

REST Web Service using APL64 Cross-platform Component

Table of Contents

Overview	2
Download Sample Solution	2
Create Rest Web Service	3
Create a New Visual Studio Project	3
Select the project template	3
Configure the REST service project	4
Modify the RestWebService project C# code	5
Create the APL64 Cross-platform Component	10
Create the public function 'CurrentValue' in the 'Assay' workspace	11
Use the APL64 CPC Utility	13
Complete the CPC Utility dialog	13
Save the CPC information file to the Source folder	14
Create the CPC in the Target folder	14
Publish the CPC Nuget Package Locally	15
Install the CPC 'Assay' Nuget Package into the Project	16
Set x64 Build Configuration for Rest Service Project	17
Start the REST Service	18
Access the REST Service from a Browser	19
Create a .Net Client for the Rest Web Service	21
Add a new Project to the Solution called 'RestClient'	21
Select the Visual Studio template:	21
Specify the 'RestClient' project name	22
Provide the Additional Information	22
Add an Assay.cs class to the RestClient project with this code:	23
Replace the code in the Program.cs file of the RestClient project	24
Run the RestClient Project	31
Learn More	36

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:

- [C](#)reate a new Assay record
- [R](#)ead all or specified Assay record(s)
- [U](#)ppdate an Assay record
- [D](#)elate 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:\.

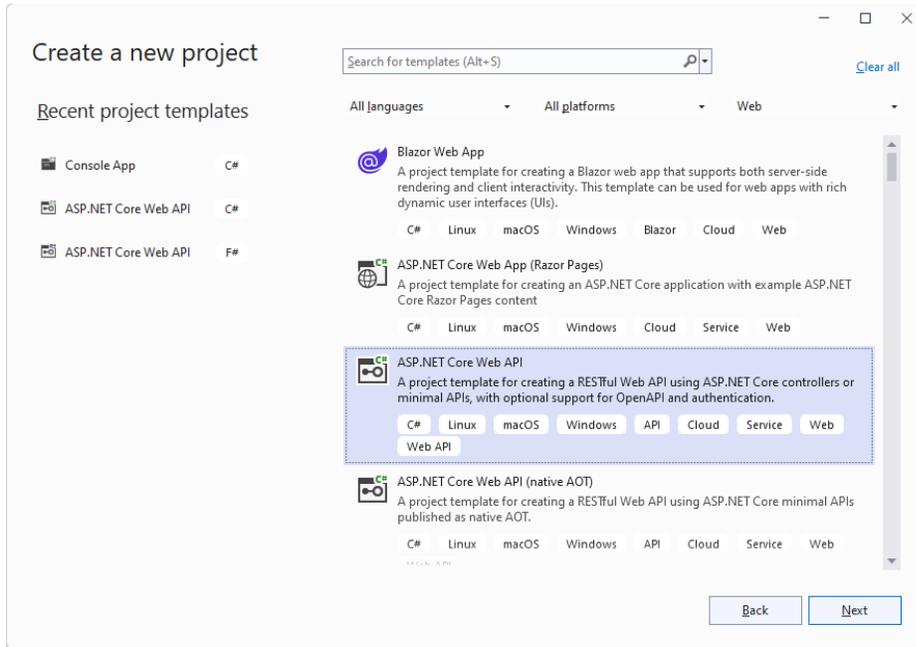
The zip-format file contains a Visual Studio Github repository with:

- 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

Configure your new project

ASP.NET Core Web API C# Linux macOS Windows API Cloud Service Web Web API

Project name

Location
 ..

Solution name ⓘ

Place solution and project in the same directory

Project will be created in "C:\APL64CpcRestSvc\APL64CpcRestSvc\APL64CpcRestSvc\"

Back Next

Provide the additional information

Additional information

ASP.NET Core Web API C# Linux macOS Windows API Cloud Service Web Web API

Framework ⓘ

.NET 8.0 (Long Term Support)

Authentication type ⓘ

None

Configure for HTTPS ⓘ

Enable container support ⓘ

Container OS ⓘ

Linux

Container build type ⓘ

Dockerfile

Enable OpenAPI support ⓘ

Do not use top-level statements ⓘ

Use controllers ⓘ

Enlist in .NET Aspire orchestration ⓘ

Back
Create

Modify the RestWebService project C# code

Delete the WeatherForecast.cs code file

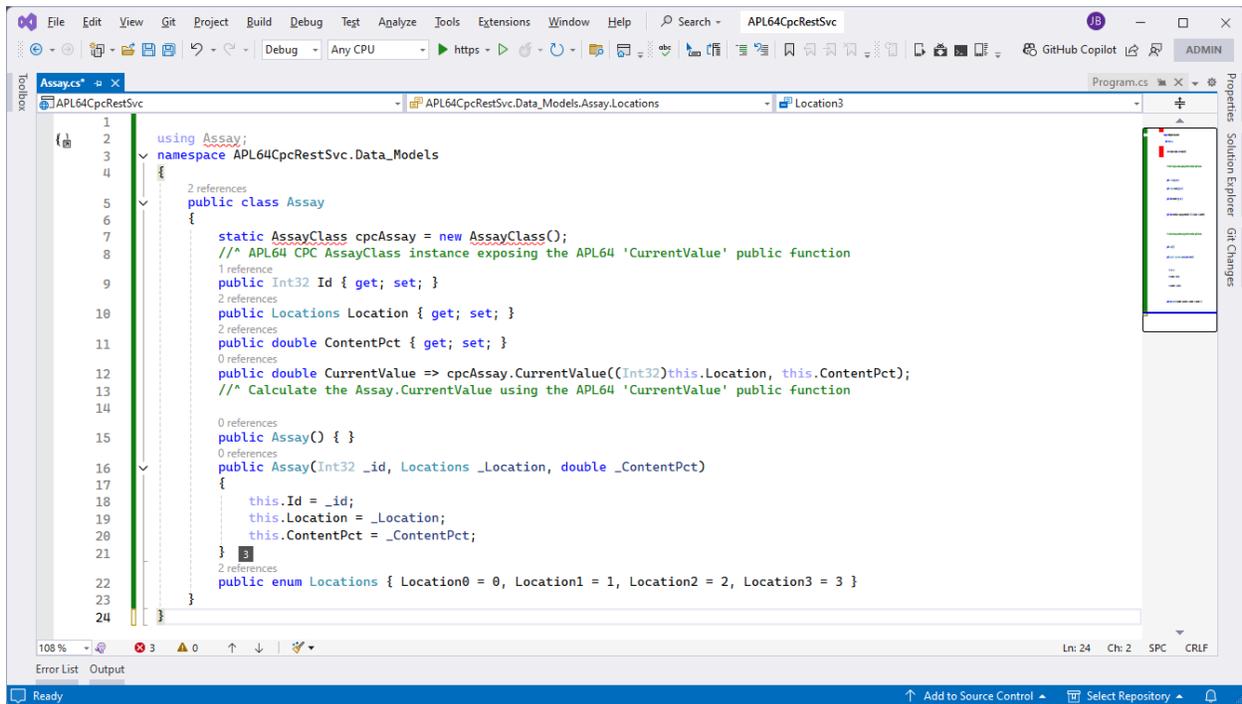
Add the 'Data Models' folder to the APL64CpcRestSvc project

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 }
    }
}
```

```
}  
}
```

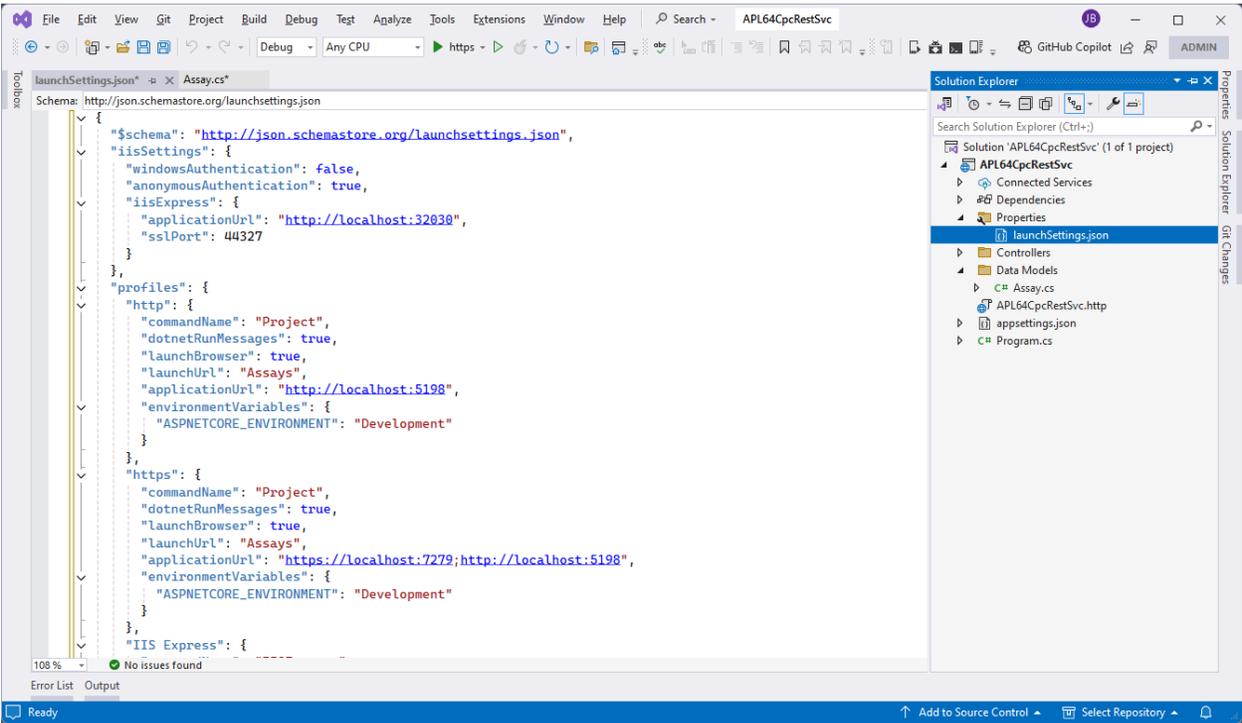


The Assay class definition exposes the applicable properties and methods of an Assay record.

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

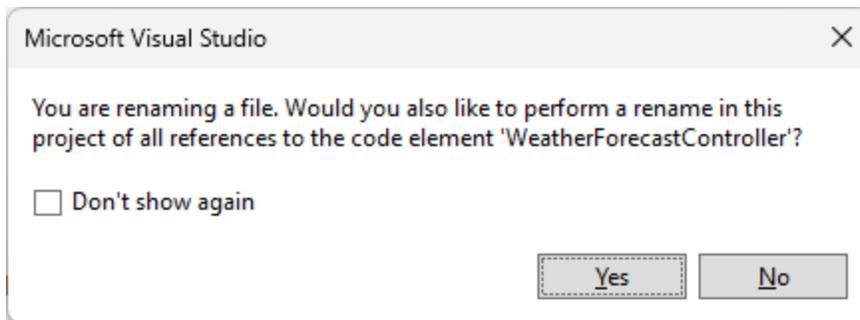
```
{  
  "$schema": "http://json.schemastore.org/launchsettings.json",  
  "iisSettings": {  
    "windowsAuthentication": false,  
    "anonymousAuthentication": true,  
    "iisExpress": {  
      "applicationUrl": "http://localhost:32030",  
      "sslPort": 44327  
    }  
  },  
  "profiles": {  
    "http": {  
      "commandName": "Project",  
      "dotnetRunMessages": true,  
      "launchBrowser": true,  
      "launchUrl": "Assays",  
      "applicationUrl": "http://localhost:5198",  
      "environmentVariables": {  
        "ASPNETCORE_ENVIRONMENT": "Development"  
      }  
    },  
    "https": {  
      "commandName": "Project",  
      "dotnetRunMessages": true,  
      "applicationUrl": "https://localhost:5001",  
      "environmentVariables": {  
        "ASPNETCORE_ENVIRONMENT": "Development"  
      }  
    }  
  }  
}
```

```
"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.cs and 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);
        }
    }
}
```

```

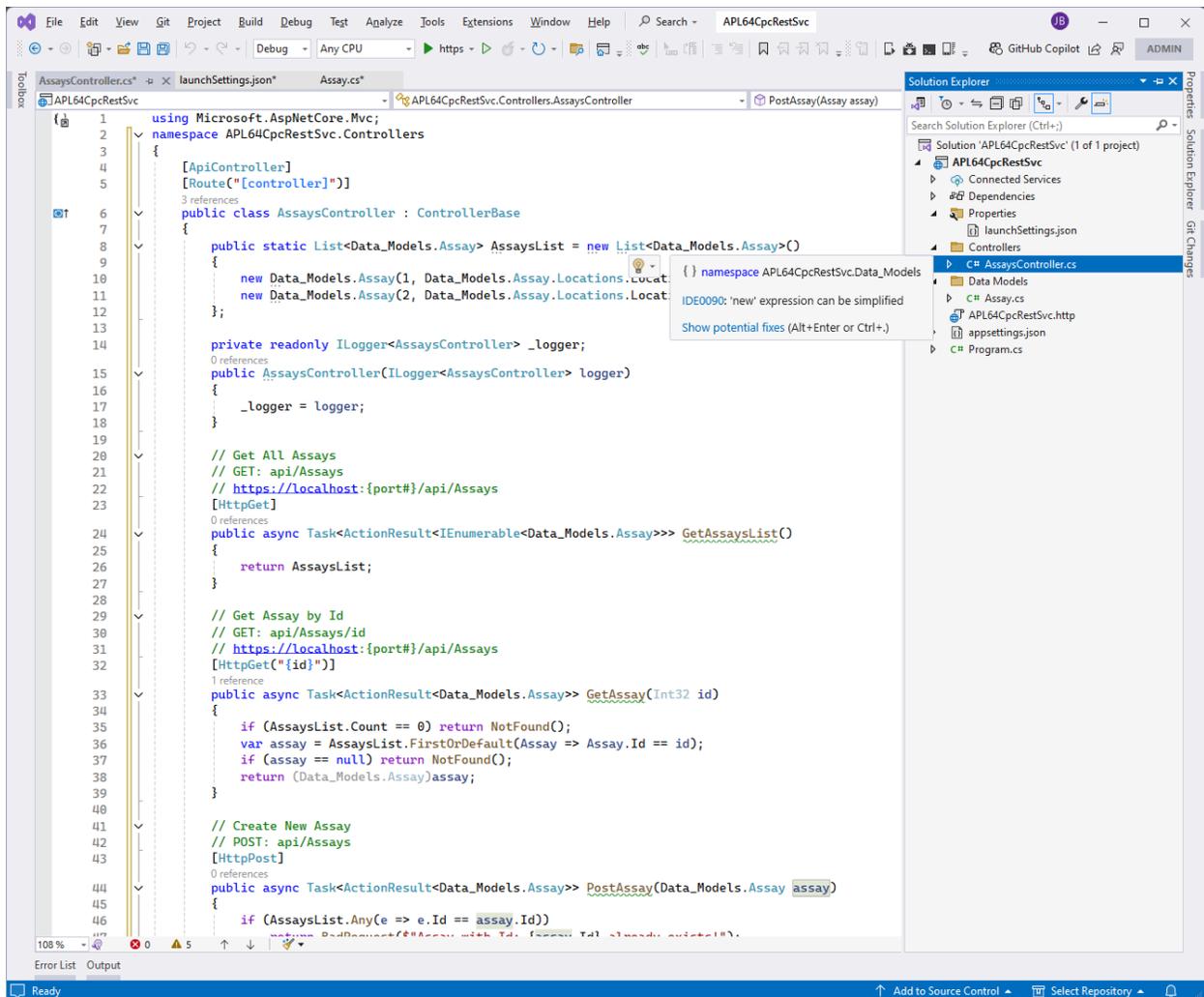
    if (assay == null) return NotFound();
    return (Data_Models.Assay)assay;
}

// 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<ActionResult> 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();
}

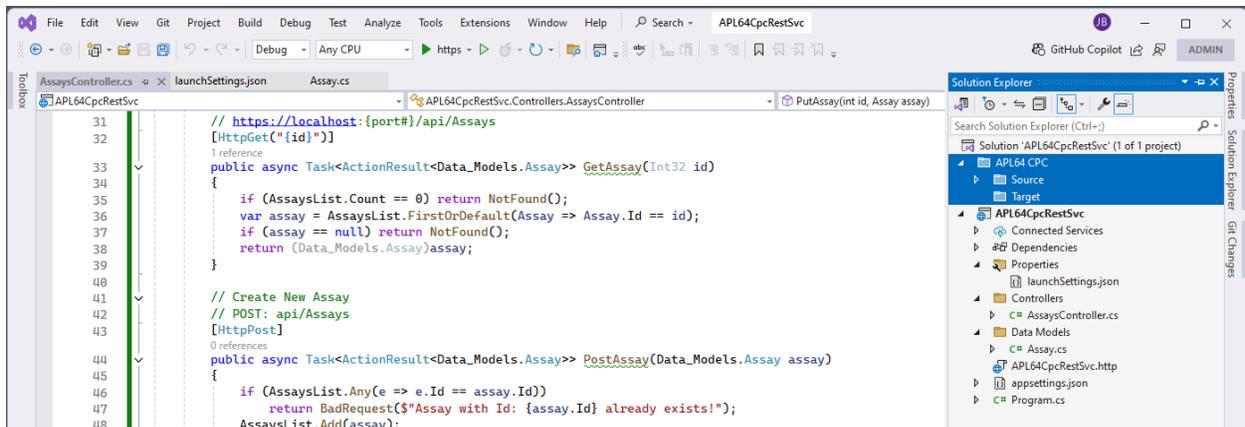
// Delete Assay by Id
// DELETE: api/Assays
[HttpDelete("{id}")]
public async Task<ActionResult> DeleteAssay(Int32 id)
{
    var Assay = AssaysList.FirstOrDefault(elt => elt.Id == id);
    if (Assay == null) return NotFound();
    AssaysList.Remove(Assay);
    return NoContent();
}
}
}
}

```



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

```

```

VCurrentValue
0  :public double@cv+CurrentValue (int@loc;double@pct);LocMult
1  locMult+0
2  :SELECT loc
3  :CASE 0
4    locMult+0
5  :CASE 1
6    locMult+1
7  :CASE 2
8    locMult+2
9  :CASE 3
10   locMult+3
11 :ENDSELECT
12 cv+pct*locMult

```

[12;14]

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

```

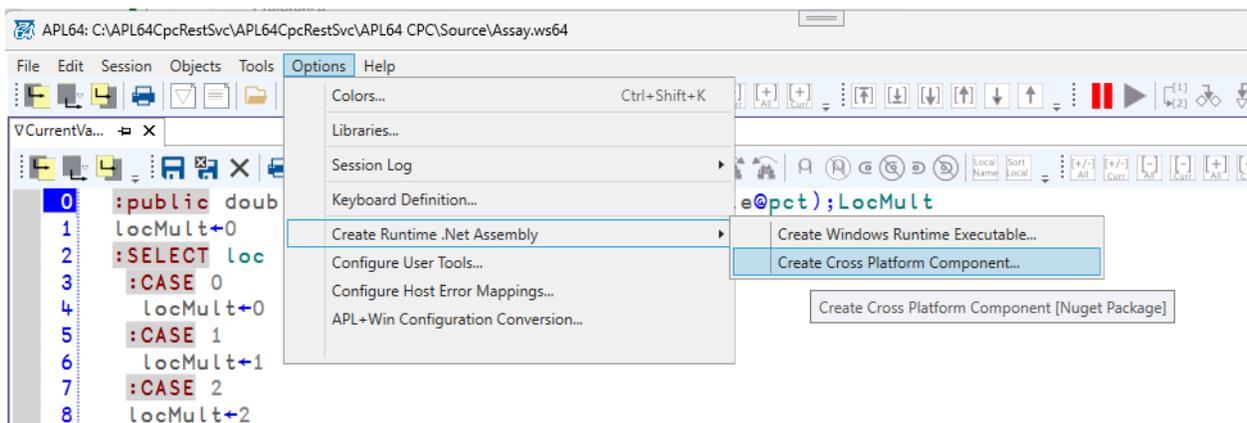
APL64: C:\APL64CpcRestSvc\APL64CpcRestSvc\APL64 CPC\Source\Assay.ws64
File Edit Session Objects Tools Options Help
VCurrentValue... X
0  :public double@cv+CurrentValue (int@loc;double@pct);LocMult
1  locMult+0
2  :SELECT loc
3  :CASE 0
4    locMult+0
5  :CASE 1
6    locMult+1
7  :CASE 2
8    locMult+2
9  :CASE 3
10   locMult+3
11 :ENDSELECT
12 cv+pct*locMult

0  )ed VCurrentValue
1  )save C:\APL64CpcRestSvc\APL64CpcRestSvc\APL64 CPC\Source\Assay.ws64
2  "C:\APL64CpcRestSvc\APL64CpcRestSvc\APL64 CPC\Source\Assay.ws64" SAVED 11/25/2024 7:40:16 AM
3

```

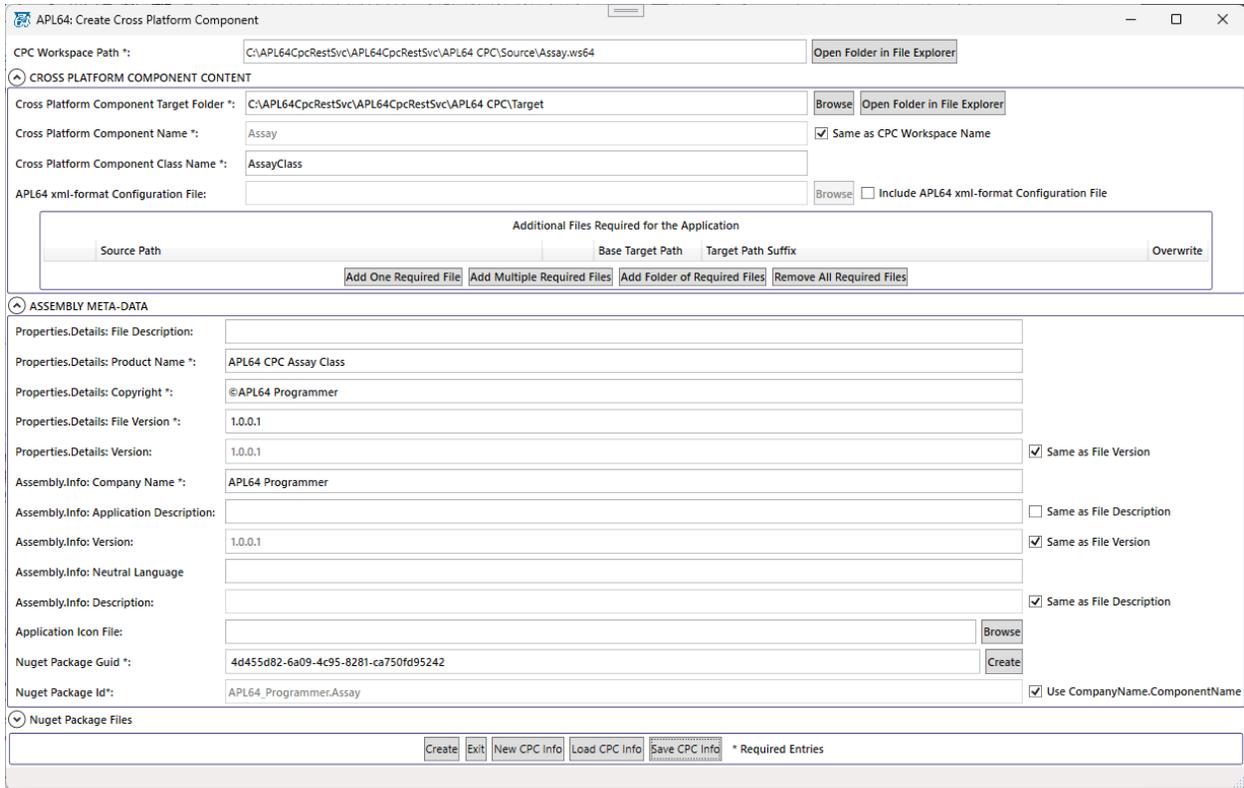
Ready | Hist: Ln: 3 Col: 6 | Ins | Classic | Num | EN_US

Use the APL64 CPC Utility

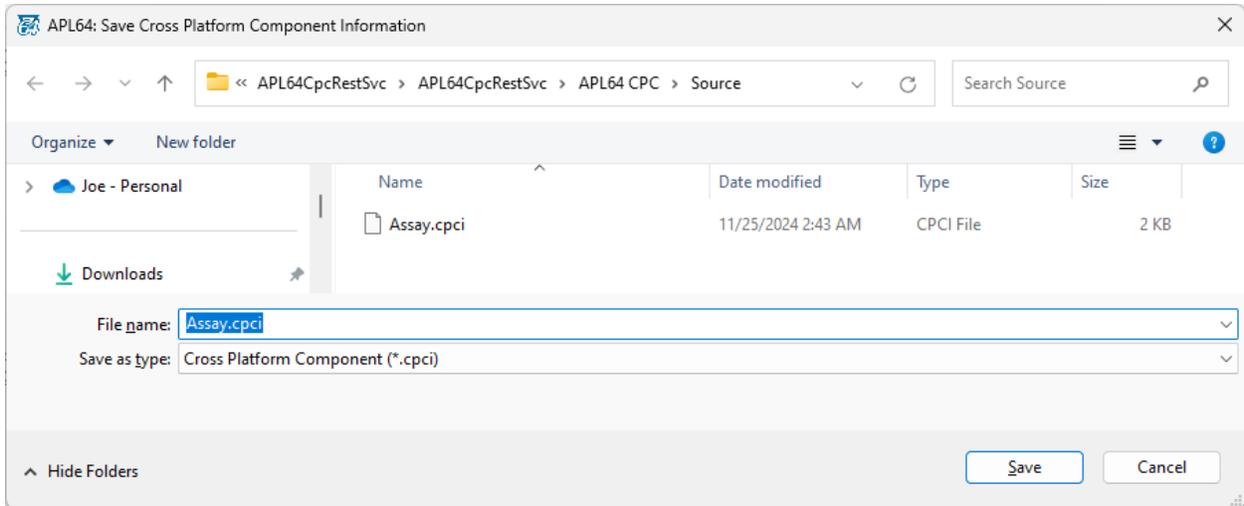


Complete the CPC Utility dialog

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

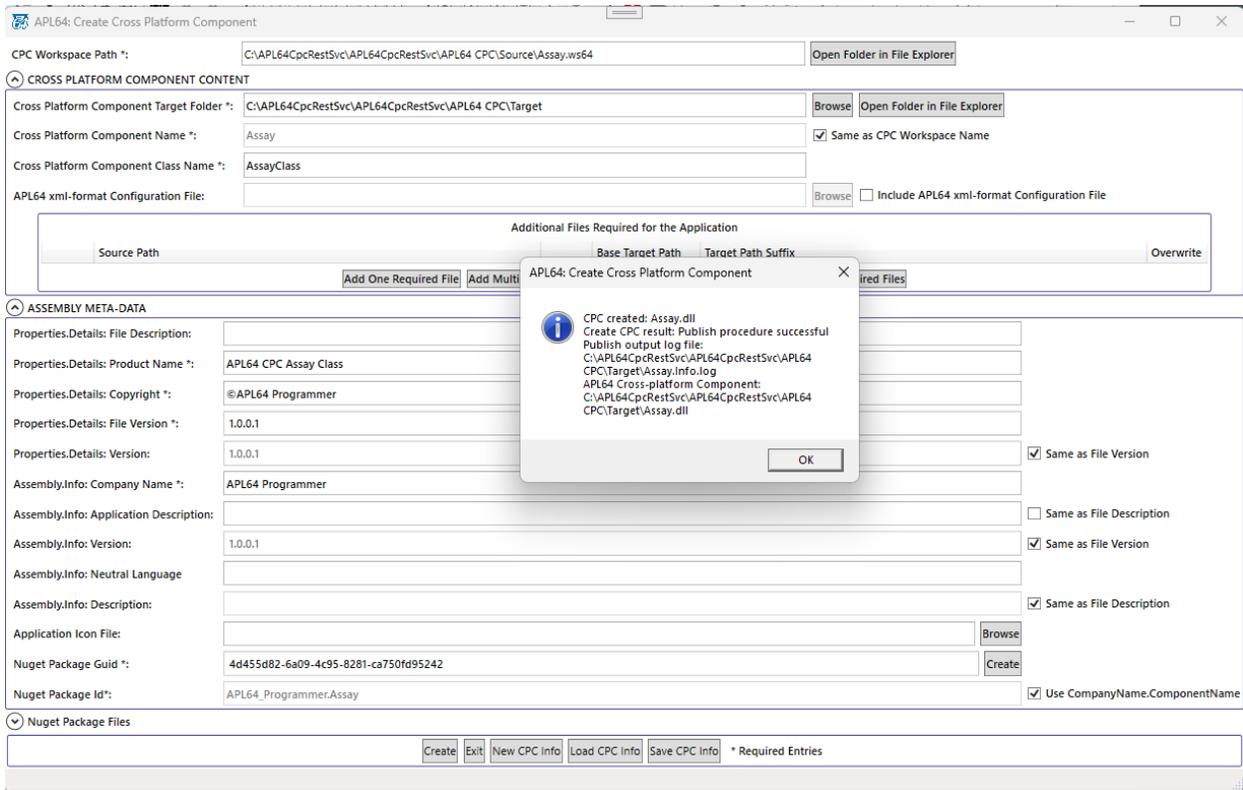


Save the CPC information file to the Source folder



Create the CPC in the Target folder

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

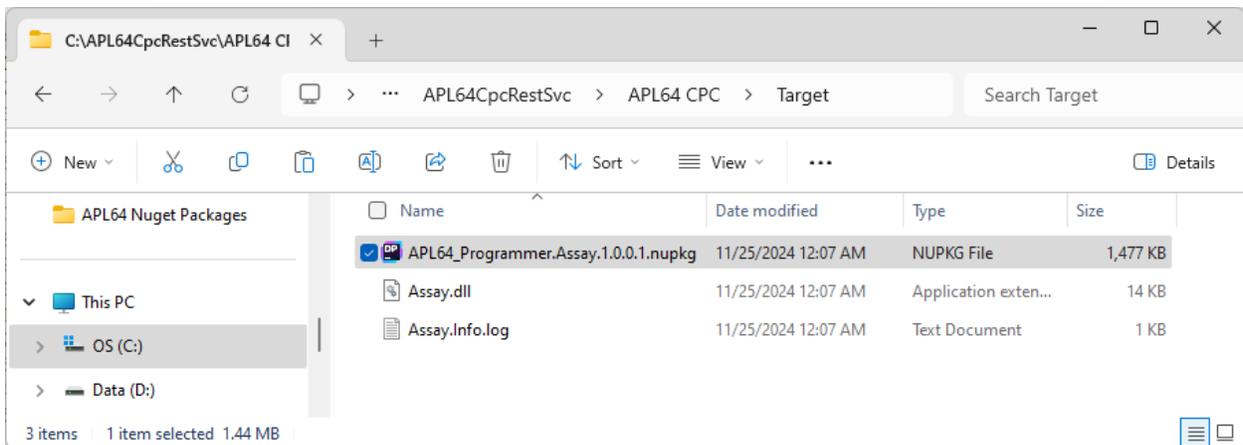


Publish the CPC Nuget Package Locally

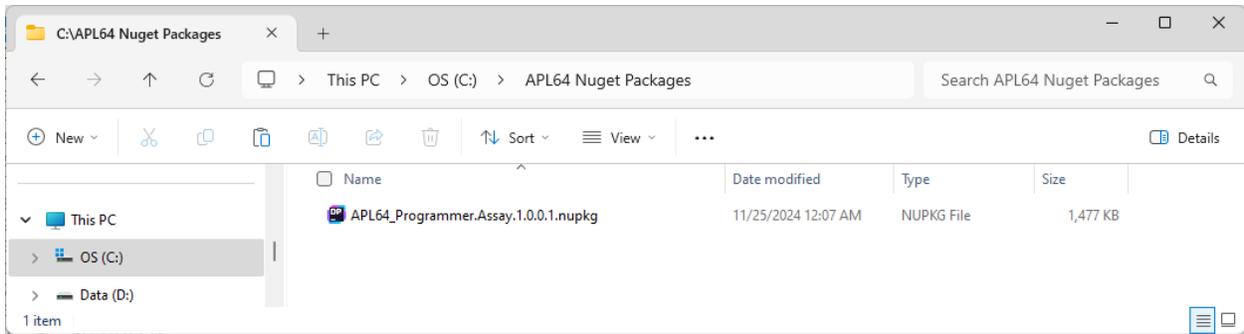
Create the folder: C:\APL64 Nuget Packages\ for testing and debugging purposes. In a production environment the CPC Nuget package would be published in a secure Nuget package repository on the Web.

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.

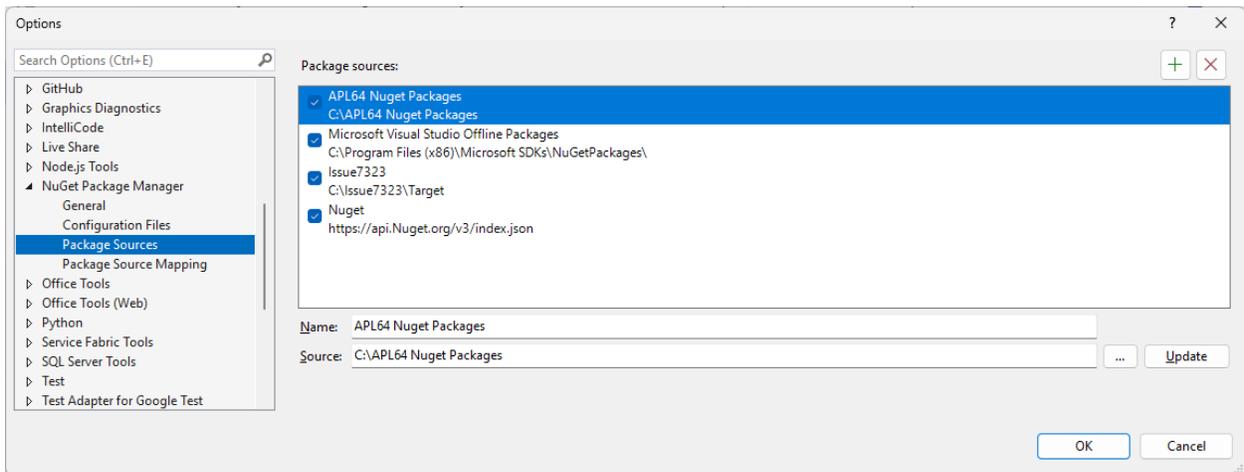
From:



To:

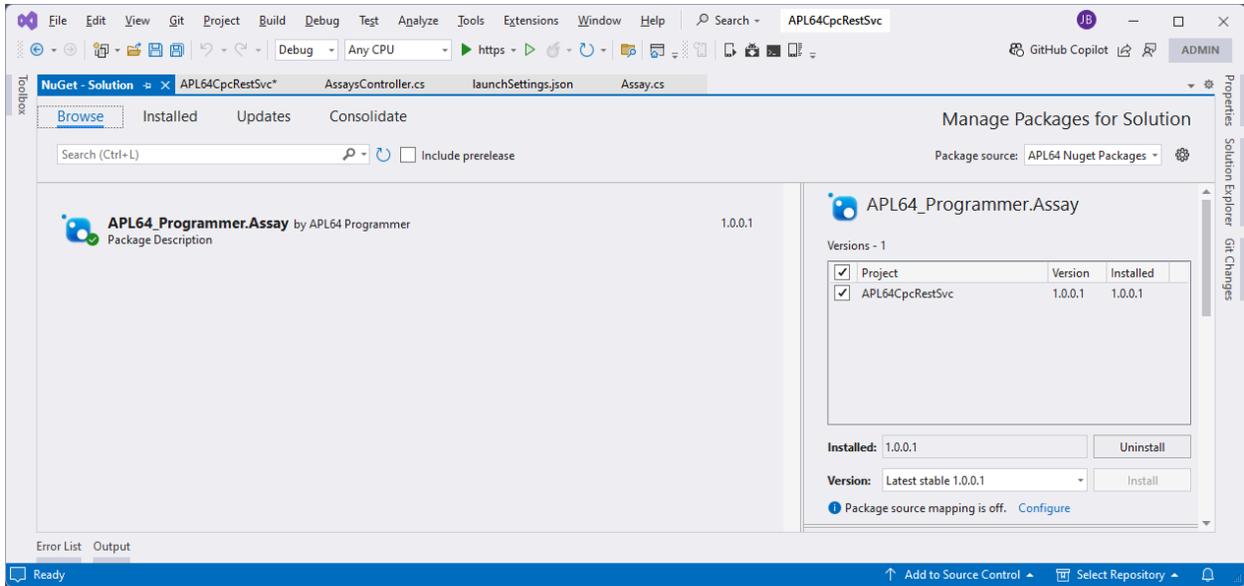


The local Nuget package publish folder must be registered with Visual Studio as a Nuget package source:



Install the CPC 'Assay' Nuget Package into the Project

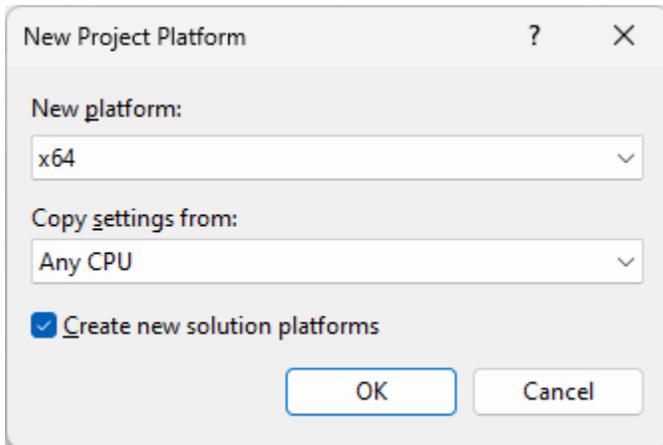
Use the Visual Studio | Tools | Nuget Package Manager | Manage Nuget Packages for the solution... dialog:

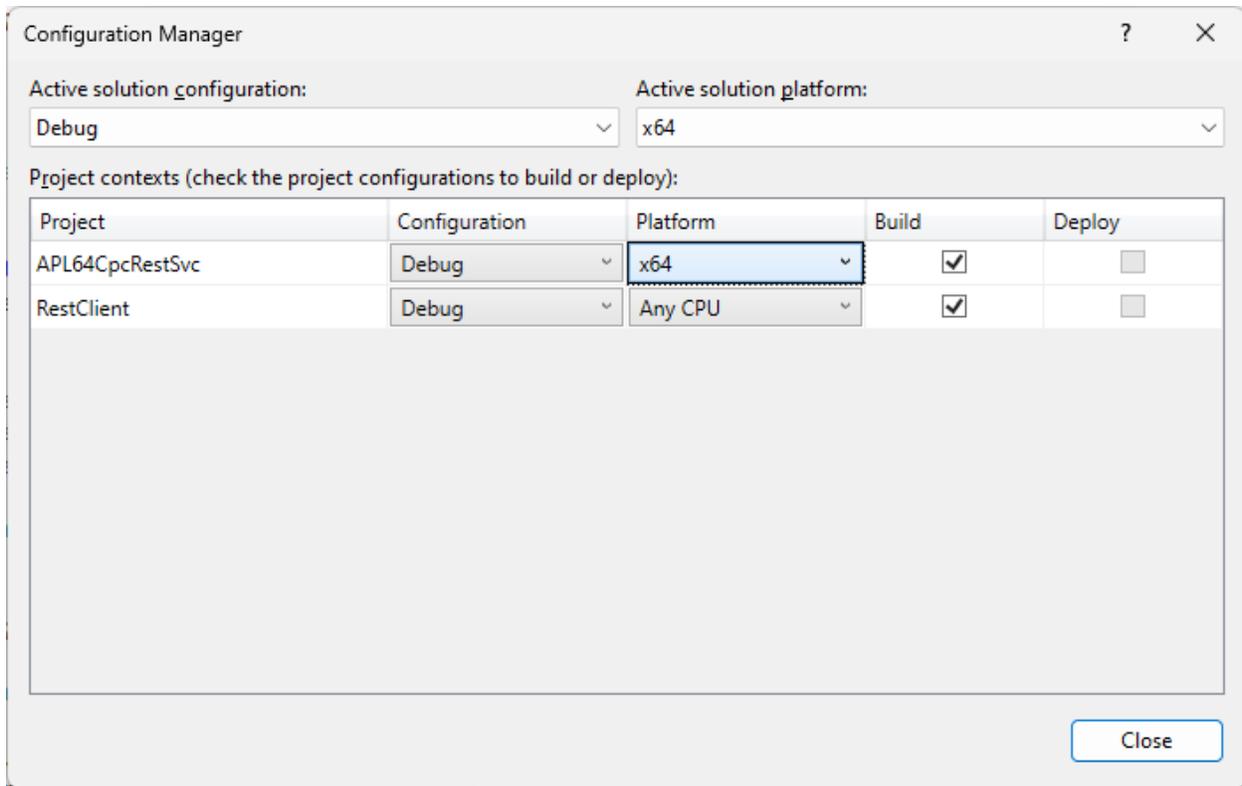


Set x64 Build Configuration for Rest Service Project

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:





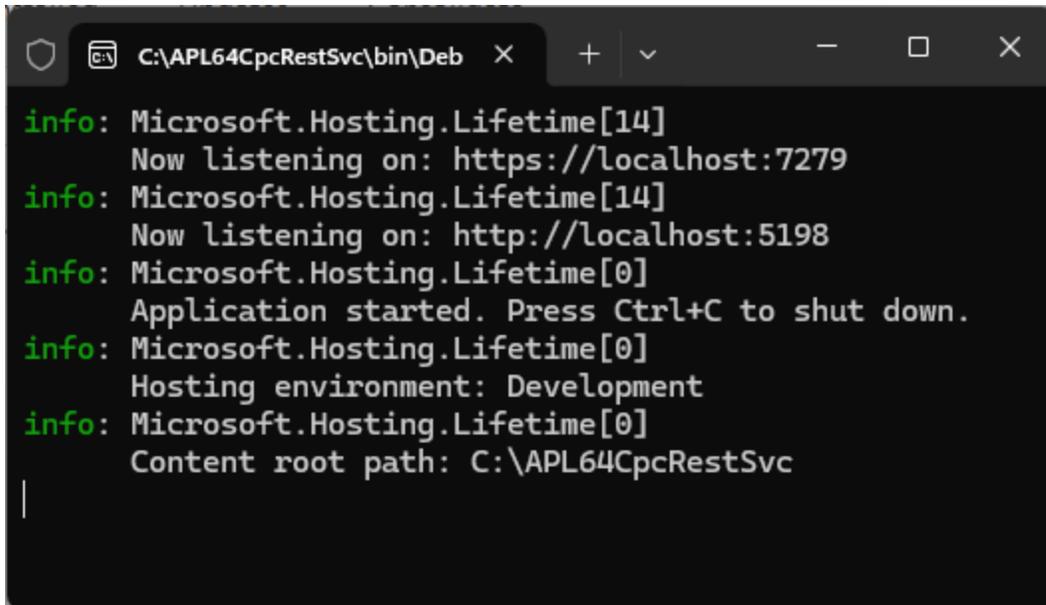
Start the REST Service

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()



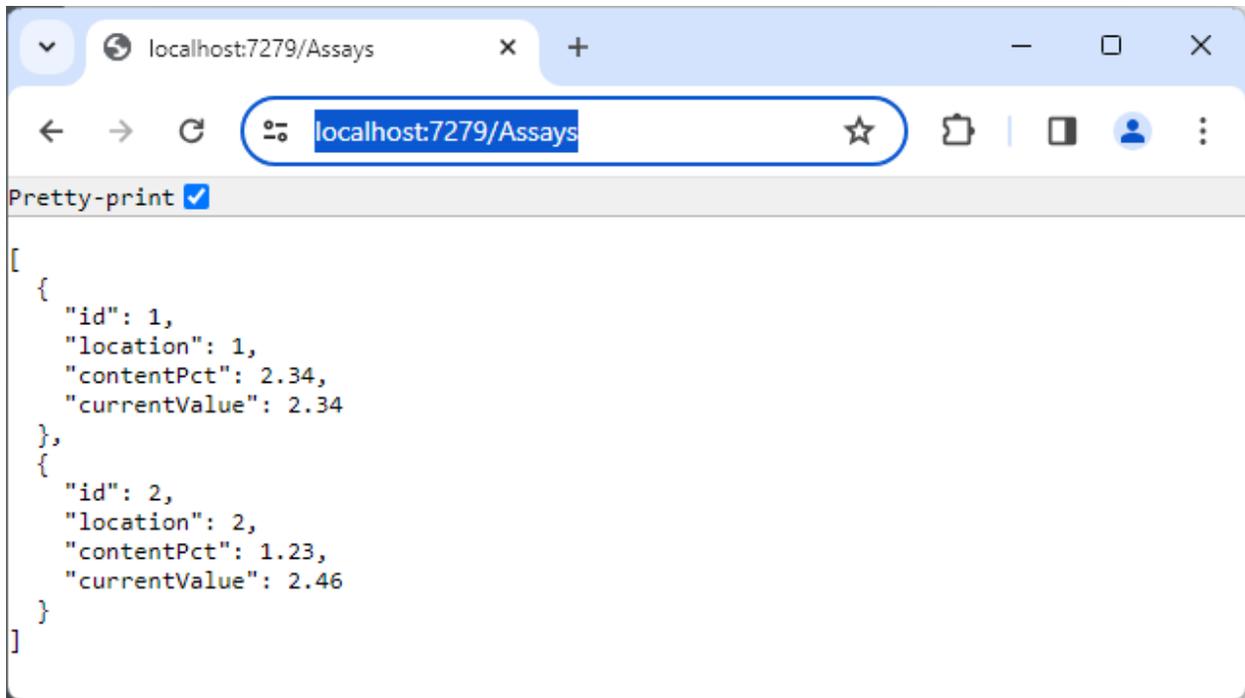
```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7279
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5198
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\APL64CpcRestSvc
```

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.

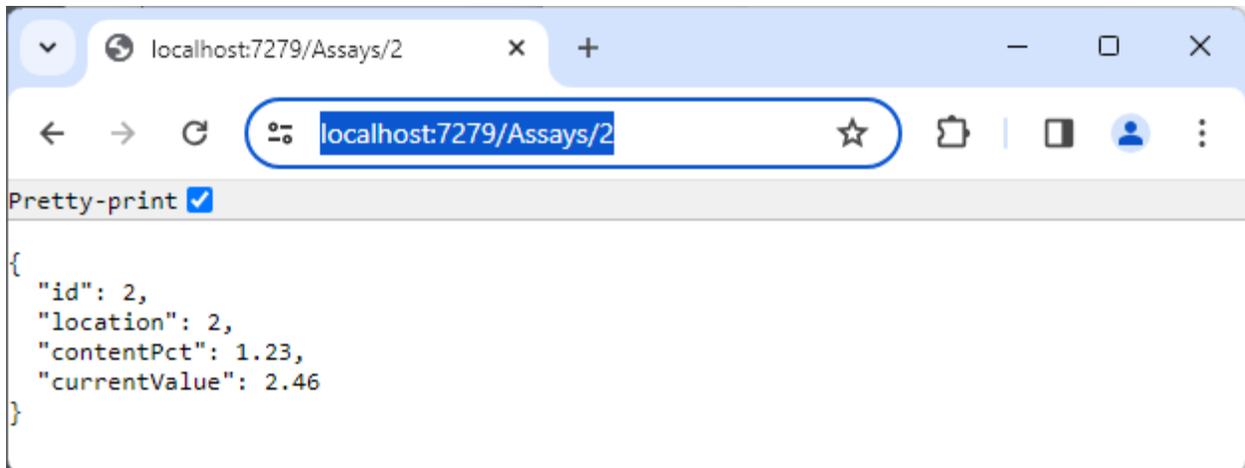
If the default browser indicates that the 'localhost:7279/Assays' site is insecure, select the 'http' option instead of the 'https' option in Visual Studio to start the server and see the 'localhost:5198/Assay' web page in the browser.

Access the REST Service from a Browser

localhost:7279/Assays



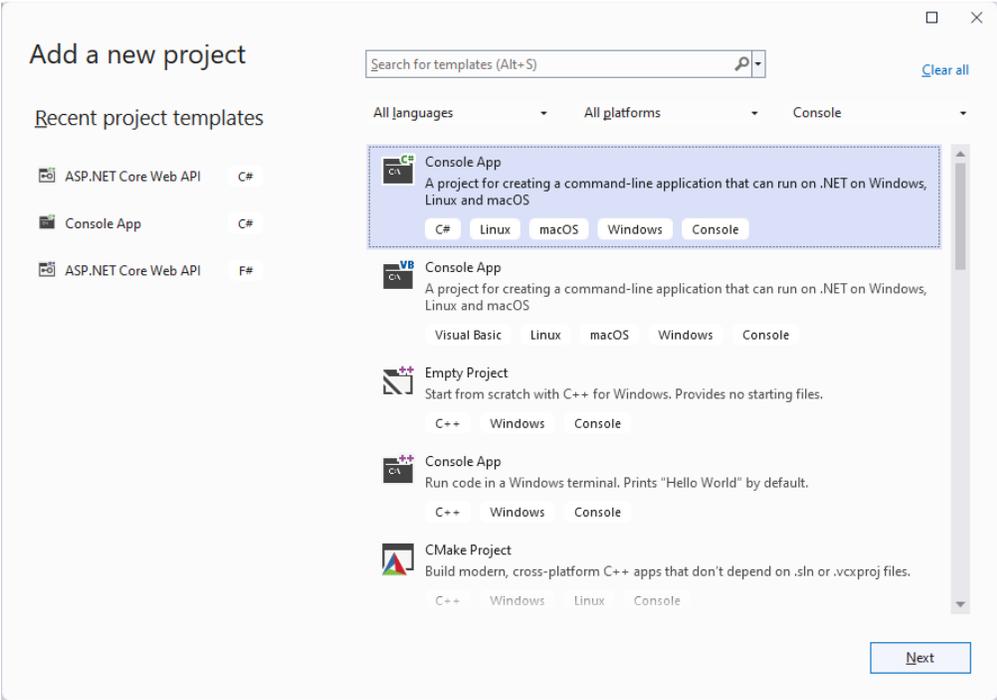
`localhost:7279/Assays/2`



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

Configure your new project

Console App C# Linux macOS Windows Console

Project name
RestClient

Location
C:\APL64CpcRestSvc\

Project will be created in "C:\APL64CpcRestSvc\RestClient\"

Back Next

Provide the Additional Information

Additional information

Console App C# Linux macOS Windows Console

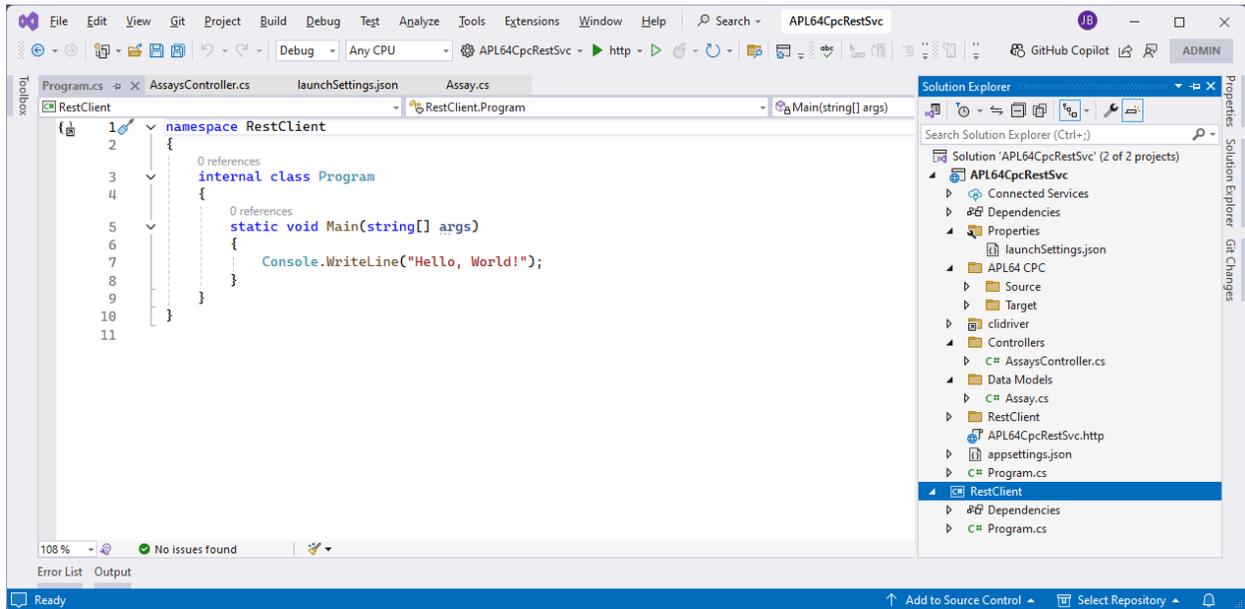
Framework
.NET 8.0 (Long Term Support)

Do not use top-level statements

Enable native AOT publish

Back Create

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

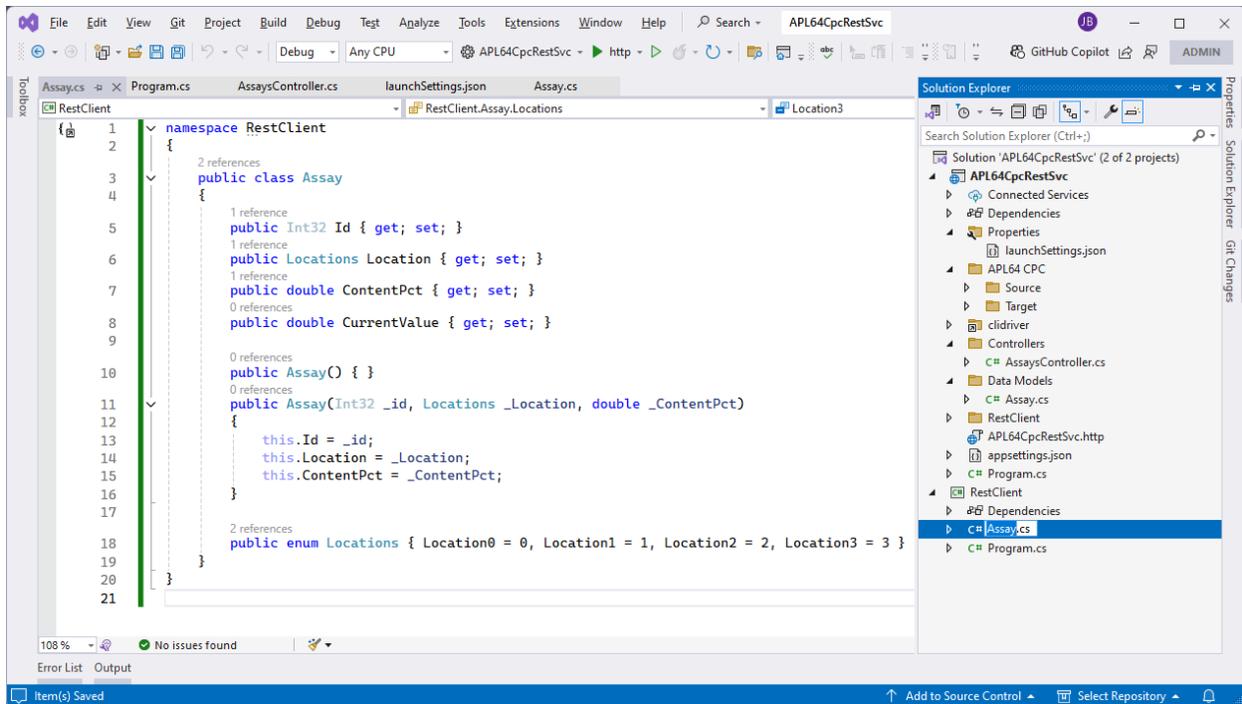


Add an Assay.cs class to the RestClient project with this code:

```
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)

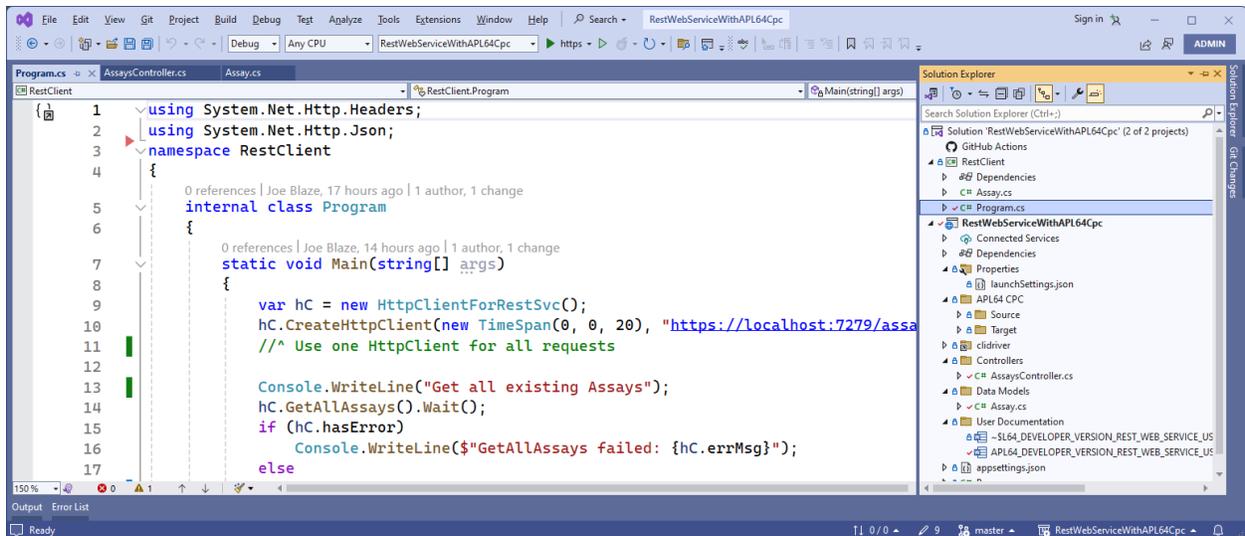
```

```

    {
        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;
    }
}
}
}

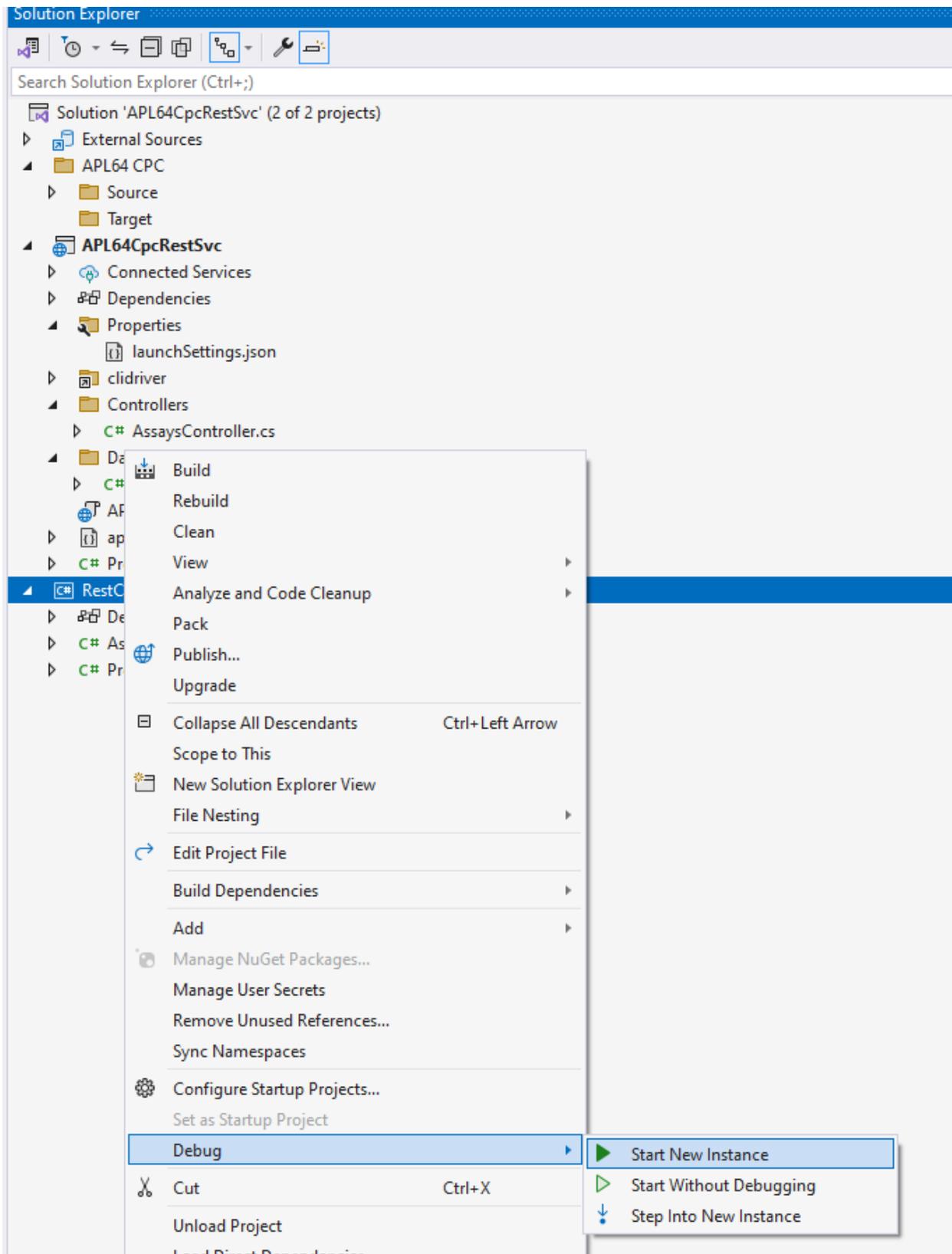
```



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:



```
C:\Users\joebl\source\repos\APLNowLLC\RestWebServiceWithAPL64Cpc\RestWebServiceWithAPL64Cpc\RestClient\bin\Deb...
Get all existing Assays
GetAllAssays result:
[{"id":1,"location":1,"contentPct":2.34,"currentValue":2.34},{"id":2,"location":2,"contentPct":1.23,"
currentValue":2.46}]

Get Assays for Id=2
GetAssay(2) result:
{"id":2,"location":2,"contentPct":1.23,"currentValue":2.46}

Create new Assay for Id=3, Location=Location3, ContentPct=4.33
CreateAssay(3, Assay.Locations.Location3, 4.33) result:
{"id":3,"location":3,"contentPct":4.33,"currentValue":12.99}

Update Assay for Id=3 , Location=Location1, ContentPct=2.22
Get all existing Assays
GetAllAssays result:
[{"id":1,"location":1,"contentPct":2.34,"currentValue":2.34},{"id":2,"location":2,"contentPct":1.23,"
currentValue":2.46},{"id":3,"location":1,"contentPct":2.22,"currentValue":2.22}]

Delete Assay for Id=3
Get all existing Assays
GetAllAssays result:
[{"id":1,"location":1,"contentPct":2.34,"currentValue":2.34},{"id":2,"location":2,"contentPct":1.23,"
currentValue":2.46}]
```

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:

```
*new 1 - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1
1 [{"id": 1,
2   {
3     "id": 1,
4     "location": 1,
5     "contentPct": 2.34,
6     "currentValue": 2.34
7   }},
8   {
9     "id": 2,
10    "location": 2,
11    "contentPct": 1.23,
12    "currentValue": 2.46
13  }
14 ]
JSON file length: 175 lines: 14 Ln: 1 Col: 1 Pos: 1 Windows (CR LF) UTF-8 INS
```

Get Assays for Id=2

```
*new 1 - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? +
new 1
1
2     "id": 2,
3     "location": 2,
4     "contentPct": 1.23,
5     "currentValue": 2.46
6
length : 77 lines Ln: 6 Col: 2 Pos: 78 Windows (CR LF) UTF-8 INS
```

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

```
*new 1 - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ? +
new 1
1
2     "id": 3,
3     "location": 3,
4     "contentPct": 4.33,
5     "currentValue": 12.99
6
length : 78 lines Ln: 6 Col: 2 Pos: 79 Windows (CR LF) UTF-8 INS
```

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

The screenshot shows a Notepad++ window titled '*new 1 - Notepad++ [Administrator]'. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, and Window. The toolbar contains various icons for file operations and editing. The main text area contains a single JSON object on line 2:

```
1  
2     "id": 3,  
3     "location": 1,  
4     "contentPct": 2.22,  
5     "currentValue": 2.22  
6
```

The status bar at the bottom indicates: length : 77 lines Ln : 6 Col : 2 Pos : 78 Windows (CR LF) UTF-8 INS

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

The screenshot shows a Notepad++ window titled '*new 1 - Notepad++ [Administrator]'. The menu bar and toolbar are the same as in the previous screenshot. The main text area contains a JSON array with two objects:

```
1  
2     {  
3         "id": 1,  
4         "location": 1,  
5         "contentPct": 2.34,  
6         "currentValue": 2.34  
7     },  
8     {  
9         "id": 2,  
10        "location": 2,  
11        "contentPct": 1.23,  
12        "currentValue": 2.46  
13    }  
14
```

The status bar at the bottom indicates: length : 175 line: Ln : 14 Col : 2 Pos : 176 Windows (CR LF) UTF-8 INS

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.