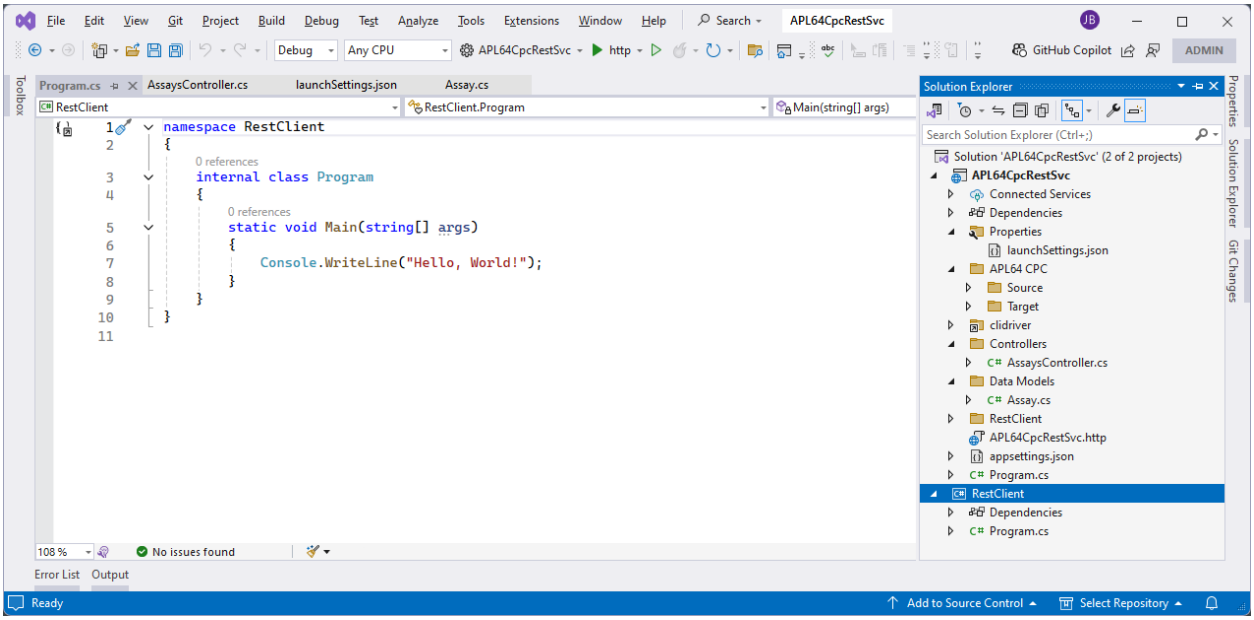
Specify the ‘RestClient’ project name



Provide the Additional Information

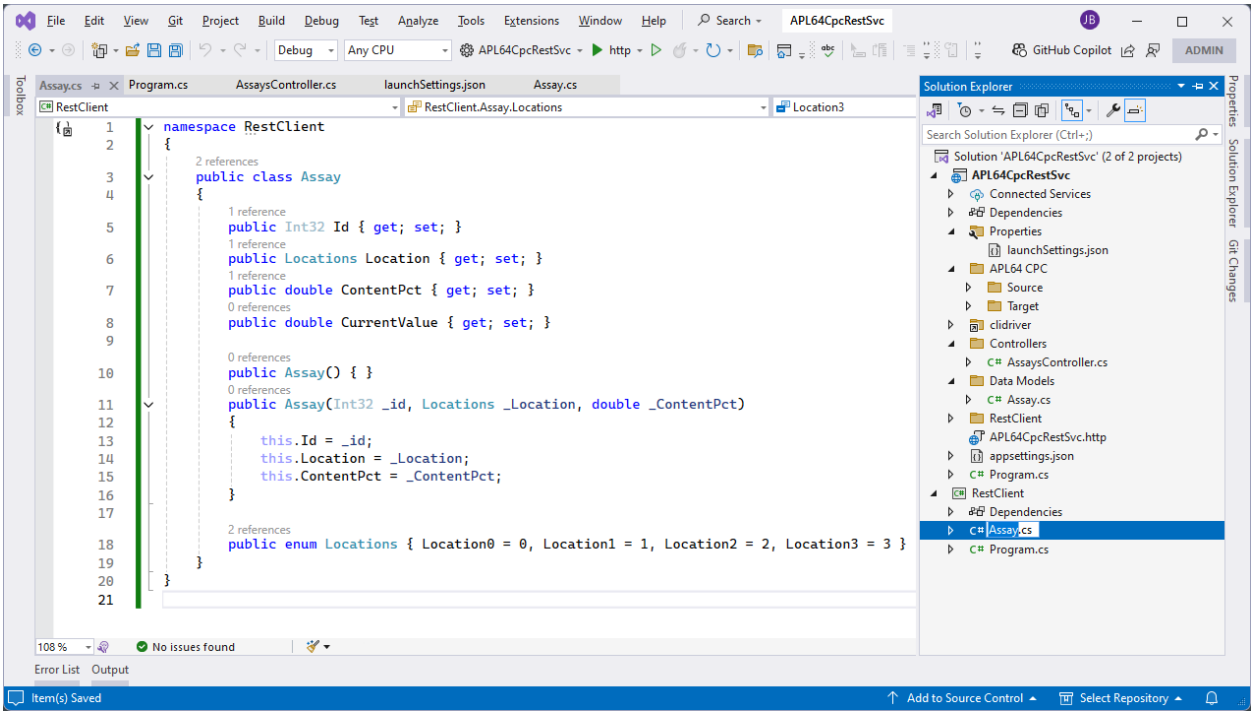


The RestClient project will now be visible in the Visual Studio Solution Explorer:



Add an Assay.cs class to the RestClient project with this code:

```
namespace RestClient
{
  public class Assay
  {
    public Int32 Id { get; set; }
    public Locations Location { get; set; }
    public double ContentPct { get; set; }
    public double CurrentValue { get; set; }

    public Assay() { }
    public Assay(Int32 _id, Locations _Location, double _ContentPct)
    {
      this.Id = _id;
      this.Location = _Location;
      this.ContentPct = _ContentPct;
    }

    public enum Locations { Location0 = 0, Location1 = 1, Location2 = 2, Location3 = 3 }
  }
}
```



Replace the code in the Program.cs file of the RestClient project with this code:

```
using System.Net.Http.Headers;
using System.Net.Http.Json;
namespace RestClient
{
  internal class Program
  {
    static void Main(string[] args)
    {
      var hC = new HttpClientForRestSvc();
      hC.CreateHttpClient(new TimeSpan(0, 0, 20), "https://localhost:7279/assays");
      //^ Use one HttpClient for all requests

      Console.WriteLine("Get all existing Assays");
      hC.GetAllAssays().Wait();
      if (hC.hasError)
        Console.WriteLine($"GetAllAssays failed: {hC.errMsg}");
      else
        Console.WriteLine($"GetAllAssays result:\r\n {hC.jsonResponse}");

      Console.WriteLine();
      Console.WriteLine("Get Assays for Id=2");
      hC.GetAssay(2).Wait();
      if (hC.hasError)
        Console.WriteLine($"GetAssay(2) failed: {hC.errMsg}");
      else
```

```
            Console.WriteLine($"GetAssay(2) result:\r\n {hC.jsonResponse}");

        Console.WriteLine();
        Console.WriteLine("Create new Assay for Id=3, Location=Location3, ContentPct=4.33");
        hC.CreateAssay(3, Assay.Locations.Location3, 4.33).Wait();
        if (hC.hasError)
          Console.WriteLine($"CreateAssay(3, Assay.Locations.Location3, 4.33) failed:
{hC.errMsg}");
else
          Console.WriteLine($"CreateAssay(3, Assay.Locations.Location3, 4.33) result:\r\n
{hC.jsonResponse}");

        Console.WriteLine();
        Console.WriteLine("Update Assay for Id=3 , Location=Location1, ContentPct=2.22");
        hC.UpdateAssay(3, Assay.Locations.Location1, 2.22).Wait();
        if (hC.hasError)
          Console.WriteLine($"UpdateAssay(3) failed: {hC.errMsg}");

        Console.WriteLine("Get all existing Assays");
        hC.GetAllAssays().Wait();
        if (hC.hasError)
          Console.WriteLine($"GetAllAssays failed: {hC.errMsg}");
        else
          Console.WriteLine($"GetAllAssays result:\r\n {hC.jsonResponse}");

        Console.WriteLine();
        Console.WriteLine("Delete Assay for Id=3");
        hC.DeleteAssay(3).Wait();
        if (hC.hasError)
          Console.WriteLine($"DeleteAssay(3) failed: {hC.errMsg}");

        Console.WriteLine("Get all existing Assays");
        hC.GetAllAssays().Wait();
        if (hC.hasError)
          Console.WriteLine($"GetAllAssays failed: {hC.errMsg}");
        else
          Console.WriteLine($"GetAllAssays result:\r\n {hC.jsonResponse}");

        hC.DisposeHttpClient();
        Console.ReadLine();
      }
    }

    public class HttpClientForRestSvc
    {
      private HttpClient client;
      public string errMsg = "";
      public bool hasError;
```

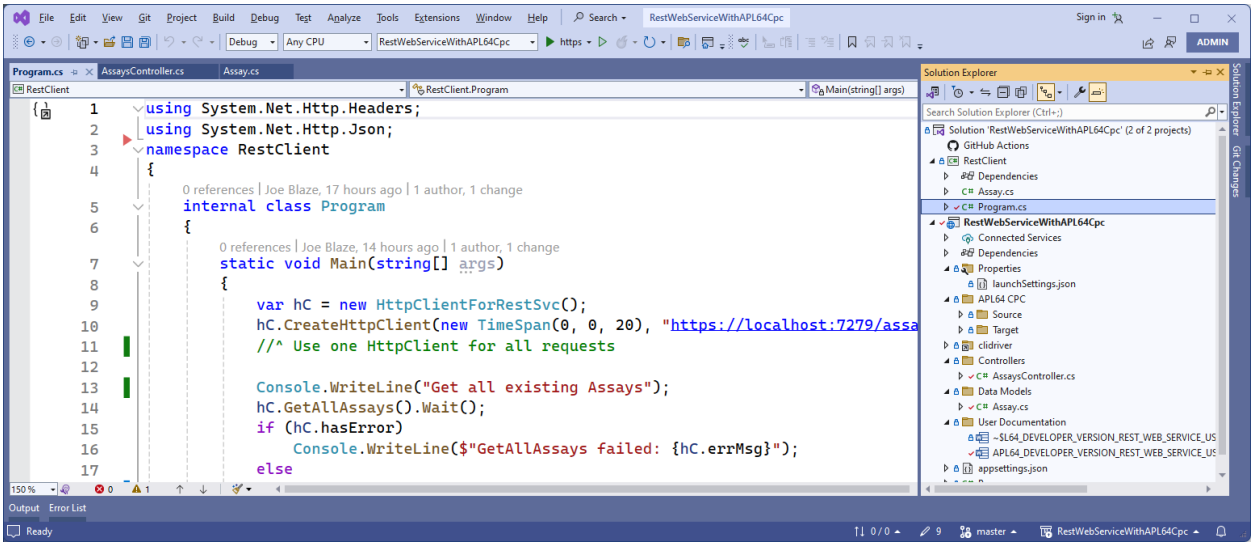
```
    public string jsonResponse = "";
    public string returnUrl = "";

    public HttpClientForRestSvc() { }

    public void CreateHttpClient(TimeSpan httpClientTimeout, string baseAddress)
    {
      client = new HttpClient();
      client.Timeout = httpClientTimeout;
      client.BaseAddress = new Uri(baseAddress);
      client.DefaultRequestHeaders.Accept.Clear();
      client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
    }

    public void DisposeHttpClient()
    {
      if (client != null) { client.Dispose(); }
    }

    public async Task GetAllAssays()
    {
      hasError = true;
      errMsg = string.Empty;
      jsonResponse = string.Empty;
      try
      {
        var response = await client.GetAsync("Assays");
        response.EnsureSuccessStatusCode();
        if (!response.IsSuccessStatusCode)
          errMsg = $"Status Code: {response.StatusCode} Reason
Phrase:{response.ReasonPhrase}";
        else
        {
          hasError = false;
          jsonResponse = await response.Content.ReadAsStringAsync();
        }
      }
      catch (OperationCanceledException ex) when (ex.InnerException is TimeoutException tex)
      {
        errMsg = $"Timed out: {ex.Message}, {tex.Message}";
        return;
      }
      catch (OperationCanceledException ex)
      {
        if (ex.InnerException != null)
          errMsg = $"{ex.Message}, {ex.InnerException.Message}";
        else
```

```
            errMsg = ex.Message;
          return;
        }
        catch (Exception ex)
        {
          errMsg = ex.Message;
          return;
        }
      }
      public async Task GetAssay(Int32 id)
      {
        hasError = true;
        errMsg = string.Empty;
        jsonResponse = string.Empty;
        try
        {
          var response = await client.GetAsync($"Assays/{id}");
          response.EnsureSuccessStatusCode();
          if (!response.IsSuccessStatusCode)
            errMsg = $"Status Code: {response.StatusCode} Reason
Phrase:{response.ReasonPhrase}";
          else
          {
            hasError = false;
            jsonResponse = await response.Content.ReadAsStringAsync();
          }
        }
        catch (OperationCanceledException ex) when (ex.InnerException is TimeoutException tex)
        {
          errMsg = $"Timed out: {ex.Message}, {tex.Message}";
          return;
        }
        catch (OperationCanceledException ex)
        {
          if (ex.InnerException != null)
            errMsg = $"{ex.Message}, {ex.InnerException.Message}";
          else
            errMsg = ex.Message;
          return;
        }
        catch (Exception ex)
        {
          errMsg = ex.Message;
          return;
        }
      }
      public async Task GetOptions()
      {
```

```
        hasError = true;
        errMsg = string.Empty;
        jsonResponse = string.Empty;

        try
        {
          using HttpRequestMessage request = new(HttpMethod.Options, "Assays");
          using HttpResponseMessage response = await client.SendAsync(request);

          if (!response.IsSuccessStatusCode)
            errMsg = $"Status Code: {response.StatusCode} Reason
Phrase:{response.ReasonPhrase}";
          else
          {
            hasError = false;
            jsonResponse = await response.Content.ReadAsStringAsync();
          }
        }
        catch (OperationCanceledException ex) when (ex.InnerException is TimeoutException tex)
        {
          errMsg = $"Timed out: {ex.Message}, {tex.Message}";
          return;
        }
        catch (OperationCanceledException ex)
        {
          if (ex.InnerException != null)
            errMsg = $"{ex.Message}, {ex.InnerException.Message}";
          else
            errMsg = ex.Message;
          return;
        }
        catch (Exception ex)
        {
          errMsg = ex.Message;
          return;
        }
      }

      public async Task CreateAssay(Int32 id, Assay.Locations location, double contentPct)
      {
        hasError = true;
        errMsg = string.Empty;
        jsonResponse = string.Empty;
        var Assay = new Assay(id, location, contentPct);
        try
        {
          var response = await client.PostAsJsonAsync($"Assays", Assay);
          response.EnsureSuccessStatusCode();
```

```
            if (!response.IsSuccessStatusCode)
              errMsg = $"Status Code: {response.StatusCode} Reason
Phrase:{response.ReasonPhrase}";
            else
            {
              hasError = false;
              var todo = await response.Content.ReadFromJsonAsync<Assay>();
              jsonResponse = await response.Content.ReadAsStringAsync();
            }
          }
          catch (OperationCanceledException ex) when (ex.InnerException is TimeoutException tex)
          {
            errMsg = $"Timed out: {ex.Message}, {tex.Message}";
            return;
          }
          catch (OperationCanceledException ex)
          {
            if (ex.InnerException != null)
              errMsg = $"{ex.Message}, {ex.InnerException.Message}";
            else
              errMsg = ex.Message;
            return;
          }
          catch (Exception ex)
          {
            errMsg = ex.Message;
            return;
          }
        }

        public async Task UpdateAssay(Int32 id, Assay.Locations location, double contentPct)
        {
          hasError = true;
          errMsg = string.Empty;
          jsonResponse = string.Empty;
          var Assay = new Assay(id, location, contentPct);
          try
          {
            var response = await client.PutAsJsonAsync($"Assays/{id}", Assay);
            response.EnsureSuccessStatusCode();
            if (response.IsSuccessStatusCode) hasError = false;
          }
          catch (OperationCanceledException ex) when (ex.InnerException is TimeoutException tex)
          {
            errMsg = $"Timed out: {ex.Message}, {tex.Message}";
            return;
          }
          catch (OperationCanceledException ex)
```

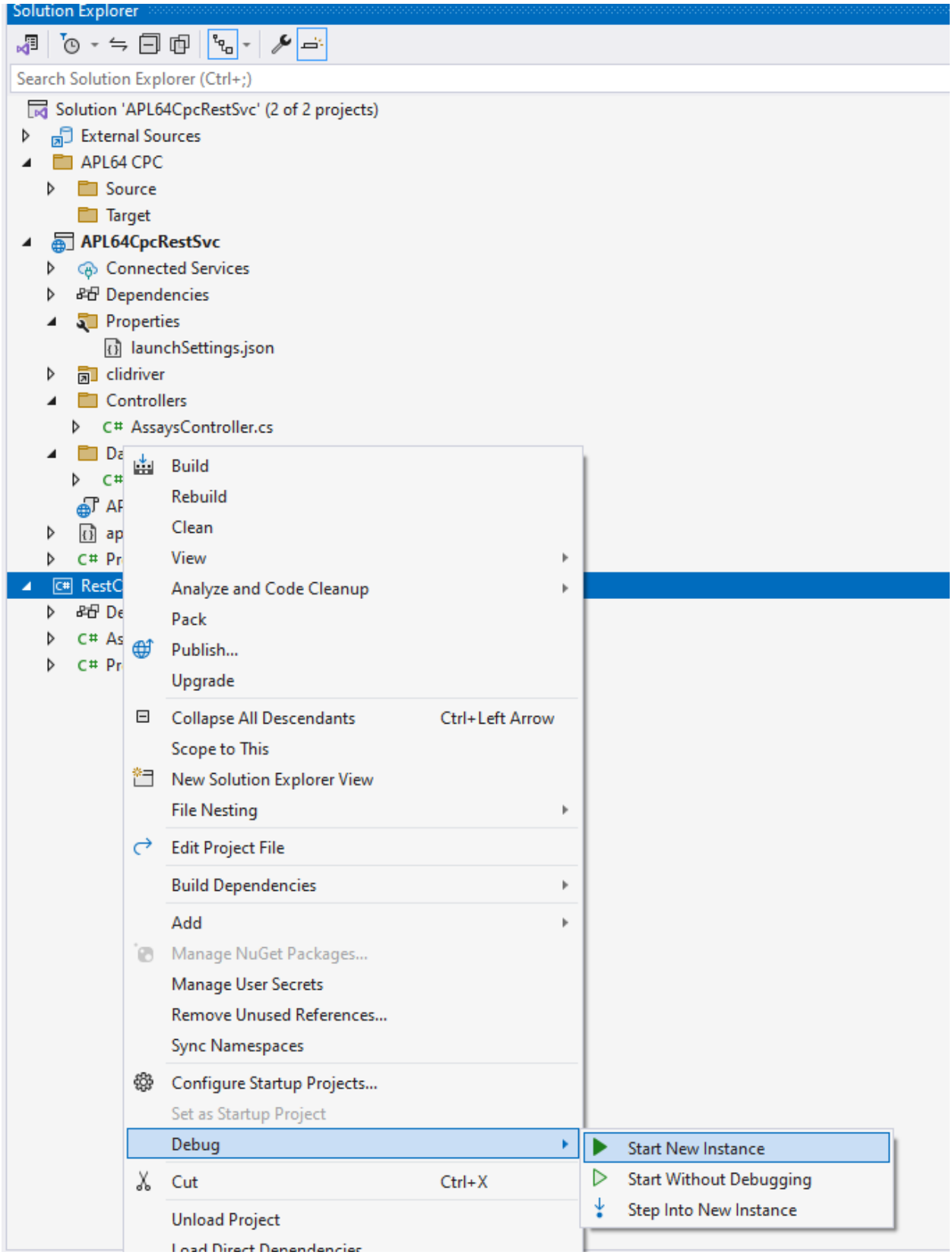
```
        {
            if (ex.InnerException != null)
                errMsg = $"{ex.Message}, {ex.InnerException.Message}";
            else
                errMsg = ex.Message;
            return;
        }
        catch (Exception ex)
        {
            errMsg = ex.Message;
            return;
        }
    }

    public async Task DeleteAssay(Int32 id)
    {
        hasError = true;
        errMsg = string.Empty;
        jsonResponse = string.Empty;
        try
        {
            var response = await client.DeleteAsync($"Assays/{id}");
            response.EnsureSuccessStatusCode();
            if (response.IsSuccessStatusCode) hasError = false;
        }
        catch (OperationCanceledException ex) when (ex.InnerException is TimeoutException tex)
        {
            errMsg = $"Timed out: {ex.Message}, {tex.Message}";
            return;
        }
        catch (OperationCanceledException ex)
        {
            if (ex.InnerException != null)
                errMsg = $"{ex.Message}, {ex.InnerException.Message}";
            else
                errMsg = ex.Message;
            return;
        }
        catch (Exception ex)
        {
            errMsg = ex.Message;
            return;
        }
    }
  }
}
```

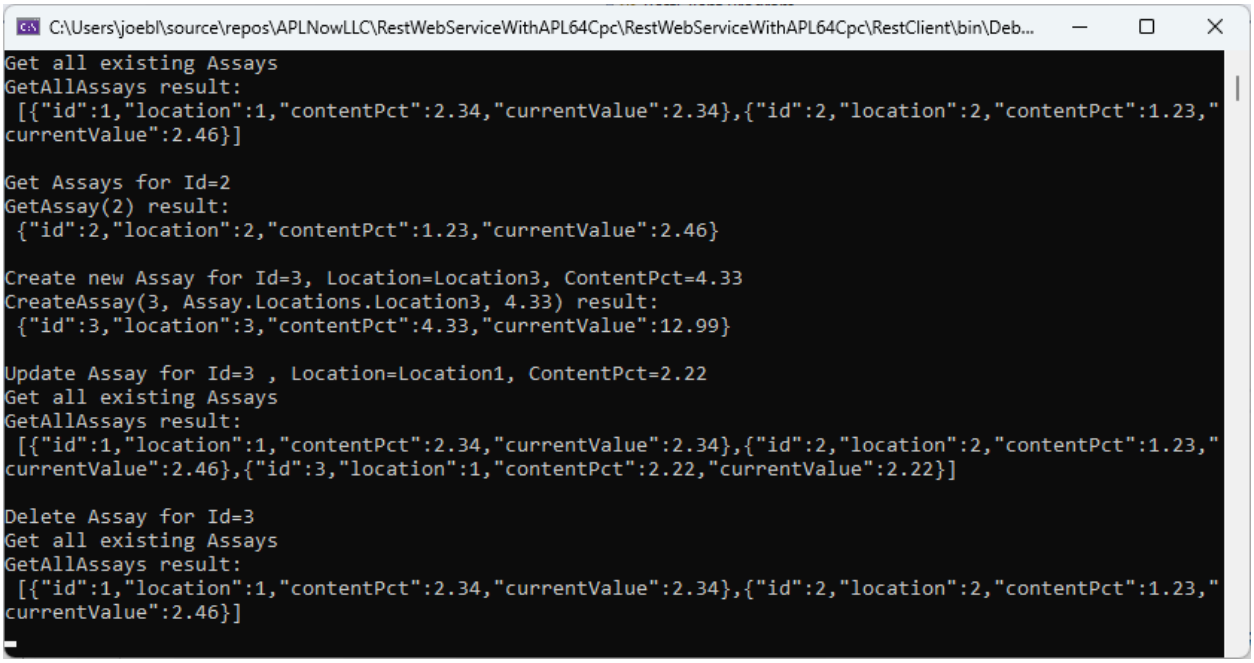


Run the RestClient Project

With the RestWebService project running, debug the RestClient project in Visual Studio to observe the CRUD operations, including the use of the APL64 programmer-defined, public CurrentValue function in the APL64 CPC AssayClass.

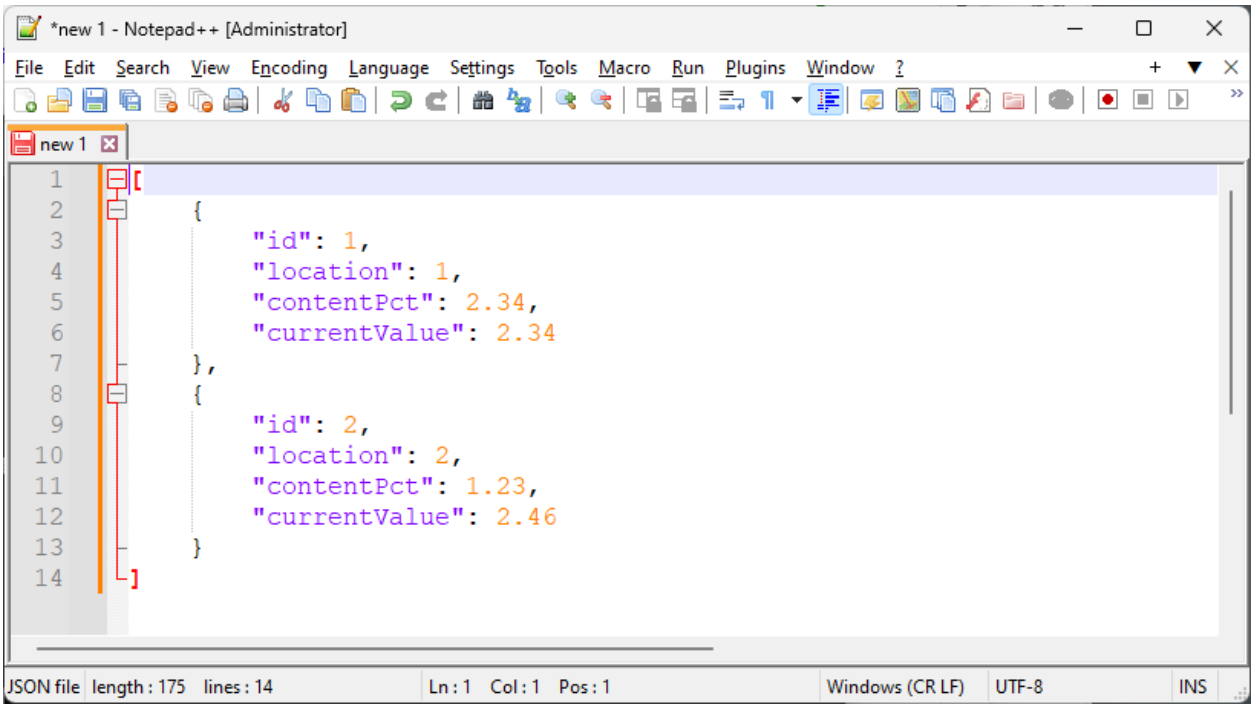
Right click the RestClient project in the Visual Studio Solution Explorer, select Debug | Start New Instance:



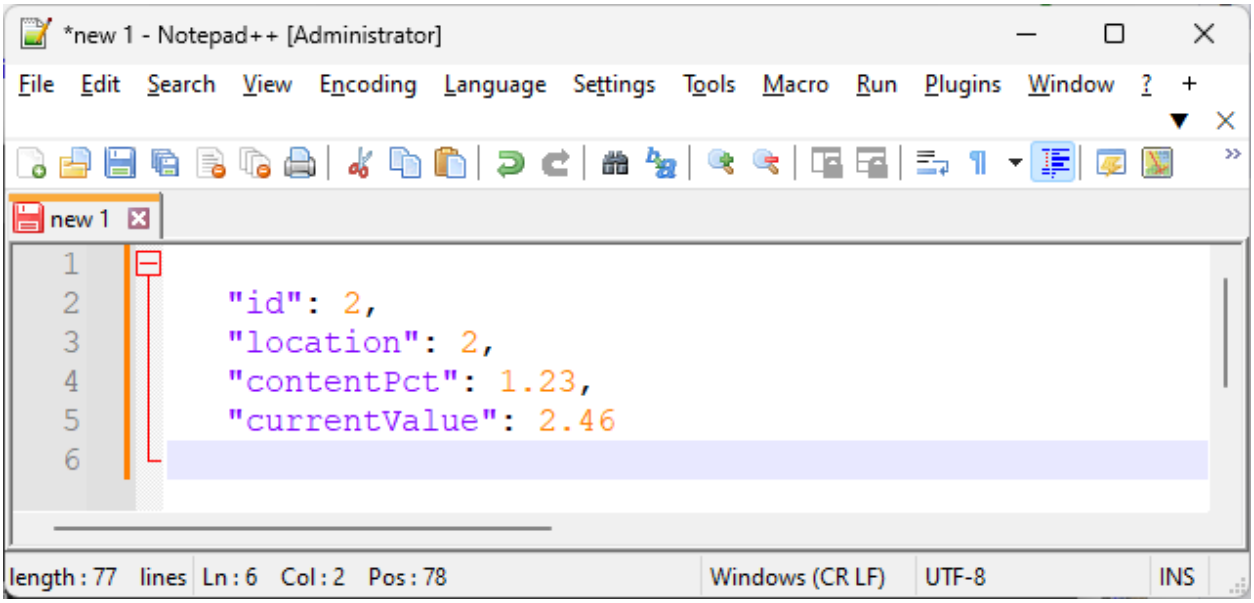


The output of the RestClient illustrates the results of the 'CRUD' operations on the server-side data source. The 'currentValue' element is computed using the APL64 CPC. The results are presented in json format. Copying the results from the browser and displaying them in Notepad++:

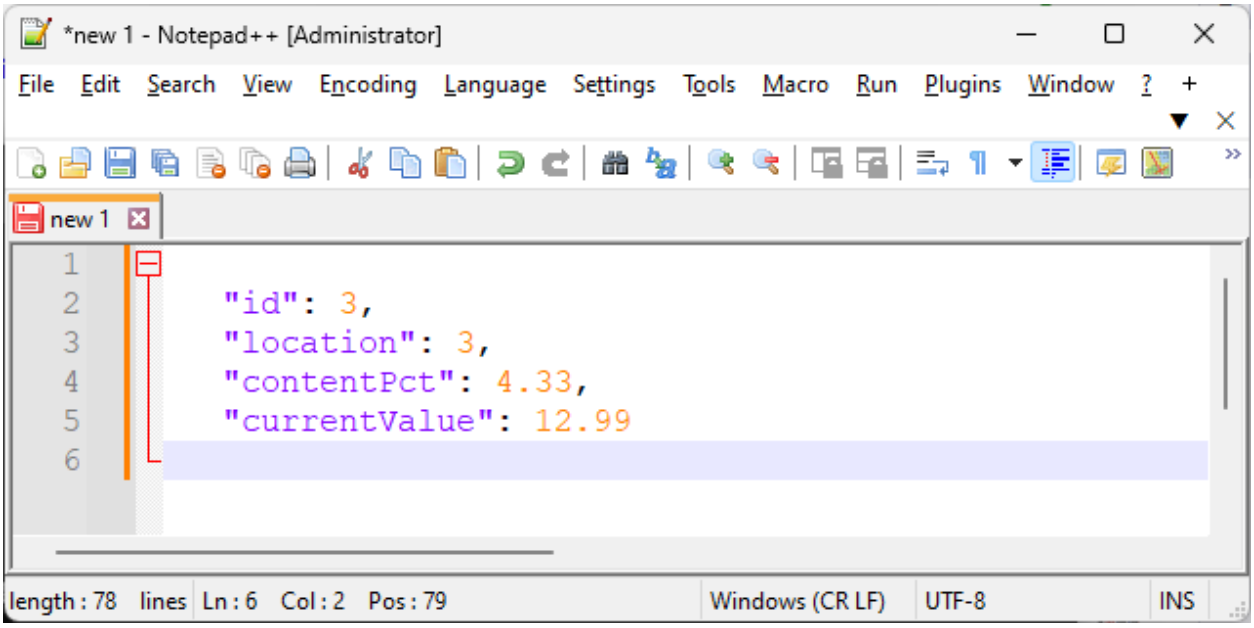
Get All Existing Assays:



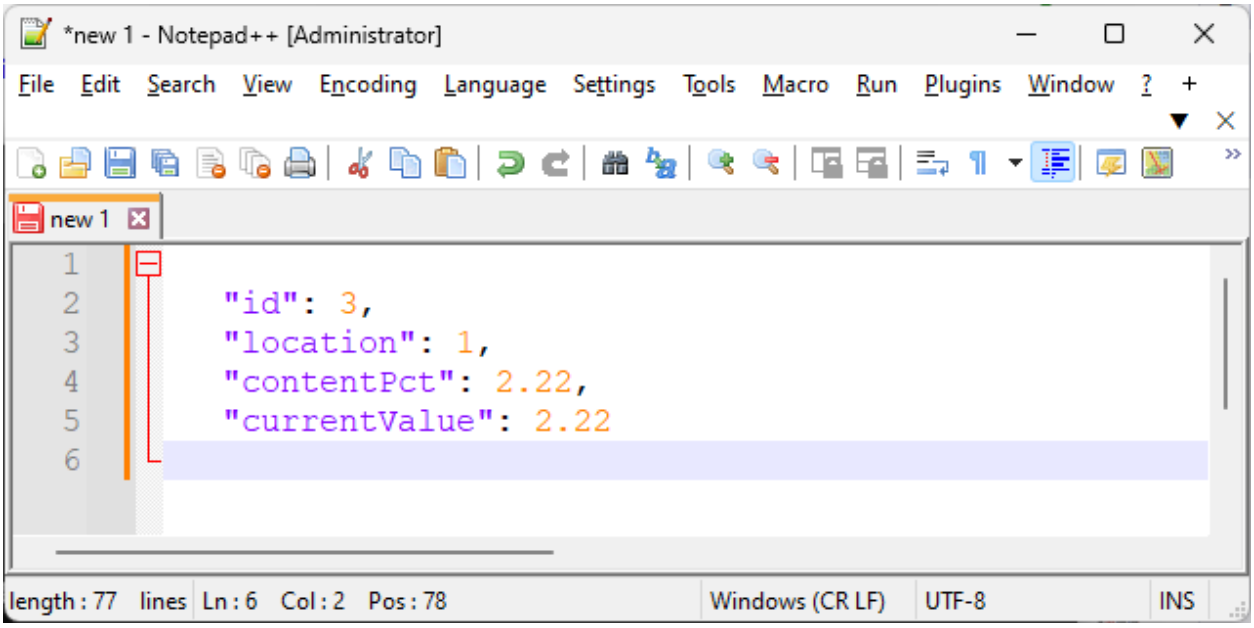
Get Assays for Id=2



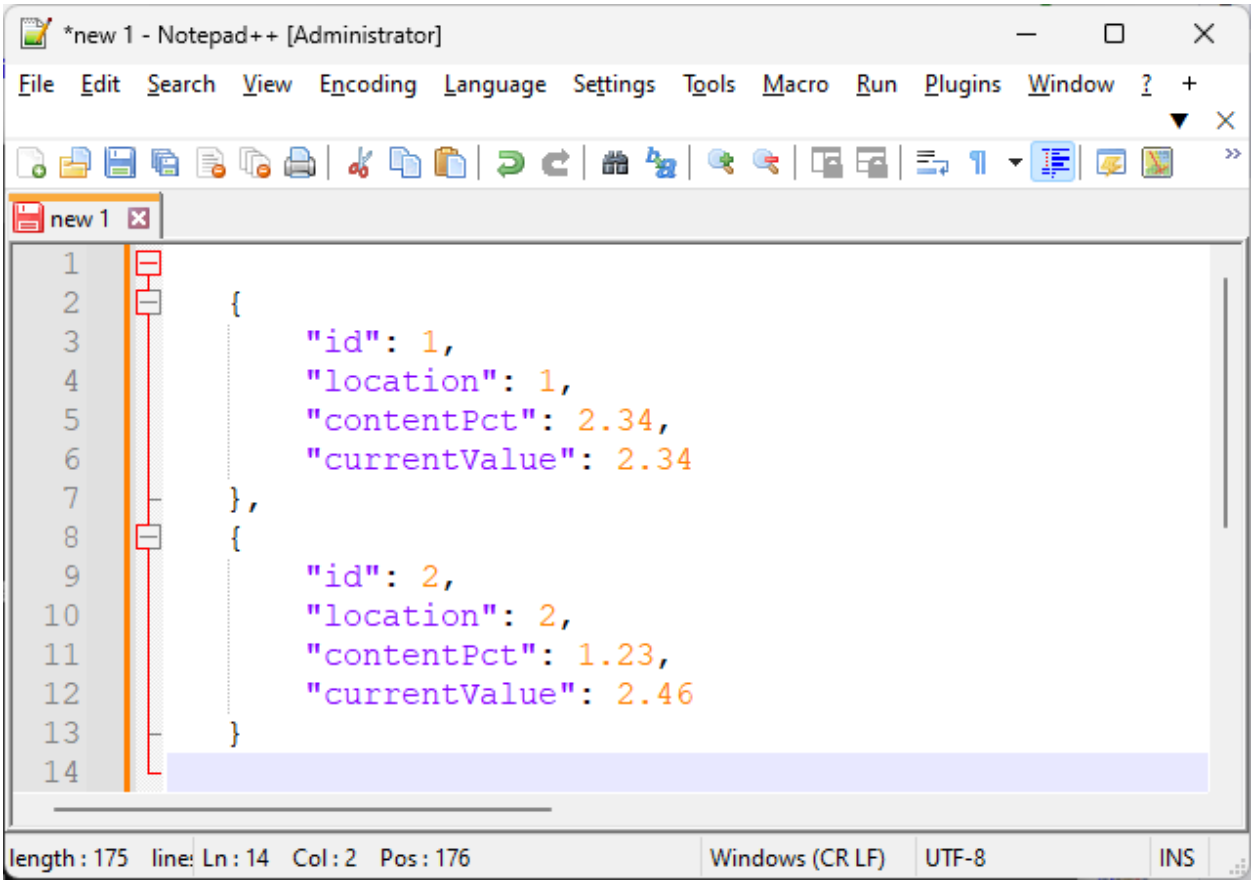
Create new Assay for Id=3, Location=Location3, ContentPct=4.33:



Update Assay for Id=3 , Location=Location1, ContentPct=2.22:



Delete Assay for Id=3 and Get all existing Assays:



Learn More

The REST web service example described in this document is simplified so that basic concepts are emphasized.

The APL64 cross-platform component could be used for additional purposes in the REST web service to support other REST actions.

For production use, a rest web server and an HttpClient must incorporate client authentication, a responsive and resilient data base and other important features. For more information about making HTTP client requests, see here.

The RestClient project in this example is a .Net Console project. This RestClient project provides an appropriate design if the REST web service is to be from a client web server.

An alternative REST web service client could be an HTML page containing an input form suitable for a web browser-based application.

To obtain an APL64 subscription or for customized consulting, contact sales@apl2000.com

For APL64 subscribers, contact support@apl2000.com for technical assistance.