REST API Server Code

Build REST API endpoints in FrameworX.

Looking for FrameworX's own REST API?

This page is a scripting example for **building a custom HTTP endpoint inside a solution** with a script. FrameworX already exposes a built-in REST API for live runtime data — to read/write tags, alarms, and history over HTTP, see the Runtime REST API Reference and the FrameworX REST APIs overview.

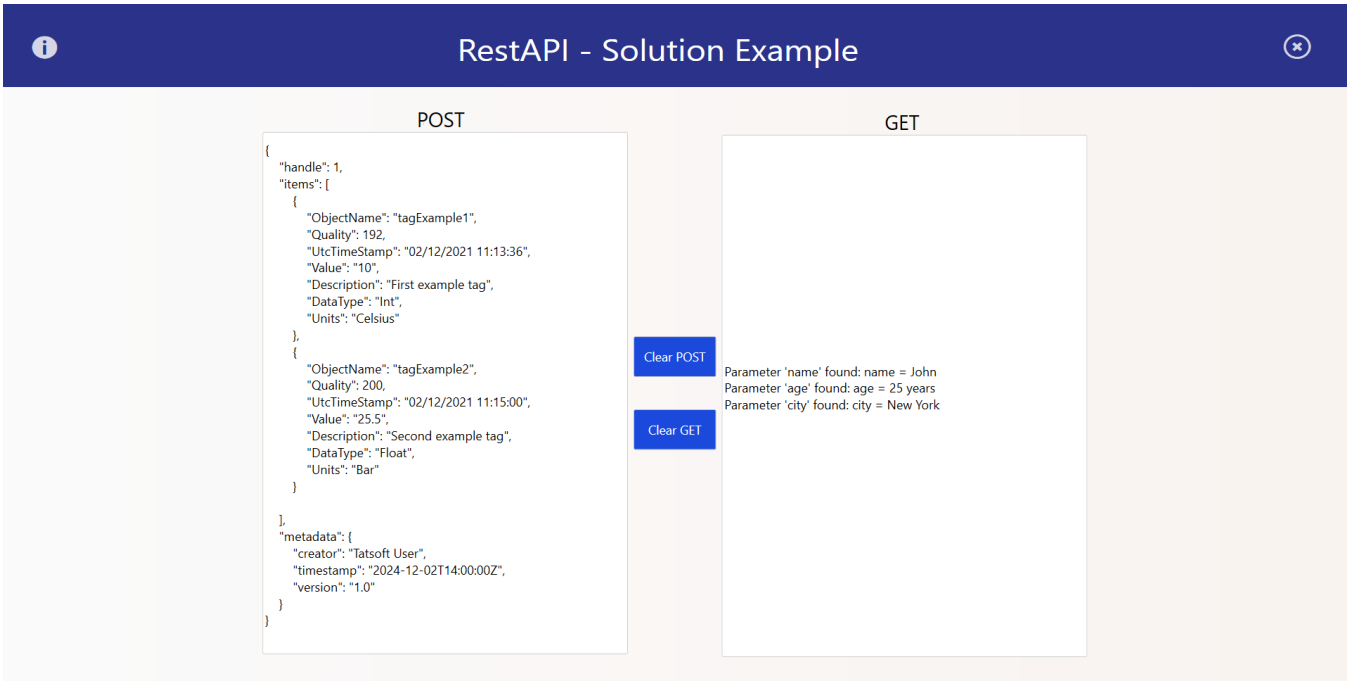
Download the Solution Example here: Rest API - Solution Example

- **Solution Name:** REST API
- **Software version:** v10
- **Keywords:** RestAPI, Service Listener

Summary
Technical Information

Summary

This solution example demonstrates how to use a custom HTTP server to process GET and POST requests. The server listens for requests and provides a method to handle them (CustomProcessRequest). It allows processing of query parameters in GET requests and request bodies in POST requests.



Technical Information

This solution implements a HTTP server using the T.Library.tRPC to handle GET and POST requests. The server is initialized with a callback function, CustomProcessRequest, which provides a dynamic way to process incoming HTTP requests.

For GET requests, the server inspects the query string parameters, specifically looking for predefined keys such as name, age, and city. If any of these parameters are present, their values are logged and appended to a text tag (@Tag.Get) for tracking or further processing.

For POST requests, the server reads and logs the body of the request, storing its content in the @Tag.Post text tag.

```
private T.Library.tRPC.Server server = null;

public void Initialize()
{
        server = new T.Library.tRPC.Server() { CustomProcessRequest = this.CustomProcessRequest };
        server.StartListen(20001);
}

public void Uninitialize()
{
        server.Dispose();
        server = null;
}

private bool CustomProcessRequest(T.Library.tRPC.IHttpRequest request, T.Library.tRPC.IHttpResponse response)
{

        @Info.Trace("Request1: " + request.Uri);

        if (request.Method == "GET" && request.UriPath.Contains("/solution"))
    {
        var queryParameters = request.QueryString;
        @Info.Trace("query: " + queryParameters.ToString());

        var expectedParameters = new[] { "name", "age", "city" };

        foreach (var parameter in expectedParameters)
        {
            if (queryParameters.ToString().Contains(parameter))
            {
                var value = queryParameters[parameter];
                @Info.Trace($"Parameter '{parameter}' found: {value}");
                @Tag.Get += $"Parameter '{parameter}' found: {value}";
            }

        }

    }
        else if (request.Method == "POST" && request.UriPath.Contains("/solution"))
    {
        using (var reader = new StreamReader(request.Body))
        {
            var requestBody = reader.ReadToEnd();

            if (!string.IsNullOrWhiteSpace(requestBody))
            {
                @Info.Trace("POST Request Body: " + requestBody);
                                @Tag.Post = requestBody;
            }
                }
        }

        return true;
}
```

Scripts Tasks ServerStartup: The server is initialized using calling the Initialize method.

```
@Script.Class.RestApi.Initialize();
```

Scripts Tasks ServerShutdown: The server is properly shut down using the Uninitialize method.

```
@Script.Class.RestApi.Uninitialize();
```

For this solution example, **Postman software** was used to test and validate the HTTP server functionality. This software is a powerful and user-friendly API testing application that allows developers to easily send HTTP requests and inspect responses.

You can download the **Postman** by accessing the official website at the following link: https://www.postman.com/

In this section...
