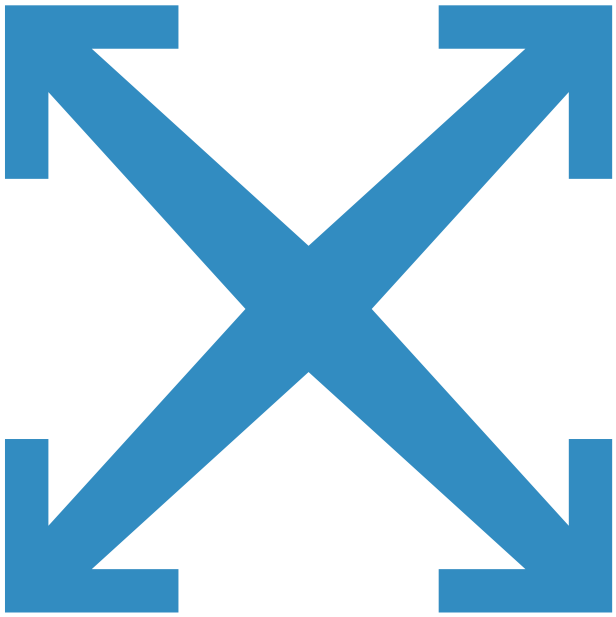


Crosser Node SDK Documentation

v5.0.0

Table of Contents

Introduction

This documentation will guide you in building custom modules for the Crosser platform.

Lets start with covering the basic nomenclature & concept.

What´s a Module?

A Module is the smallest part of a Flow. Basically we have 3 types of modules

- Input Modules
- Function Modules
- Output Modules

Input Modules will either create data within it-self, collect data from a data source or receive data from the outside world like HTTP, MQTT, WebSocket etc. Regardless of producing, collecting or receiving data, the Input Modules purpose is to serve the flow with data to process.

These modules can pass data to `Function Modules` and `Output Modules`, but never to another `Input Module`.

Function Modules can be very complex. They all have in common that they will receive data and at some point pass data to the next module(s) in the flow. Receiving and sending is not necessarily a 1 to 1 relationship.

These modules can pass data to other `Function Modules` and `Output Modules`, but never to an `Input Module`.

Output Modules will receive data from other module types and send to external systems like applications/services/databases/etc.

All module types, including output modules, should always send a result out so it is possible to continue building on the result or that we can retry or log any failures. The output should as default contain the data from the incoming messages.

Module UI

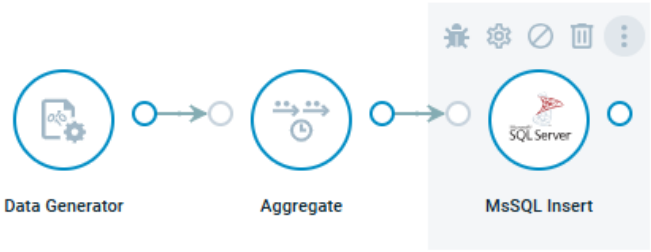
When building modules with this SDK a UI is built automatically. It is dependent on the module type and the settings specified, see Module Settings

Modules will be covered in depth later

What´s a Flow?

A flow is just a container for grouping modules together.

Let´s look at a simple flow (created in our FlowStudio) with one module of each type, just to get a feeling what we are talking about.

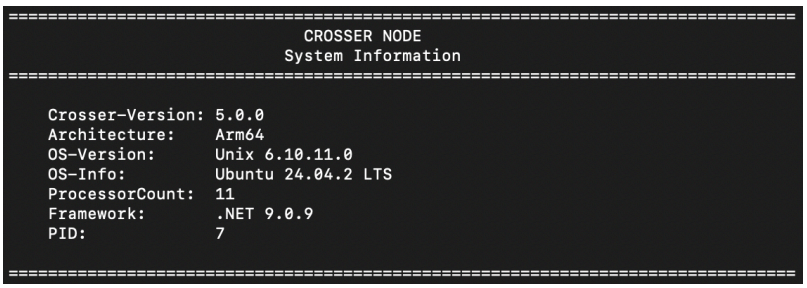


The Node

The Crosser Node is the engine that runs your flows. It is responsible for executing the logic defined by your modules, moving data through each step of the flow.

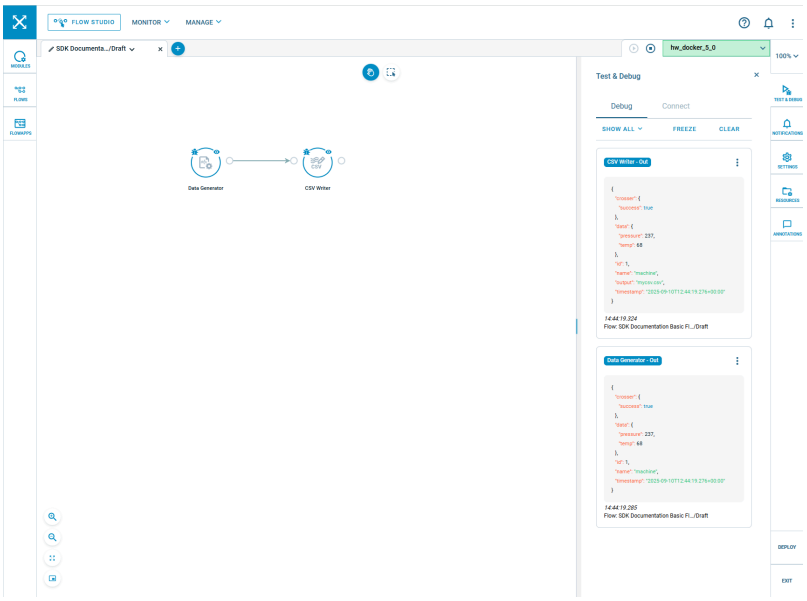
When you deploy a flow, the Node loads the flow definition, instantiates each module, and orchestrates the data through the flow. The Node ensures that each module receives the right data, applies its logic, and passes results to the next module or external destination.

In essence, the Node is the runtime environment for your modules connected in flow, handling message routing, managing resources, and providing the reliability and scalability needed for real-time analytics and integration.



The FlowStudio

The Flow Studio is a drag-and-drop visual design tool where you select and connect pre-built building blocks called modules to develop your flows and the Flow Studio is a central component in Crosser Cloud.


FLOW STUDIO
MONITOR
MANAGE
SDK Documenta.../Draft
hw_docker_5_0
100%
MODULES
FLOWS
FLOWAPPS
Data Generator
CSV Writer
Test & Debug
Debug
Connect
SHOW ALL
FREEZE
CLEAR
CSV Writer - Out
{
"processor": {
"success": true
},
"data": {
"pressure": 237,
"temp": 68
},
"id": 5,
"name": "machine",
"output": "myenv.csv",
"timestamp": "2025-09-10T12:44:19.276+00:00"
}
14:44:19.324
Flow: SDK Documentation Basic Fl.../Draft
Data Generator - Out
{
"processor": {
"success": true
},
"data": {
"pressure": 237,
"temp": 68
},
"id": 5,
"name": "machine",
"timestamp": "2025-09-10T12:44:19.276+00:00"
}
14:44:19.285
Flow: SDK Documentation Basic Fl.../Draft
TEST & DEBUG
NOTIFICATIONS
SETTINGS
RESOURCES
ANNOTATIONS
DEPLOY
EXIT

Getting Started

> TIP
>
> You will need to have .NET 9.0 SDK or later installed on your machine.
>
> Tools used are the dotnet cli and VSCODE, but you can use tools you are familiar with.

This guide will help you through the basics of working with the Crosser Node SDK to create your first module.

- Creating a new project
- Installing the SDK
- Creating a custom module

We recommend you to have some basic understanding around:

- C#
- Nuget packages

>> Create a Project

Create Project

To create a project for your Crosser Flow Module you can either create a new project from scratch or use the the official Crosser.Templates The template will create a project with the correct structure and items for you. See the readme for instructions on how to use the template.

Create Project from Scratch

Open a terminal and run the following command to create a new class library project(Replace MyOrg with your organization name).

> WARNING
>
> - The name of your project and module needs to be unique within your organization.
> - The project name will become the package name (by default), so make sure that the name is available on nuget It is highly recommended to include your organization name in the package name to avoid name conflicts.

```
dotnet new classlib -o RandomNumberModule -n MyOrg.FlowModule.RandomNumberModule
```

You will now have a new folder `RandomNumberModule` containing your a project `MyOrg.FlowModule.RandomNumberModule`.

>> Install SDK

Install SDK

If you created your project using the Crosser.Templates NuGet package, the SDK is already installed and you can skip this step and move on to >> Create a Module.

Navigate into the folder where you created your project

```
cd RandomNumberModule
```

and then install the SDK from nuget.org. Make sure you have nuget.org configured as a package source.

```
dotnet add package Crosser.EdgeNode.Flows
```

If you look in the `MyOrg.FlowModule.RandomNumberModule.csproj` file you should see something like:

```
<ItemGroup>
  <PackageReference Include="Crosser.EdgeNode.Flows" Version="5.0.0" />
</ItemGroup>
```

To make sure that everything is ok you can build the project. You should not get any errors.

```
dotnet build
```

This is the basics of setting up your project with the Crosser SDK.

>> Create a Module

Modules

> TIP
>
> A deeper explanation can be found in the Guidelines.

Module Design Principles

A well-designed module should:

- Have a clear purpose
  Each module should solve a single, well-defined task. This makes it easier to build flows where each module has a single responsibility.
- Hide complexity
  Expose only simple, intuitive settings to users, even when the underlying implementation is complex.
- Define settings clearly
  Specify what properties can be configured and explain the purpose of each property.
- Fit a specific role in the flow
  Design modules to be used to produce or receive data (Input), process data (Function), or send data (Output) in a flow, not all roles at once.

Next Steps

>> Creating a Module

Creating a Module

We will show how to implement a `Random Number` module in 4 steps

> TIP
>
> Namespaces used on this page
>
> - using Crosser.EdgeNode.Flows.Models.Abstractions.Models;
> - using Crosser.EdgeNode.Flows;

To create a custom module we have to inherit the class `FlowModule<T>`.

The base class force you to implement a few things:

- The `UserFriendlyName` property
- A call to the base class constructor saying what `FlowModuleType` the module is
- The implementation of `MessageReceived`

> TIP
>
> You can also implement other methods to handle lifecycle events. These are covered in Module Lifetime

```
public class RandomNumberModule : FlowModule<RandomNumberModuleSettings>
{
    public override string UserFriendlyName => "Random Number";

    public RandomNumberModule() : base(FlowModuleType.Function) {}

    protected override Task MessageReceived(IFlowMessage message)
    {
        // We will cover this in step: Receive Messages
        throw new NotImplementedException();
    }
}
```

Since we did not create the `RandomNumberModuleSettings` class yet this will not compile.

Next step is defining the settings!

>> Settings

Create Settings

Settings are optional but recommended for almost all modules. Settings define configurable properties that users can adjust in Flow Studio.

> TIP
>
> Required namespaces:
>
> - `using Crosser.EdgeNode.Common.Abstractions.Utilities.Validation`
> - `using Crosser.EdgeNode.Flows.Abstractions`
> - `using System.ComponentModel`
> - `using System.ComponentModel.DataAnnotations`

To create a settings class we must inherit the class `FlowModuleSettings`: For simplicity we will add the class to same file as the module, but in a real-world scenario it is recommended to create a separate file for better organization.

```
public class RandomNumberModuleSettings : FlowModuleSettings
{
    [Display(Name = "Minimum Value", Description = "The lowest value in the
random range")]
    [DefaultValue(1)]
    [Range(1, 10000)]
    public int TargetMin { get; set; } = 1;

    [Display(Name = "Maximum Value", Description = "The highest value in the
random range")]
    [DefaultValue(100)]
    [Range(1, 10000)]
    public int TargetMax { get; set; } = 100;

    [Display(Name = "Target Property", Description = "The property to store the
random value in")]
    [DefaultValue("data")]
    [MinLength(1)]
    [MaxLength(64)]
    public string TargetName { get; set; } = "data";
}
```

> TIP
>
> Note that by using the DefaultValue, Description and Display attributes we will get automatic default values, names and descriptions in the FlowStudio UI when the users are using your module.

Add Validation

Now we will add validation for the properties so that users can't save modules if settings are invalid. Below we override the Validate method and use the built-in extensions for validation. We also show how to add errors based on custom conditions.

```
public class RandomNumberModuleSettings : FlowModuleSettings
{
    // Properties hidden for readability

    public override void Validate(SettingsValidator validator)
    {
        // Built-in validators
        validator.Validate(nameof(this.TargetMin), this.TargetMin)
            .MinValue(0)
            .MaxValue(10000);

        validator.Validate(nameof(this.TargetMax), this.TargetMax)
            .MinValue(1)
            .MaxValue(10000);

        validator.Validate(nameof(this.TargetName), this.TargetName)
            .NotNull()
            .MinLength(1)
            .MaxLength(64);

        // Custom validation logic
        if (this.TargetMin >= this.TargetMax)
        {
            validator.AddError(nameof(this.TargetMin),
                "Minimum value must be less than maximum value");
        }
    }
}
```

TIP

See Module Settings Guidelines for more attributes and validation options.

>> Module Credentials

Module Credentials

Credentials allow you to securely store sensitive information like passwords and API keys without hardcoding them in your module. They are part of the settings but configured in Crosser Control Center.

If you don't need credentials, skip to the next section: >> Receive Messages

> TIP
>
> Required namespaces:
>
> - `Crosser.EdgeNode.Flows.Abstractions`
> - `System.ComponentModel`
> - `System.ComponentModel.DataAnnotations`
> - `NJsonSchema.Annotations`

Basics

Credentials are created and managed in Crosser Control Center. Once created, you can reference them in your module settings.

Built-in Credential Types

Available credential types are defined in Crosser.EdgeNode.Flows.Abstractions.Credential.Types:

- `UsernameAndPassword` - Basic username/password authentication
- `ApiKey` - API key authentication
- `ConnectionString` - Connection string for databases
- `Certificate` - Client certificates for mutual TLS
- `Data` - Custom credential data (JSON or binary)

Adding Credential Property

Add a credential property to your settings class with the `JsonSchemaExtensionData` attribute:

```
public class MyModuleSettings : FlowModuleSettings
{
    [Display(Name = "Database Credentials")]
    [JsonSchemaExtensionData(Credential.ATTRIBUTE,
Credential.Types.ConnectionString)]
    public Guid? CredentialId { get; set; }
}
```

Accessing Credentials

The best place to retrieve the credentials is in your module's `Initialize` method using the `GetCredentialContentAsync` extension method:

```
public override async Task<IError?> Initialize()
{
    if (!this.Settings.CredentialId.HasValue)
    {
        return new Error("Credential not configured");
    }

    var credential = await
this.GetCredentialContentAsync<CredentialWithConnectionString>(
        this.Settings.CredentialId.Value);

    if (credential.IsError || credential.Value is null)
    {
        return credential.Error ?? new Error("Failed to get credential");
    }

    this.connection = new MySqlConnection(credential.Value.ConnectionString);
    await this.connection.OpenAsync();

    return await base.Initialize();
}
```

The above example retrieves a connection string credential and uses it to open a database connection. Below are some common credential types you might use.

Common Credential Types

Username & Password

```
[Display(Name = "Database Credentials")]
[JsonSchemaExtensionData(Credential.ATTRIBUTE,
Credential.Types.UsernameAndPassword)]
public Guid? DatabaseCredential { get; set; }
```

Accessing the credential:

```
var credential = await
this.GetCredentialContentAsync<CredentialWithUsernamePassword>(
    this.Settings.DatabaseCredential.Value);
```

```
var connectionString = $"Server={server};User Id=
{credential.Value.Username};Password={credential.Value.Password}";
```

API Key

```
[Display(Name = "API Key")]
[JsonSchemaExtensionData(Credential.ATTRIBUTE, Credential.Types.ApiKey)]
public Guid? ApiKeyCredential { get; set; }
```

Accessing the credential:

```
var apiKey = await this.GetCredentialContentAsync<CredentialWithApiKey>(
    this.Settings.ApiKeyCredential.Value);

this.httpClient.DefaultRequestHeaders.Add("X-API-Key", apiKey.Value.ApiKey);
```

Certificate

```
[Display(Name = "Client Certificate")]
[JsonSchemaExtensionData(Credential.ATTRIBUTE, Credential.Types.Certificate)]
public Guid? ClientCertificate { get; set; }
```

Accessing the credential:

```
var cert = await this.GetCredentialContentAsync<CredentialWithCertificate>(
    this.Settings.ClientCertificate.Value);

var handler = new HttpClientHandler();
handler.ClientCertificates.Add(cert.Value.X509Certificate2);
this.httpClient = new HttpClient(handler);
```

Custom Credentials

For unsupported credential types, use `Credential.Types.Data` with custom JSON or binary data:

```
[Display(Name = "Custom Credentials")]
[JsonSchemaExtensionData(Credential.ATTRIBUTE, Credential.Types.Data)]
public Guid CustomCredential { get; set; }  // Required (not nullable)
```

Accessing custom data:

```
var customData = await this.GetCredentialContentAsync<CredentialWithData>(
    this.Settings.CustomCredential);

// Parse JSON data
var authData = JsonSerializer.Deserialize<MyCustomAuthData>(
    Encoding.UTF8.GetString(customData.Value.Data));
```

Multiple Credential Types

Allow users to choose between different credential types by specifying them as comma-separated values:

```
public class FlexibleAuthModuleSettings : FlowModuleSettings
{
    [Display(Name = "Authentication", Description = "Supports Username/Password or
API Key")]
    [JsonSchemaExtensionData(Credential.ATTRIBUTE, "UsernamePassword,ApiKey")]
    public Guid? AuthCredential { get; set; }
}
```

Pattern 1: Type Detection with Fallback

Handle multiple types at runtime by trying each type sequentially:

```
public override async Task<IError?> Initialize()
{
    if (!this.Settings.AuthCredential.HasValue)
    {
        return new Error("Authentication credential not configured");
    }

    // Try username/password first
    var userPass = await
this.GetCredentialContentAsync<CredentialWithUsernamePassword>(
        this.Settings.AuthCredential.Value);

    if (!userPass.IsError && userPass.Value is not null)
    {
        Information("Using username/password authentication");
        await this.AuthenticateWithPassword(
            userPass.Value.Username,
            userPass.Value.Password);
        return await base.Initialize();
    }
```

```
    // Fall back to API key
    var apiKey = await this.GetCredentialContentAsync<CredentialWithApiKey>(
        this.Settings.AuthCredential.Value);

    if (!apiKey.IsError && apiKey.Value is not null)
    {
        Information("Using API key authentication");
        await this.AuthenticateWithApiKey(apiKey.Value.ApiKey);
        return await base.Initialize();
    }

    return new Error("No valid authentication credentials configured. " +
                     "Please configure either Username/Password or API Key.");
}

private async Task AuthenticateWithPassword(string username, string password)
{
    this.httpClient.DefaultRequestHeaders.Authorization =
        new System.Net.Http.Headers.AuthenticationHeaderValue(
            "Basic",
            Convert.ToBase64String(
                System.Text.Encoding.UTF8.GetBytes($"{username}:{password}")));

    await this.ValidateConnection();
}

private async Task AuthenticateWithApiKey(string apiKey)
{
    this.httpClient.DefaultRequestHeaders.Add("X-API-Key", apiKey);
    await this.ValidateConnection();
}

private async Task ValidateConnection()
{
    var response = await this.httpClient.GetAsync("/api/health");
    if (!response.IsSuccessStatusCode)
    {
        throw new FlowModuleException("Failed to validate authentication");
    }
}
```

Pattern 2: Multiple Credentials for Different Services

Use multiple credential properties when your module integrates with different services:

```
public class IntegrationModuleSettings : FlowModuleSettings
{
    [Display(Name = "Primary Service", Description = "Credentials for primary API")]
    [JsonSchemaExtensionData(Credential.ATTRIBUTE, Credential.Types.ApiKey)]
    public Guid? PrimaryApiCredential { get; set; }

    [Display(Name = "Database", Description = "Database connection")]
    [JsonSchemaExtensionData(Credential.ATTRIBUTE,
Credential.Types.ConnectionString)]
    public Guid? DatabaseCredential { get; set; }

    [Display(Name = "Secondary Service", Description = "Optional secondary API")]
    [JsonSchemaExtensionData(Credential.ATTRIBUTE,
Credential.Types.UsernameAndPassword)]
    public Guid? SecondaryApiCredential { get; set; }
}
```

Accessing multiple credentials:

```
public override async Task<IError?> Initialize()
{
    // Primary API (required)
    if (!this.Settings.PrimaryApiCredential.HasValue)
    {
        return new Error("Primary API credential required");
    }

    var primaryKey = await this.GetCredentialContentAsync<CredentialWithApiKey>(
        this.Settings.PrimaryApiCredential.Value);

    if (primaryKey.IsError || primaryKey.Value is null)
    {
        return primaryKey.Error ?? new Error("Failed to get primary
API credential");
    }

    this.primaryClient = new HttpClient();
    this.primaryClient.DefaultRequestHeaders.Add("X-API-
Key", primaryKey.Value.ApiKey);

    // Database (required)
    if (!this.Settings.DatabaseCredential.HasValue)
    {
        return new Error("Database credential required");
    }
```

```
    var dbCredential = await
this.GetCredentialContentAsync<CredentialWithConnectionString>(
        this.Settings.DatabaseCredential.Value);

    if (dbCredential.IsError || dbCredential.Value is null)
    {
        return dbCredential.Error ?? new Error("Failed to get database credential");
    }

    this.dbConnection = new SqlConnection(dbCredential.Value.ConnectionString);
    await this.dbConnection.OpenAsync();

    // Secondary API (optional)
    if (this.Settings.SecondaryApiCredential.HasValue)
    {
        var secondaryCredential = await
this.GetCredentialContentAsync<CredentialWithUsernamePassword>(
            this.Settings.SecondaryApiCredential.Value);

        if (!secondaryCredential.IsError && secondaryCredential.Value is not null)
        {
            this.secondaryClient = new HttpClient();
            var authValue = Convert.ToBase64String(
                System.Text.Encoding.UTF8.GetBytes(
                    $"{secondaryCredential.Value.Username}:
{secondaryCredential.Value.Password}"));
            this.secondaryClient.DefaultRequestHeaders.Authorization =
                new System.Net.Http.Headers.AuthenticationHeaderValue("Basic",
authValue);

            Information("Secondary API configured");
        }
        else
        {
            Warning("Secondary API credential configured but failed to load");
        }
    }

    return await base.Initialize();
}
```

Pattern 3: Dynamic Credential Type with Helper Method

Create a helper method to abstract credential type detection:

```
public class SmartAuthModule : FlowModule<SmartAuthModuleSettings>
{
    private IAuthStrategy? authStrategy;

    public override async Task<IError?> Initialize()
    {
        if (!this.Settings.AuthCredential.HasValue)
        {
            return new Error("Authentication credential required");
        }

        this.authStrategy = await
this.CreateAuthStrategy(this.Settings.AuthCredential.Value);

        if (this.authStrategy is null)
        {
            return new Error("Failed to create authentication strategy. " +
                             "Credential type not supported or invalid.");
        }

        await this.authStrategy.ConfigureHttpClient(this.httpClient);

        return await base.Initialize();
    }

    private async Task<IAuthStrategy?> CreateAuthStrategy(Guid credentialId)
    {
        // Try API Key
        var apiKey = await this.GetCredentialContentAsync<CredentialWithApiKey>
(credentialId);
        if (!apiKey.IsError && apiKey.Value is not null)
        {
            Information("Using API Key authentication");
            return new ApiKeyAuthStrategy(apiKey.Value.ApiKey);
        }

        // Try Username/Password
        var userPass = await
this.GetCredentialContentAsync<CredentialWithUsernamePassword>(credentialId);
        if (!userPass.IsError && userPass.Value is not null)
        {
            Information("Using Basic authentication");
            return new BasicAuthStrategy(userPass.Value.Username,
userPass.Value.Password);
        }
```

```
        // Try Certificate
        var cert = await this.GetCredentialContentAsync<CredentialWithCertificate>
(credentialId);
        if (!cert.IsError && cert.Value is not null)
        {
            Information("Using Certificate authentication");
            return new CertificateAuthStrategy(cert.Value.X509Certificate2);
        }

        return null;
    }
}

// Strategy interface
public interface IAuthStrategy
{
    Task ConfigureHttpClient(HttpClient client);
}

// API Key strategy
public class ApiKeyAuthStrategy : IAuthStrategy
{
    private readonly string apiKey;

    public ApiKeyAuthStrategy(string apiKey) => this.apiKey = apiKey;

    public Task ConfigureHttpClient(HttpClient client)
    {
        client.DefaultRequestHeaders.Add("X-API-Key", this.apiKey);
        return Task.CompletedTask;
    }
}

// Basic auth strategy
public class BasicAuthStrategy : IAuthStrategy
{
    private readonly string username;
    private readonly string password;

    public BasicAuthStrategy(string username, string password)
    {
        this.username = username;
        this.password = password;
    }

    public Task ConfigureHttpClient(HttpClient client)
```

```
    {
        var authValue = Convert.ToBase64String(
            System.Text.Encoding.UTF8.GetBytes($"{this.username}:{this.password}"));
        client.DefaultRequestHeaders.Authorization =
            new System.Net.Http.Headers.AuthenticationHeaderValue("Basic",
authValue);
        return Task.CompletedTask;
    }
}

// Certificate auth strategy
public class CertificateAuthStrategy : IAuthStrategy
{
    private readonly X509Certificate2 certificate;

    public CertificateAuthStrategy(X509Certificate2 certificate) => this.certificate
= certificate;

    public Task ConfigureHttpClient(HttpClient client)
    {
        // Note: Certificate must be configured on HttpClientHandler before
creating HttpClient
        // This is a simplified example
        return Task.CompletedTask;
    }
}
```

Related Topics

- Module Settings - Settings validation and patterns
- Security Best Practices - Secure credential handling
- External Integrations - Using credentials with external services

>> Receive Messages

Receive Messages

The `MessageReceived` method is called whenever a message arrives at your module. This is where you implement your module's core logic.

> TIP
>
> Required namespaces:
>
> - `Crosser.EdgeNode.Flows.Models.Abstractions.Models`
> - `Crosser.EdgeNode.Flows`
> - `System.Threading.Tasks`

Implementation

For our random number module, we'll generate a random value and add it to the message:

```
public class RandomNumberModule : FlowModule<RandomNumberModuleSettings>
{
    public override string UserFriendlyName => "Random Number";

    private readonly Random random = new();

    public RandomNumberModule() : base(FlowModuleType.Function) { }

    protected override Task MessageReceived(IFlowMessage message)
    {
        // Generate random value based on settings
        var randomValue = this.random.Next(
            this.Settings.TargetMin,
            this.Settings.TargetMax);

        // Add value to message using the configured property name
        message.Set(this.Settings.TargetName, randomValue);

        // For now we just return null...
        return null;
    }
}
```

What we do with the value passed into the MessageReceived method is very specific for each module. This module is simple, we just add a new random value to a property decided by the settings.

>> Send Messages

Send Messages

Sending data to the next module(s) in a flow is easy, you just call the `Next` method. We continue the example from the step Receive Messages and all we need to do is to add the `async` keyword to the method and call `Next` instead of returning null.

```
private System.Random random;

protected override async Task MessageReceived(IFlowMessage message)
{
    // Generate random value based on settings
    var randomValue = this.random.Next(
        this.Settings.TargetMin,
        this.Settings.TargetMax);

    // Add value to message using the configured property name
    message.Set(this.Settings.TargetName, randomValue);


    // Pass the message to the next module(s) in the flow
    await this.Next(message);
}
```

This is the normal use case, but there are times when you want to pass data from other methods than `MessageReceived`. We will cover that later on.

>> Summary

Summary

In 4 steps we created a simple module, added settings with validation and learned how to receive/send messages. The code used in these steps is summarized below.

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using Crosser.EdgeNode.Flows.Models.Abstractions.Models;
using Crosser.EdgeNode.Common.Abstractions.Utilities.Validation;
using Crosser.EdgeNode.Flows;
using Crosser.EdgeNode.Flows.Abstractions;

namespace MyOrg.FlowModule.RandomNumberModule;

public class RandomNumberModule : FlowModule<RandomNumberModuleSettings>
{
    public override string UserFriendlyName => "Random Number";
    private readonly Random random = new();
    public RandomNumberModule() : base(FlowModuleType.Function) { }

    protected override async Task MessageReceived(IFlowMessage message)
    {
        // Generate random value based on settings
        var randomValue = this.random.Next(
            this.Settings.TargetMin,
            this.Settings.TargetMax);

        // Add value to message using the configured property name
        message.Set(this.Settings.TargetName, randomValue);

        // For now we just return null...
        // Pass the message to the next module(s) in the flow
        await this.Next(message);
    }
}

public class RandomNumberModuleSettings : FlowModuleSettings
{
    [Display(Name = "Minimum Value", Description = "The lowest value in the
random range")]
    [DefaultValue(1)]
    [Range(1, 10000)]
    public int TargetMin { get; set; } = 1;
```

```
    [Display(Name = "Maximum Value", Description = "The highest value in the
random range")]
    [DefaultValue(100)]
    [Range(1, 10000)]
    public int TargetMax { get; set; } = 100;

    [Display(Name = "Target Property", Description = "The property to store the
random value in")]
    [DefaultValue("data")]
    [MinLength(1)]
    [MaxLength(64)]
    public string TargetName { get; set; } = "data";

    public override void Validate(SettingsValidator validator)
    {
        // Built-in validators
        validator.Validate(nameof(this.TargetMin), this.TargetMin)
            .MinValue(1)
            .MaxValue(10000);

        validator.Validate(nameof(this.TargetMax), this.TargetMax)
            .MinValue(1)
            .MaxValue(10000);

        validator.Validate(nameof(this.TargetName), this.TargetName)
            .NotNull()
            .MinLength(1)
            .MaxLength(64);

        // Custom validation logic
        if (this.TargetMin >= this.TargetMax)
        {
            validator.AddError(nameof(this.TargetMin),
                "Minimum value must be less than maximum value");
        }
    }
}
```

You can now build your module with

```
dotnet build .\MyOrg.FlowModule.RandomNumberModule.csproj
```

You should of course test your modules (see Testing Your Module), but if you are happy with it you can deploy your module to Crosser Cloud.

[>> Deploy Your Module]

Deploy Modules

To use your custom module in Crosser Cloud, you need to package it and publish the package to NuGet.

Before creating the package, the module needs some documentation and an icon.

>> Add Module Documentation

Add Module Documentation

Modules should have documentation, a teaser, an icon, and a changelog. The documentation, teaser and changelog is written in markdown and the icon is a squared svg file.

Create a `docs` folder in the root of your project.

Documentation Files

For our `RandomNumberModule`, we need to add three files to the `docs` folder named after the module class:

- `RandomNumberModule.md` - Main documentation
- `RandomNumberModule.teaser.md` - Short description for hover tooltip
- `RandomNumberModule.svg` - Module icon.

> NOTE
>
> The filenames must match the module class name exactly (case sensitive).

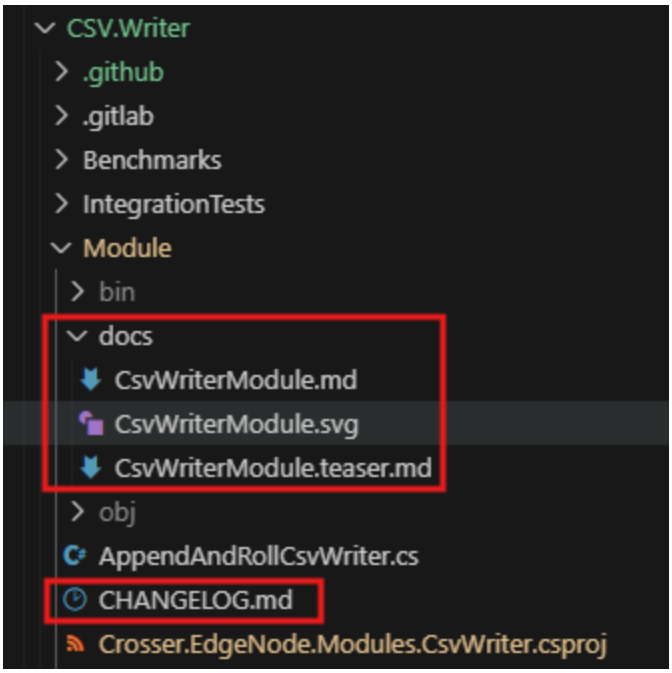
File Descriptions

- **`{classname}.md`**: Main documentation shown when you click a module in a flow
- **`{classname}.teaser.md`**: Teaser text shown when you hover over the module in the modules menu
- **`{classname}.svg`**: Icon representing the module in the UI (square format recommended)

Changelog

The changelog should be added in the same directory as the `csproj` file. Add a `CHANGELOG.md` file in the root of your project to document changes between versions. This helps users understand what's new, fixed, or changed in each release.

Project Layout Example



> TIP
>
> See our documentation example and changelog example for more inspiration, or view the documentation source and the changelog source.

>> Package Modules

Creating a Package

Creating a package for your module is easy.

First off you need to make sure the documentation files are included when creating the package. Add the following to your `*.csproj` file:

```
<ItemGroup>
  <Content Include="docs\**\*">
    <Pack>true</Pack>
    <PackagePath>docs\</PackagePath>
  </Content>
</ItemGroup>
<ItemGroup>
  <PackageReference Include="Crosser.EdgeNode.Flows" Version="5.0.0" />
</ItemGroup>
<ItemGroup>
  <Content Include="CHANGELOG.md">
    <Pack>true</Pack>
    <PackagePath>CHANGELOG.md</PackagePath>
  </Content>
</ItemGroup>
```

[!Warning]

> Make sure you do not reference `NJJsonSchema` directly in your project. The correct version will be included with the `Crosser.EdgeNode.Flows` package.

Now you can create the package using the `dotnet pack` command. This command will create a nuget package with version 1.0.0 for your project since we did not specify a version in the csproj file yet. You should run this command from the folder where your `*.csproj` file is located

```
dotnet pack .\MyOrg.FlowModule.RandomNumberModule.csproj -c release
```

You will now have a `.nupkg` package under `RandomNumberModule/bin/release`

TIP

Here is an example of package properties you can set in your .csproj file. Here we use VersionPrefix to set the version in order to add a suffix when we pack.

```
<PropertyGroup>
  <VersionPrefix>1.0.1</VersionPrefix>
  <TargetFramework>net9.0</TargetFramework>
  <PackageId>Crosser.EdgeNode.Module.Samples</PackageId>
  <Authors>Crosser Technologies</Authors>
  <PackageRequireLicenseAcceptance>false</PackageRequireLicenseAcceptance>
  <PackageReleaseNotes>N/A</PackageReleaseNotes>
  <Copyright>Copyright © 2016-2025 Crosser Technologies. All rights
reserved.</Copyright>
  <Company>Crosser Technologies</Company>
  <Product>Example Modules</Product>
</PropertyGroup>
<ItemGroup>
  <Content Include="docs\**\*">
    <Pack>true</Pack>
    <PackagePath>docs\</PackagePath>
  </Content>
</ItemGroup>
<ItemGroup>
  <Content Include="CHANGELOG.md">
    <Pack>true</Pack>
    <PackagePath>CHANGELOG.md</PackagePath>
  </Content>
</ItemGroup>
</Project>
```

When running `dotnet pack` with the configuration above a package with id `CrosserEdgeNode.Module.Samples` and version `1.0.1` will be created. Running `dotnet pack --version-suffix =beta1` will create a package with version `1.0.1-beta1`.

Please note that you will need the sections `ItemGroup` to get documentation and changelog included.

>> Upload Package

Upload Modules

To upload a package to Nuget you need an account, so make sure you got that (it is free).

Create an API key

Once logged in to Nuget you can click on your username and select `API keys`. Then generate a new key that can push new packages.

Upload Package

```
dotnet nuget push Crosser.FakeModule.0.0.1.nupkg -k <API_KEY> -s
https://api.nuget.org/v3/index.json
```

Nuget will now validate and index your package. You can see the process for the package if you are logged in to Nuget.

> WARNING
>
> You will not be able to take the next step until your package is indexed by Nuget, and that might take a while

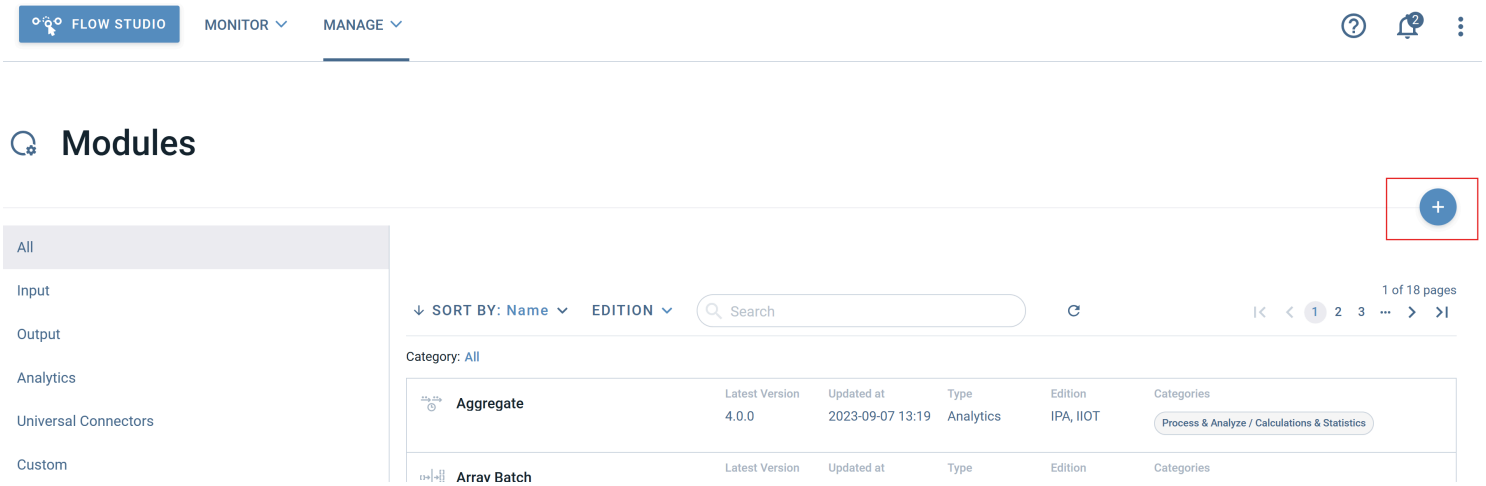
>> Register Package

Register Package

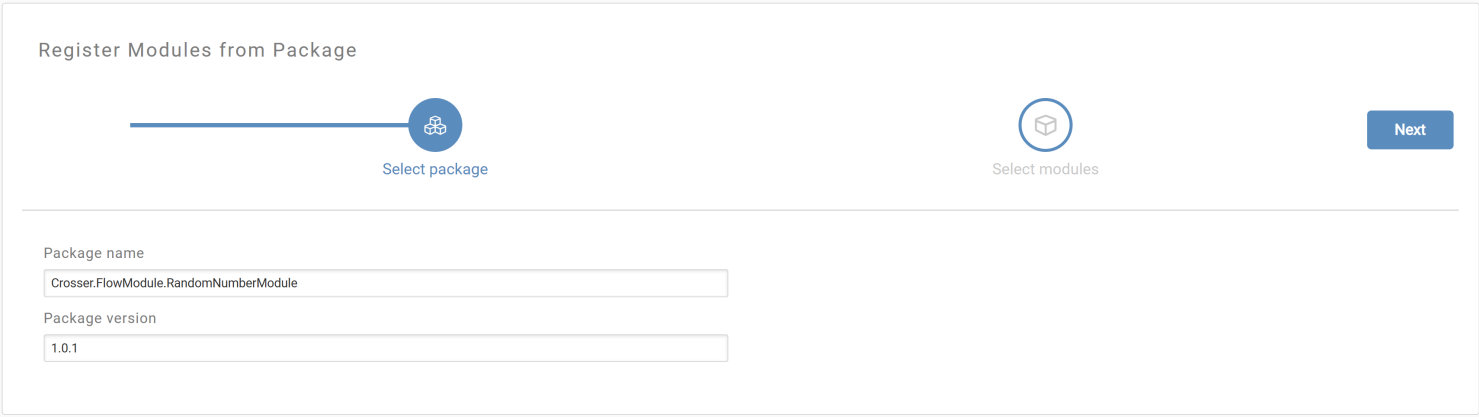
> NOTE
>
> You can proceed with this step when your package is indexed at Nuget

To be able to register your package you will need an account at Crosser Cloud with permission to register modules. If you do not have an account visit crosser.io and get in contact with us.

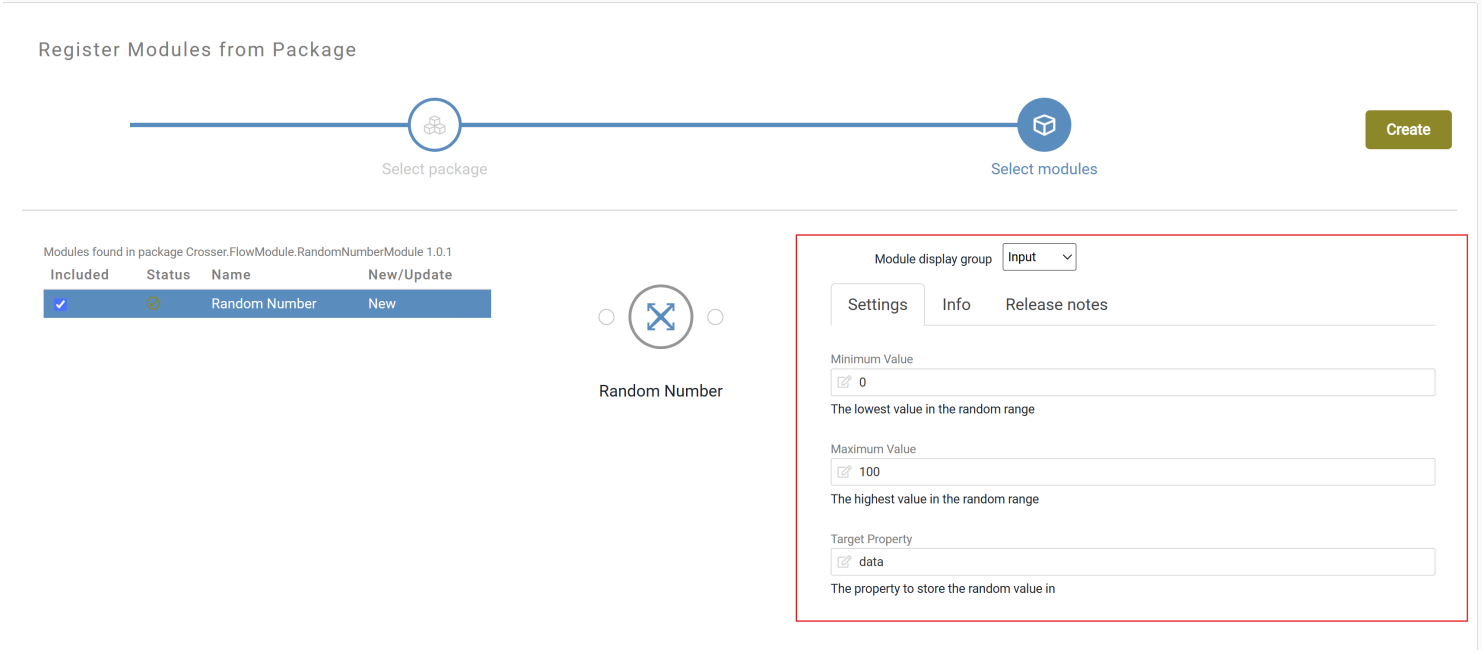
Once logged in, you navigate via the menu `Manage` and select `Modules`. Select the `+` button to register a new package.



Fill in the name and version number of your package and click on `Next` to download and analyze your package.



In the next step you will see your module discovered. You can review your settings, documentation and release notes.



Now just click `Create` and the package will be registered.

The package is now available in your list of modules in the FlowStudio.

> **NOTE**
>
> You will need to write the exact package name and a version that is uploaded to nuget.
>
> For example:
>
> - Name: Crosser.EdgeNode.Modules.Http
> - Version: 2.0.0
>
> You can´t register a package that is already registered

You can now try your new module in a flow!

[>> Try It]

Testing Your Modules

This section provides guidance on how to test custom modules you've developed for the Crosser platform. As a module developer, you'll need to validate that your modules work correctly both in isolation and as part of a complete flow.

Prerequisites

Before testing your modules, you should be familiar with:

- Installing and configuring the Crosser Node
- The Flow Studio
- Testing flows in remote sessions

For video tutorials on these concepts, check out the Crosser Academy.

Testing Approaches

There are several ways to test your custom modules:

1. Local Unit Testing and Integration Testing - Test module logic in isolation using the TestHelper package
2. Test the module in a flow - Test modules as part of a complete flow using remote sessions
3. Troubleshooting - Diagnose and resolve common module development issues

Quick Start

If you are already familiar with the Crosser Node setup and Flow Studio, jump directly to:

- Creating and debugging test flows
- Troubleshooting module issues

>> Build & Debug a Flow

Build & Debug a Flow

We will build a simple test flow to validate your custom module using just 2 modules:

- Data Generator (to provide test data to your module)
- Your Custom Module (the module you want to test and debug)

For your specific module, there may be other modules more suitable to generate the test data, such as an MQTT Subscriber, HTTP Listener, or File Reader. The Data Generator is a simple way to get started.

Prerequisites

Before you begin, ensure you have:

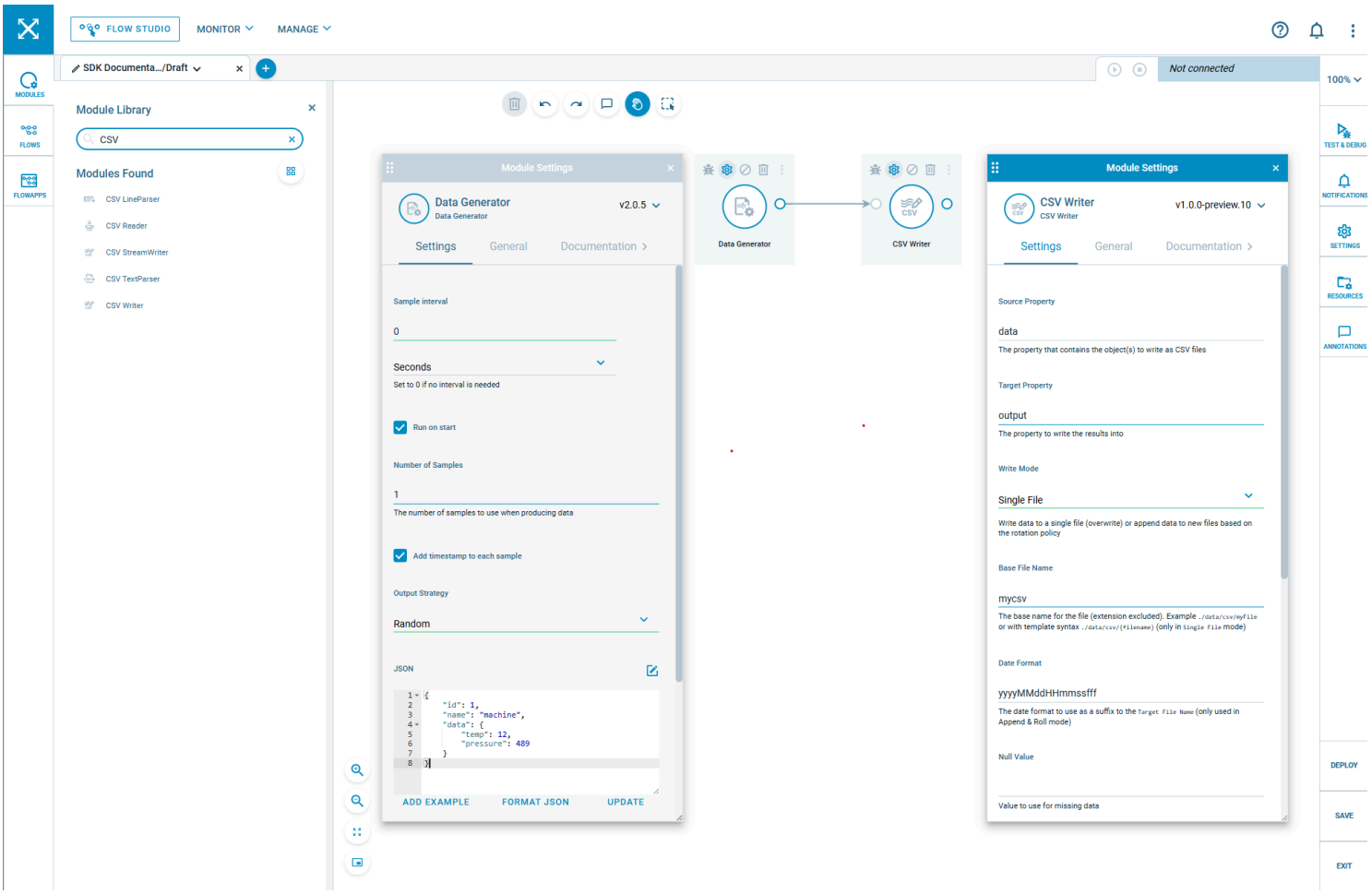
- A running Crosser Node where you have access to view the logs.
- The module is registered in Crosser Control Center.
- The module package is uploaded to nuget.org.

When the flow is built, we will open a remote session to the Node from Flow Studio. The flow will be temporarily deployed to the Node, allowing you to see real-time data flow and debug information directly in the browser.

Building the Test Flow

Step 1: Create a New Flow

1. Open Flow Studio in Crosser Control Center. A new flow draft will appear automatically.
2. Find the Data Generator and your custom module in the module palette on the left.
3. Drag both modules onto the canvas.
4. Connect the Data Generator's output to your custom module's input by drawing a line between them.



Step 2: Configure the Data Generator

The Data Generator will provide test data for your module. Configure it to generate data that matches what your module expects:

1. Click on the Data Generator module
2. In the settings panel, click on the ADD EXAMPLE button and configure the JSON field with a JSON structure that your module can process.
3. Set an appropriate Interval . Use `0` in case only one message is needed for testing.

Example data configuration:

```
{
 "id": 1,
 "name": "machine",
 "data": {
  "temp": 12,
  "pressure": 489
 }
}
```

Step 3: Configure Your Custom Module

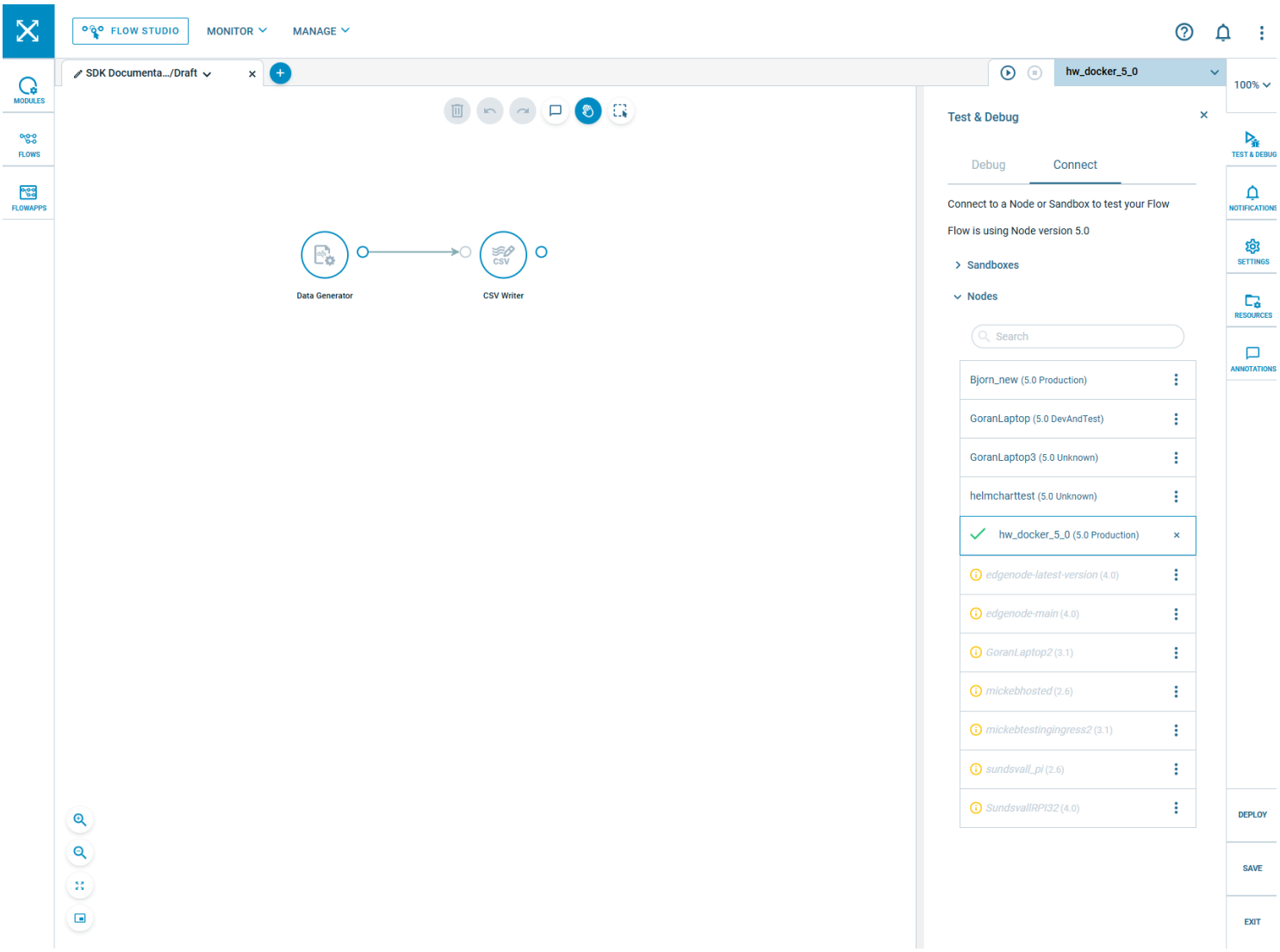
1. Click on your custom module
2. Configure any required settings in the settings panel including the source property. This setting tells your module where to find the input data in the incoming message.
3. Set up any credentials or resources your module needs
4. You can read the documentation for your module by selecting the Documentation tab
5. Save the flow with a descriptive name like "Test MyModule". If the flow is only for personal testing, you can make it private.

Start Testing Your Module

To test our module we will run it in Test & Debug mode connected to your Crosser Node.

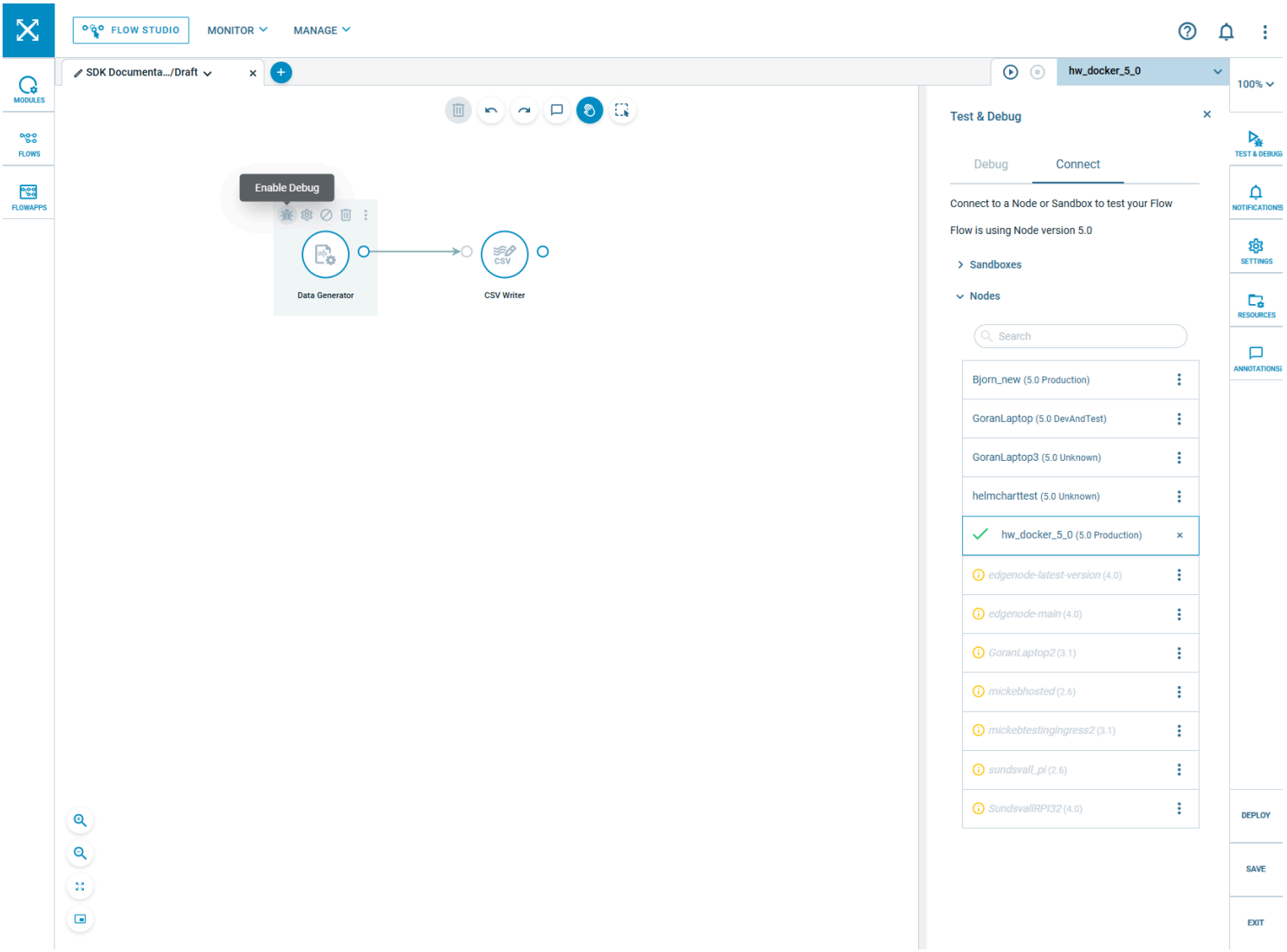
Step 1: Connect to Your Node

1. Go to the Test & Debug panel in Flow Studio
2. Select your Node from the list of available nodes
3. The connection indicator should show that you're connected



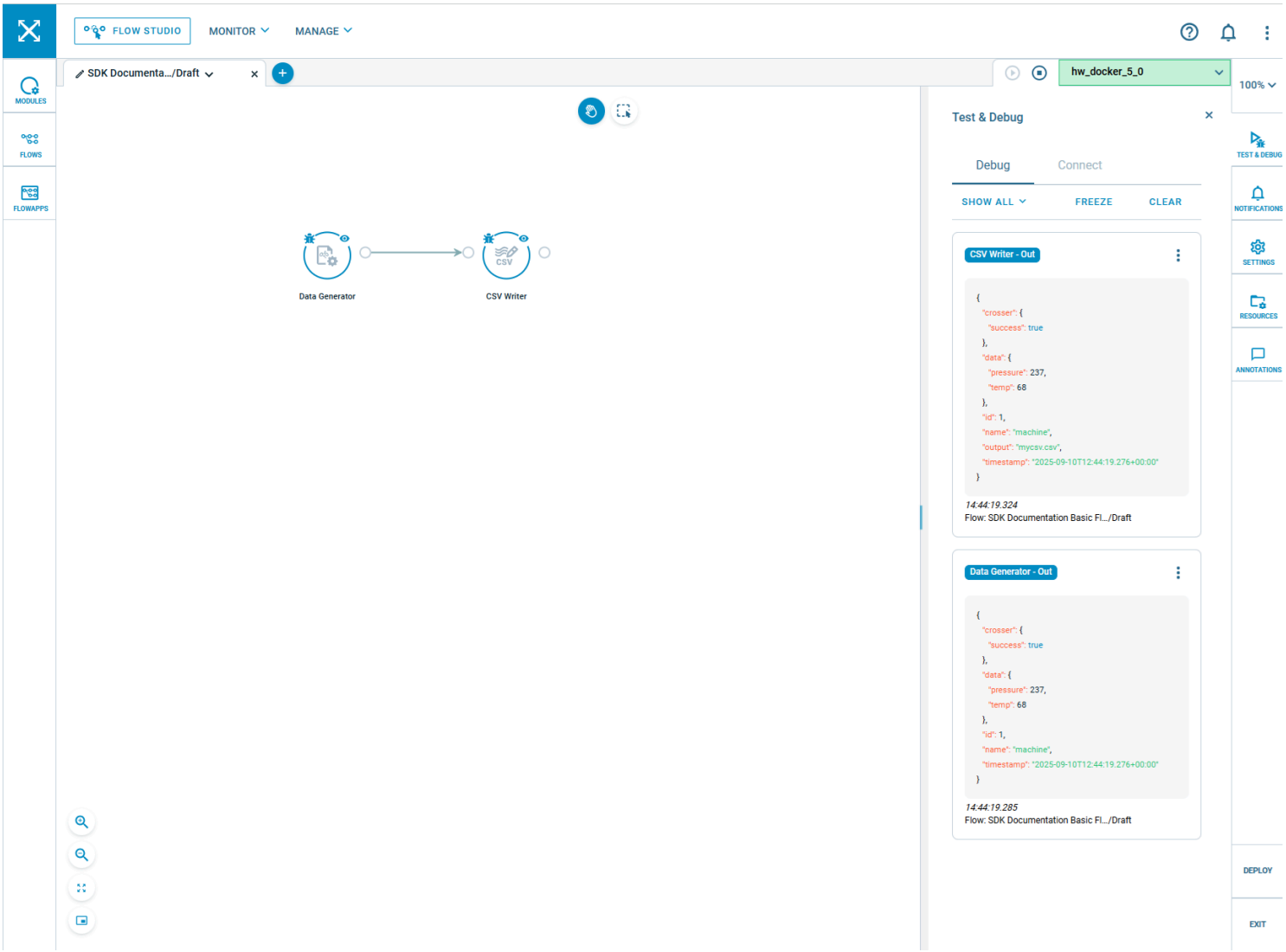
Step 2: Configure Debug On Your Module

You can enable debug output on your module by clicking on the debug icon on the module in the flow. This will allow you to see debug messages in the debug panel when the flow is running.



Step 3: Start the Remote Session

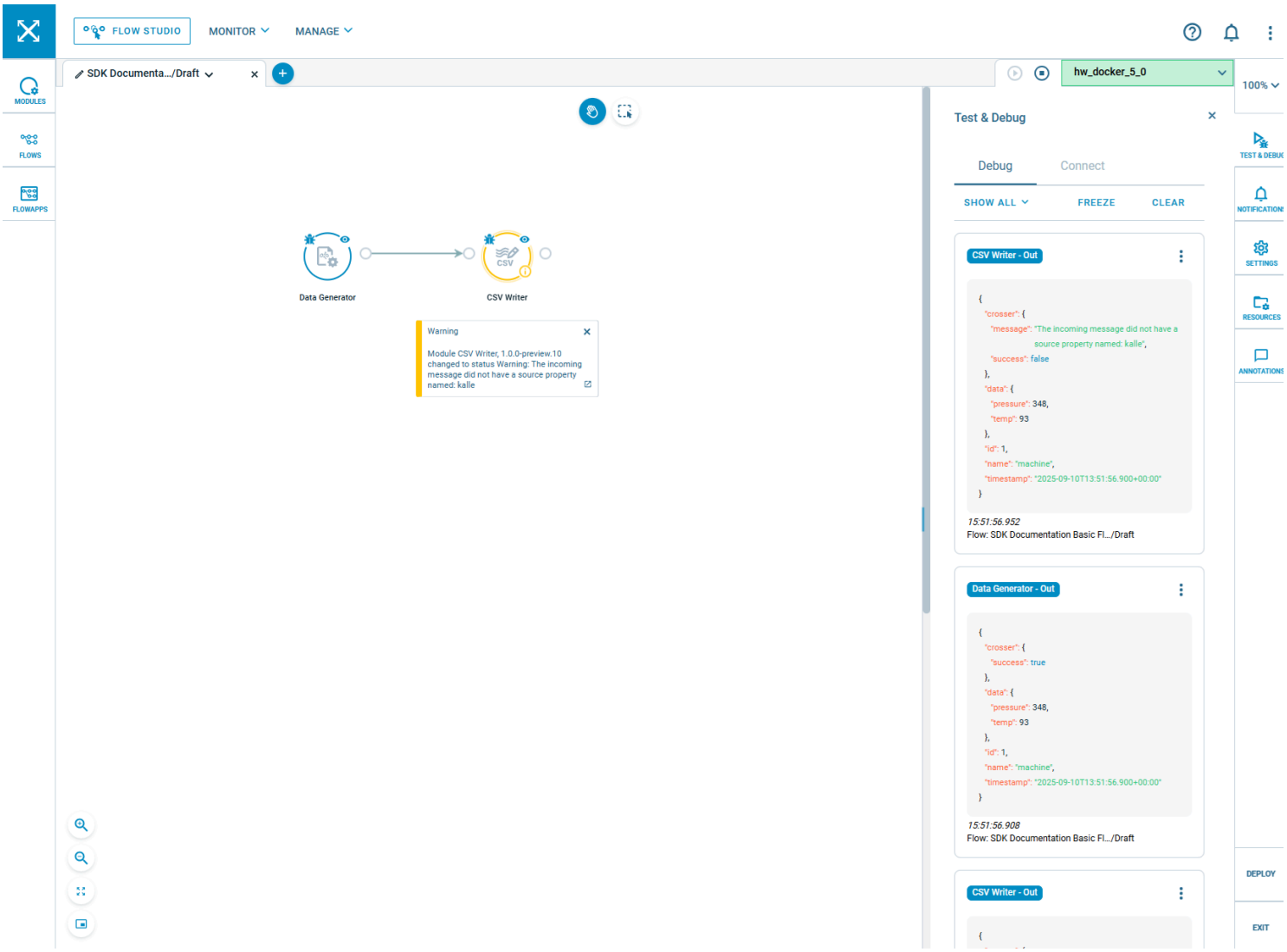
1. Click the Play button in the Test & Debug toolbar to start the remote session
2. The flow will be deployed to your Node (this may take a few seconds as your flow package is downloaded to the Node.)
3. Once the flow is running you will receive messages in the debug panel from the modules where debug is enabled.



Step 4 Monitor Your Module

Once the flow is running, you can monitor your module in real-time:

Debug Panel shows messages flowing through your module. Notifications Any debug or error messages from your module. Module Status: Each module shows a yellow status indicator when a warning is present. If there is an error the flow will be stopped.



Debugging Your Module

Common Issues and Solutions

Module Not Starting:

- Verify all required settings and credentials are configured correctly.
- Look for error messages in the debug panel.
- Check the Node logs for issues.

No Output Data:

- Verify the input module is producing messages (check debug panel).
- Ensure your module is properly processing input messages.
- Check for exceptions that are ignored in your module code.

Unexpected Output:

- Compare input vs output messages in the debug panel.

- View the types of the input and output data by using a debug module in the flow.
- Add debug logging to your module code to trace data transformation.
- Verify your module logic handles the input data format correctly. The simplest way is to reproduce the error in a unit test using the same input data.

Iterating on Your Module

1. Make Changes: Update your module code based on test results.
2. Redeploy: Upload the updated module package to nuget (it can take some time for nuget to index the package) and then register it in Crosser Control Center.
3. Update Flow: In Flow Studio, update your flow to use the latest version of your module.
4. Restart Session: Restart the remote session to use the new version.
5. Retest: Verify your changes work as expected.

Advanced Testing

For more comprehensive testing:

- Vary Input Data: Modify the Data Generator to test different scenarios.
- Load Testing: Increase the Data Generator frequency to test performance.
- Error Conditions: Configure the Data Generator to send invalid data to test error handling.
- Add Debug Module: Insert a Debug module after your custom module to capture and inspect all output.

Next Steps

Once you've validated your module works correctly in this simple flow:

1. Test it in more complex flows with multiple modules.
2. Monitor performance and logs in a production-like environment.

For troubleshooting complex issues, see the Module Troubleshooting Guide.

Module Troubleshooting Guide

This guide helps you diagnose and resolve common issues when developing and testing custom modules for the Crosser Node.

Debugging Strategies

1. Start with Local Unit Tests

Always begin troubleshooting with local unit tests using the TestHelper package:

- Isolate the problem to your module logic
- Test with known input data
- Verify expected outputs
- Check error handling

See testing for detailed guidance.

2. Use Debug Module in Flows

When testing in remote sessions you can include a Debug module.

- Add it to capture both input and output
- Monitor message flow and data types in real-time

3. Add Comprehensive Logging

You can use the built-in static `ILog` instance to log messages at different levels. See logging for more information on logging.

Common Issues and Solutions

Module Failing to Start

Symptoms:

- Module status shows "Error" in remote sessions
- Module doesn't process any messages
- Startup exceptions in logs

Common Causes:

1. Initialization Issues Check if there might be issues in the Initialization code in the module.
2. Resource Access Issues
    - Verify credentials are properly configured

- Check network connectivity for external resources
- Validate file paths and permissions

No Message Processing

Symptoms:

- Module starts successfully
- No output messages produced
- Debug Module shows no activity

Debugging Steps:

1. Check Input Messages
   Is the `Source Property` configured correctly? Add logging to confirm messages are received and have the expected format:

```
protected override async Task MessageReceived(IFlowMessage message)
{
     Logger?.Information("Received message: {@Message}", message);

     // Log message properties
     foreach (var prop in message.GetProperties())
     {
          Logger?.Debug("Property {Key}: {Value}", prop.Key, prop.Value);
     }

     // Your processing logic here
}
```

2. Verify Message Routing
   - Check flow connections in Flow Studio
   - Ensure your module is connected to data sources
   - Verify upstream modules are producing messages
3. Check Error Handling Make sure exceptions are handled correctly.

```
protected override async Task MessageReceived(IFlowMessage message)
{
    try
    {
        await ProcessMessage(message);
    }
    catch (Exception ex)
```

```
        {
            var error = $"The incoming message could not be
    processed {ex.Message}";
            SetStatus(Status.Warning, error); // Or SetError if so critical that
    the flow should stop/restart
            message.SetError(error);
        }
        finally
        {
            await Next(message);
        }
    }
```

Performance Issues

Symptoms:

- Slow message processing
- Growing message queues
- High CPU or memory usage

Optimization Strategies:

1. Async/Await Patterns

```
// ❌ Blocking - blocks the thread
protected override async Task MessageReceived(IFlowMessage message)
{
    var result = SomeSlowOperation(); // Synchronous call
    await Send(result);
}

// ✅ Non-blocking - proper async
protected override async Task MessageReceived(IFlowMessage message)
{
    var result = await SomeSlowOperationAsync(); // Asynchronous call
    await Send(result);
}
```

2. Resource Management

```
public class MyModule : FlowModule<MyModuleSettings>, IDisposable
{
    private readonly HttpClient httpClient;
```

```
// ❌ Recreates instance for every message
protected override async Task MessageReceived(IFlowMessage message)
{
    this.httpClient = new HttpClient(); // Creates a new instance for every
message - bad!
    await this.httpClient.GetAsync("http://example.com");
}

}
   // ✅ One time initialization
public override async Task<IError?> Initialize()
{
    this.httpClient = new HttpClient(); // Create once
    return await base.Initialize();
}

 protected override async Task MessageReceived(IFlowMessage message)
{
    await this.httpClient.GetAsync("http://example.com");
}

protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        this.httpClient?.Dispose(); // Dispose when module is disposed
    }
    base.Dispose(disposing);
}
```

State Management Issues

Improper state management can lead to unpredictable behavior, especially when multiple instances of the same module exist in a flow or when using timers.

Shared Static State Between Module Instances

Problem: Using static variables causes multiple instances of the same module to share state, leading to unexpected behavior.

```
// ❌ WRONG - Static variables are shared across all instances
public class CounterModule : FlowModule<CounterModuleSettings>
{
```

```
    private static int counter = 0; // This is shared across ALL instances!

    protected override async Task MessageReceived(IFlowMessage message)
    {
        counter++; // All instances increment the same counter
        var result = new FlowMessage();
        result.SetValue("count", counter);
        await Send(result);
    }
}
```

Issue: If you have two CounterModule instances in the same flow, they will share the same `counter` variable, causing unpredictable counting behavior.

Solution: Use instance variables instead:

```
// ✅ CORRECT - Each instance has its own state
public class CounterModule : FlowModule<CounterModuleSettings>
{
    private int counter = 0; // Each instance has its own counter

    protected override async Task MessageReceived(IFlowMessage message)
    {
        this.counter++; // Only this instance's counter is incremented
        var result = new FlowMessage();
        result.SetValue("count", this.counter);
        await Send(result);
    }
}
```

Best Practices for State Management

1. Avoid Static Variables: Always use instance variables unless you specifically need shared state across all module instances in a flow.

2. Use Thread-Safe Collections: When using timers or background tasks, protect shared state with locks or use concurrent collections:

```
    private readonly ConcurrentQueue<IFlowMessage> messageQueue = new();
```

3. Separate Concerns: Use different variables for different purposes (input storage, processing state, output generation).

4. Clean Up Resources: Always dispose of timers and other resources properly.
5. Test with Multiple Instances: When testing your module, try using multiple instances in the same flow to verify they don't interfere with each other.

Advanced Debugging

Performance Profiling

Use profiling tools to identify bottlenecks:

1. Built-in Metrics:
    - Monitor processing times in logs
    - Track message queue sizes
    - Watch memory usage patterns
2. Custom Metrics:

```
private readonly Stopwatch processingTimer = new();

protected override async Task MessageReceived(IFlowMessage message)
{
    this.processingTimer.Restart();

    try
    {
        await ProcessMessage(message);
    }
    finally
    {
        this.processingTimer.Stop();
        Logger?.Debug("Processing took {ElapsedMs}ms",
this.processingTimer.ElapsedMilliseconds);
    }
}
```

Memory Leak Detection

Watch for common memory leak patterns:

1. Event Handler Leaks:

```
// ❌ Can cause memory leaks
public MyModule(ILog logger) : base(FlowModuleType.Function)
{
```

```
        SomeService.OnEvent += HandleEvent; // Never unsubscribed
    }

    // ✅ Proper cleanup
    protected override void Dispose(bool disposing)
    {
        if (disposing)
        {
            SomeService.OnEvent -= HandleEvent;
        }
        base.Dispose(disposing);
    }
```

2. Collection Growth:

```
    // ❌ Unbounded collection growth
    private readonly List<ProcessedData> allProcessedData = new();

    // ✅ Bounded collection with cleanup
    private readonly Queue<ProcessedData> recentData = new();
    private const int MAX_ITEMS = 1000;

    private void AddProcessedData(ProcessedData data)
    {
        this.recentData.Enqueue(data);
        while (this.recentData.Count > MAX_ITEMS)
        {
            this.recentData.Dequeue();
        }
    }
```

Testing Checklist

Use this checklist to systematically test your modules:

Unit Testing

- [ ] Module instantiates correctly
- [ ] Handles valid input data
- [ ] Produces expected output format
- [ ] Handles null/empty input gracefully
- [ ] Throws appropriate exceptions for invalid input
- [ ] Resource cleanup works properly

Integration Testing

- [ ] Module starts in Node environment
- [ ] Processes messages from upstream modules
- [ ] Outputs connect properly to downstream modules
- [ ] Performance is acceptable under normal load
- [ ] Logging provides useful information
- [ ] Error conditions are handled appropriately

Edge Case Testing

- [ ] Very large messages
- [ ] High message frequency
- [ ] Network connectivity issues (if applicable)
- [ ] Node restart/reconnection
- [ ] Memory pressure conditions
- [ ] Long-running stability

Getting Help

If you're still experiencing issues:

1. Check the Logs: Node logs contain detailed error information
2. Test Isolation: Use unit tests to isolate the problem
3. Simplify: Create a minimal reproduction case
4. Documentation: Review the SDK documentation and examples
5. Community: Reach out to the Crosser community for support

Remember: good logging is your best friend when troubleshooting module issues!

Testing Overview

The Crosser EdgeNode TestHelpers SDK simplifies unit testing of Crosser EdgeNode modules. Since modules use internal queues, it's challenging to send data and await results in unit tests. This SDK solves that problem with a fluent testing API.

What You'll Learn

- Create Test Project - Set up an xUnit test project
- Quick Start - Write your first test
- Core Concepts - Understand TestFlow and the fluent API
- Testing Guide - Comprehensive API reference
- Advanced Testing - Complex scenarios and patterns
- Best Practices - Tips for effective testing
- Troubleshooting - Common issues and solutions

Key Features

- Automatic flow management with start/stop lifecycle
- Fluent API for configuration and assertions
- Multiple output support for complex modules
- Timeout configuration for different scenarios
- Credential and resource injection

>> Create Test Project

Create Test Project

Start by creating an xUnit (or the testing framework of your choice) test project for testing your Crosser EdgeNode modules.

Create the Project

```
dotnet new xunit -n MyOrg.FlowModule.RandomNumberModule.Tests
cd MyOrg.FlowModule.RandomNumberModule.Tests
```

Add References

Add a reference to your module project:

```
dotnet add reference
../MyOrg.FlowModule.RandomNumberModule/MyOrg.FlowModule.RandomNumberModule.csproj
```

Add the TestHelpers package:

```
dotnet add package Crosser.EdgeNode.Modules.Testing
```

> TIP
>
> Use the version matching your SDK version. Available for SDK 2.5.0+

Project Structure

```
MyOrg.FlowModule.RandomNumberModule.Tests/
├── MyOrg.FlowModule.RandomNumberModule.Tests.csproj
├── RandomNumberModuleTests.cs
└── Usings.cs
```

>> Quick Start

Quick Start

Write your first module test in minutes.

Your First Test

Create `RandomNumberModuleTests.cs`:

```
using Xunit;
using Crosser.EdgeNode.Modules.Testing;
using Crosser.EdgeNode.Flows.Models.Abstractions.Models;

namespace MyOrg.FlowModule.RandomNumberModule.Tests;

public class RandomNumberModuleTests
{
    [Fact]
    public async Task GeneratesRandomNumberInRange()
    {
        // Arrange
        var module = new RandomNumberModule();
        module.Settings.TargetMin = 10;
        module.Settings.TargetMax = 20;
        module.Settings.TargetName = "randomValue";

        var flow = module.SetupFlow();

        // Act
        await flow.Start();
        await flow.Receive(new FlowMessage());
        var result = await flow.GetNextResult();
        await flow.Stop();

        // Assert
        Assert.True(result.Has("randomValue"));
        var value = result.Get<int>("randomValue");
        Assert.InRange(value, 10, 19);
    }
}
```

Run the Test

```
dotnet test
```

Expected output:

```
Passed!  - Failed:     0, Passed:     1, Skipped:     0, Total:     1
```

Understanding the Test

1. Arrange - Create and configure your module, setup test flow
2. Act - Start flow, send message, retrieve result, stop flow
3. Assert - Verify the module behaved correctly

Key Methods

- `SetupFlow()` - Wraps module in test harness
- `Start()` - Initializes and starts the module
- `Receive(message)` - Sends message to module
- `GetNextResult()` - Retrieves processed message
- `Stop()` - Stops the module

Next Steps

- Core Concepts - Understand TestFlow in depth
- Testing Guide - Complete API reference
- Advanced Testing - Complex scenarios

>> Core Concepts

Core Concepts

Understand the key concepts behind the TestHelpers SDK.

TestFlow

The main testing orchestrator that wraps your module and provides testing capabilities.

Key Features

- Automatic flow management with start/stop lifecycle
- Message routing between test modules
- Timeout configuration for different testing scenarios
- Multiple output support for complex modules

Creating a TestFlow

```
var module = new MyModule();
var flow = module.SetupFlow(timeout: 10000, numberOfOutputs: 1);
```

Parameters:

- `timeout` - Maximum time to wait for operations in milliseconds (default: 10,000ms)
- `numberOfOutputs` - Number of output channels to create (default: 1)

Fluent API

The SDK provides a fluent interface for configuring your test environment.

Dependency Injection

Add credentials and resources to your module:

```
var flow = module.SetupFlow()
    .WithCredential(new MyCredential { Id = credentialId })
    .WithResource(new MyResource { Id = resourceId });
```

Method Chaining

Build complex test scenarios with method chaining:

```
var flow = module.SetupFlow(timeout: 30000)
    .WithCredential(credential)
    .WithResource(resource);
```

```
await flow.Start();
await flow.Receive(message);
var result = await flow.GetNextResult();
await flow.Stop();
```

Test Lifecycle

Every test follows a consistent lifecycle:

1. Setup - Create module and configure settings
2. Arrange - Setup test flow with dependencies
3. Start - Initialize and start the module
4. Act - Send messages and retrieve results
5. Stop - Clean shutdown of the module
6. Assert - Verify expected behavior

Lifecycle Example

```
[Fact]
public async Task TestModuleLifecycle()
{
    // 1. Setup
    var module = new MyModule();
    module.Settings.Property = "value";

    // 2. Arrange
    var flow = module.SetupFlow();

    // 3. Start
    var startResult = await flow.Start();
    Assert.True(startResult.IsSuccess);

    // 4. Act
    await flow.Receive(new FlowMessage());
    var result = await flow.GetNextResult();

    // 5. Stop
    var stopResult = await flow.Stop();
    Assert.True(stopResult.IsSuccess);

    // 6. Assert
    Assert.NotNull(result);
}
```

Message Flow

Understanding how messages flow through your module during testing:

```
Test Code → Receive() → Module Processing → Output Queue → GetNextResult()
```

The TestFlow manages the output queue, allowing you to retrieve results asynchronously without dealing with internal module queues.

>> Testing Guide

Testing Guide

Comprehensive API reference for the TestHelpers SDK.

SetupFlow Method

Creates a TestFlow instance for testing your module.

```
public static TestFlow SetupFlow(this FlowModule module, int timeout = 10000, int
numberOfOutputs = 1)
```

Parameters:

- `timeout` - Maximum time to wait for operations in milliseconds (default: 10 seconds)
- `numberOfOutputs` - Number of output channels to create (default: 1)

Returns: A `TestFlow` instance for further configuration

Example:

```
var module = new RandomNumberModule();
var flow = module.SetupFlow();
```

Flow Control Methods

Start

Initializes and starts the module.

```
public static async Task<IResult> Start(this TestFlow testFlow)
```

Example:

```
var startResult = await flow.Start();
Assert.True(startResult.IsSuccess);
```

Stop

Stops the module gracefully.

```
public static async Task<IResult> Stop(this TestFlow testFlow)
```

Example:

```
var stopResult = await flow.Stop();
Assert.True(stopResult.IsSuccess);
```

Receive

Sends a message to the module for processing.

```
public static async Task Receive(this TestFlow testFlow, IFlowMessage message)
```

Example:

```
var m = new FlowMessage();
m.Set("data.A", "Hello");
m.Set("data.B", 5.0190582275390625m);

await flow.Receive(m);
```

Result Retrieval Methods

GetNextResult

Retrieves the next message from the module's output.

```
// Get next result from first output
public static async Task<IFlowMessage> GetNextResult(this TestFlow testFlow)

// Get next result from specific output
public static async Task<IFlowMessage> GetNextResult(this TestFlow testFlow,
int outputModuleIndex)
```

> NOTE
>
> `GetNextResult` will wait for data to be sent out from the module. You can pass in many messages and process the result one by one.

Example:

```
var result = await flow.GetNextResult();
Assert.True(result.IsSuccess);
```

GetNextResults

Collects multiple results from the module's output.

```
// Get multiple results from first output
public static async Task<IList<IFlowMessage>> GetNextResults(this TestFlow testFlow,
int numberOfResults)

// Get multiple results from specific output
public static async Task<IList<IFlowMessage>> GetNextResults(this TestFlow testFlow,
int numberOfResults, int outputModuleIndex)
```

Example:

```
[Fact]
public async Task CanGetMultipleResults()
{
    var module = new RandomNumberModule();
    module.Settings.TargetMin = 1;
    module.Settings.TargetMax = 100;
    module.Settings.TargetName = "randomValue";

    var flow = module.SetupFlow();

    await flow.Start();

    var m = new FlowMessage();
    for (var i = 0; i < 10; i++)
    {
        await flow.Receive(m.Clone());
    }

    var results = await flow.GetNextResults(10);
    Assert.Equal(10, results.Count);

    // Verify all results have random values
    foreach (var result in results)
    {
        Assert.True(result.Has("randomValue"));
        Assert.InRange(result.Get<int>("randomValue"), 1, 99);
    }
```

```
    await flow.Stop();
}
```

GetNthResult

Retrieves a specific result by index (0-based).

```
// Get the nth result from first output
public static async Task<IFlowMessage> GetNthResult(this TestFlow testFlow,
int index)

// Get the nth result from specific output
public static async Task<IFlowMessage> GetNthResult(this TestFlow testFlow, int
index, int outputModuleIndex)
```

Example:

```
[Fact]
public async Task CanGetNthResult()
{
    var module = new RandomNumberModule();
    module.Settings.TargetMin = 0;
    module.Settings.TargetMax = 100;
    module.Settings.TargetName = "value";

    var flow = module.SetupFlow(timeout: 50000);

    await flow.Start();

    var m = new FlowMessage();
    for (var i = 0; i < 10; i++)
    {
        await flow.Receive(m.Clone());
    }

    // Get the 6th result (0-based index)
    var result = await flow.GetNthResult(5);
    Assert.True(result.Has("value"));
    Assert.InRange(result.Get<int>("value"), 0, 99);

    await flow.Stop();
}
```

Dependency Injection Methods

WithCredential

Adds credentials to your module for testing.

```
// From a credential object
flow.WithCredential(new MyCredential { Id = credentialId, ApiKey = "test-key" });

// From file
flow.WithCredential<MyCredential>("path/to/credential.json");
```

Example:

```
var credential = new DatabaseCredential
{
    Id = Guid.NewGuid(),
    ConnectionString = "test-connection-string"
};

var flow = module.SetupFlow()
    .WithCredential(credential);
```

WithResource

Adds resources to your module for testing.

```
flow.WithResource(new MyResource { Id = resourceId, Data = "test-data" });
```

Example:

```
var resource = new FileResource
{
    Id = Guid.NewGuid(),
    FileName = "test-file.txt",
    Data = Encoding.UTF8.GetBytes("test content")
};

var flow = module.SetupFlow()
    .WithResource(resource);
```

>> Advanced Testing

Advanced Testing

Advanced testing scenarios for complex modules.

Testing Modules with Multiple Outputs

Some modules send messages to different outputs based on conditions. Configure multiple outputs when setting up your test flow:

```
[Fact]
public async Task TestModuleWithMultipleOutputs()
{
    var module = new MyMultiOutputModule();
    var flow = module.SetupFlow(numberOfOutputs: 2);

    await flow.Start();

    await flow.Receive(new FlowMessage());

    // Get results from different outputs
    var result1 = await flow.GetNextResult(outputModuleIndex: 0);
    var result2 = await flow.GetNextResult(outputModuleIndex: 1);

    await flow.Stop();
}
```

Testing Modules with Credentials

Inject credentials into your module for testing external integrations:

```
[Fact]
public async Task TestModuleWithCredentials()
{
    var module = new MyModule();
    var credential = new DatabaseCredential
    {
        Id = Guid.NewGuid(),
        ConnectionString = "test-connection-string"
    };

    var flow = module.SetupFlow()
        .WithCredential(credential);

    await flow.Start();
```

```
    var message = new FlowMessage();
    message.Set("query", "SELECT * FROM users");
    await flow.Receive(message);

    var result = await flow.GetNextResult();
    Assert.True(result.IsSuccess);

    await flow.Stop();
}
```

Loading Credentials from File

```
var flow = module.SetupFlow()
    .WithCredential<MyCredential>("test-credentials.json");
```

Testing Modules with Resources

Inject resources like configuration files or templates:

```
[Fact]
public async Task TestModuleWithResources()
{
    var module = new MyModule();
    var resource = new FileResource
    {
        Id = Guid.NewGuid(),
        FileName = "test-file.txt",
        Data = Encoding.UTF8.GetBytes("test content")
    };

    var flow = module.SetupFlow()
        .WithResource(resource);

    await flow.Start();

    var message = new FlowMessage();
    await flow.Receive(message);

    var result = await flow.GetNextResult();
    Assert.Contains("test content", result.Get<string>("output"));

    await flow.Stop();
}
```

Batch Message Testing

Test modules that process multiple messages:

```
[Fact]
public async Task TestBatchProcessing()
{
    var module = new RandomNumberModule();
    module.Settings.TargetMin = 1;
    module.Settings.TargetMax = 1000;
    module.Settings.TargetName = "randomId";

    var flow = module.SetupFlow(timeout: 30000);

    await flow.Start();

    // Send multiple messages
    for (int i = 0; i < 10; i++)
    {
        var message = new FlowMessage();
        message.Set("data.index", i);
        await flow.Receive(message);
    }

    // Collect all results
    var results = await flow.GetNextResults(10);
    Assert.Equal(10, results.Count);

    // Verify each result has both original and new data
    for (int i = 0; i < 10; i++)
    {
        Assert.Equal(i, results[i].Get<int>("data.index"));
        Assert.True(results[i].Has("randomId"));
    }

    await flow.Stop();
}
```

Testing Specific Result Position

Test that a specific message in a sequence is processed correctly:

```
[Fact]
public async Task TestSpecificResultPosition()
{
```

```
    var module = new RandomNumberModule();
    module.Settings.TargetMin = 0;
    module.Settings.TargetMax = 100;
    module.Settings.TargetName = "value";

    var flow = module.SetupFlow();

    await flow.Start();

    // Send multiple messages with sequence numbers
    for (int i = 0; i < 5; i++)
    {
        var message = new FlowMessage();
        message.Set("sequence", i);
        await flow.Receive(message);
    }

    // Get the 3rd result (0-indexed)
    var thirdResult = await flow.GetNthResult(2);
    Assert.Equal(2, thirdResult.Get<int>("sequence"));
    Assert.True(thirdResult.Has("value"));

    await flow.Stop();
}
```

Complete Test Class Example

A full test class showing best practices:

```
using Xunit;
using Crosser.EdgeNode.Modules.Testing;

public class RandomNumberModuleTests : IDisposable
{
    private readonly RandomNumberModule module;
    private readonly TestFlow flow;

    public RandomNumberModuleTests()
    {
        this.module = new RandomNumberModule();
        this.module.Settings.TargetMin = 0;
        this.module.Settings.TargetMax = 100;
        this.module.Settings.TargetName = "randomValue";
        this.flow = this.module.SetupFlow(timeout: 15000);
    }
```

```
[Fact]
public async Task GeneratesRandomNumberInRange()
{
    await this.flow.Start();

    var message = new FlowMessage();
    await this.flow.Receive(message);

    var result = await this.flow.GetNextResult();
    Assert.True(result.Has("randomValue"));
    var value = result.Get<int>("randomValue");
    Assert.InRange(value, 0, 99);

    await this.flow.Stop();
}

[Fact]
public async Task HandlesMultipleMessages()
{
    await this.flow.Start();

    for (int i = 0; i < 3; i++)
    {
        var message = new FlowMessage();
        message.Set("data.sequence", i);
        await this.flow.Receive(message);
    }

    var results = await this.flow.GetNextResults(3);
    Assert.Equal(3, results.Count);

    // Verify each result has both original and generated data
    for (int i = 0; i < 3; i++)
    {
        Assert.Equal(i, results[i].Get<int>("data.sequence"));
        Assert.True(results[i].Has("randomValue"));
    }

    await this.flow.Stop();
}

[Fact]
public async Task UsesConfiguredPropertyName()
{
    // Change the target property name
```

```
        this.module.Settings.TargetName = "sensor.temperature";

        var testFlow = this.module.SetupFlow();

        await testFlow.Start();

        var message = new FlowMessage();
        await testFlow.Receive(message);

        var result = await testFlow.GetNextResult();
        Assert.True(result.Has("sensor.temperature"));
        Assert.False(result.Has("randomValue"));

        await testFlow.Stop();
    }

    public void Dispose()
    {
        this.flow?.Dispose();
    }
}
```

>> Best Practices

Best Practices

Tips for writing effective module tests.

1. Configure Appropriate Timeouts

Adjust timeouts based on your module's expected processing time:

```
var module = new RandomNumberModule();

// For fast modules (default)
var flow = module.SetupFlow(timeout: 5000);

// For slower modules
var flow = module.SetupFlow(timeout: 30000);

// For debugging (very long timeout)
var flow = module.SetupFlow(timeout: 300000);
```

> TIP
>
> Use longer timeouts during debugging to avoid timeout exceptions while stepping through code.

2. Test Error Conditions

Don't just test the happy path. Verify your module handles errors gracefully:

```
[Fact]
public async Task TestInvalidSettings()
{
    var module = new RandomNumberModule();
    module.Settings.TargetMin = 100;
    module.Settings.TargetMax = 10; // Invalid: min > max
    module.Settings.TargetName = "value";

    var flow = module.SetupFlow();

    // Validation should fail during Start
    var startResult = await flow.Start();
    Assert.False(startResult.IsSuccess);
```

```
    await flow.Stop();
}
```

3. Test Module Lifecycle

Verify your module handles initialization and shutdown correctly:

```
[Fact]
public async Task TestModuleLifecycle()
{
    var module = new RandomNumberModule();
    module.Settings.TargetMin = 0;
    module.Settings.TargetMax = 100;
    module.Settings.TargetName = "value";

    var flow = module.SetupFlow();

    // Test successful start
    var startResult = await flow.Start();
    Assert.True(startResult.IsSuccess);

    // Test processing
    await flow.Receive(new FlowMessage());
    var result = await flow.GetNextResult();
    Assert.NotNull(result);
    Assert.True(result.Has("value"));

    // Test successful stop
    var stopResult = await flow.Stop();
    Assert.True(stopResult.IsSuccess);
}
```

4. Use Meaningful Test Data

Create realistic test data that represents actual use cases:

```
[Fact]
public async Task TestRandomNumberDistribution()
{
    var module = new RandomNumberModule();
    module.Settings.TargetMin = 0;
    module.Settings.TargetMax = 10;
    module.Settings.TargetName = "value";
```

```
    var flow = module.SetupFlow();

    await flow.Start();

    // Generate multiple random numbers
    var message = new FlowMessage();
    for (int i = 0; i < 100; i++)
    {
        await flow.Receive(message.Clone());
    }

    var results = await flow.GetNextResults(100);

    // Verify all values are in range
    foreach (var result in results)
    {
        var value = result.Get<int>("value");
        Assert.InRange(value, 0, 9);
    }

    await flow.Stop();
}
```

5. Organize Tests with Test Classes

Use the IDisposable pattern for test cleanup:

```
public class RandomNumberModuleTests : IDisposable
{
    private readonly RandomNumberModule module;
    private readonly TestFlow flow;

    public RandomNumberModuleTests()
    {
        this.module = new RandomNumberModule();
        this.module.Settings.TargetMin = 0;
        this.module.Settings.TargetMax = 100;
        this.module.Settings.TargetName = "randomValue";
        this.flow = this.module.SetupFlow();
    }

    [Fact]
    public async Task GeneratesRandomNumber()
    {
        await this.flow.Start();
```

```
        await this.flow.Receive(new FlowMessage());
        var result = await this.flow.GetNextResult();

        Assert.True(result.Has("randomValue"));
        Assert.InRange(result.Get<int>("randomValue"), 0, 99);

        await this.flow.Stop();
    }

    public void Dispose()
    {
        this.flow?.Dispose();
    }
}
```

6. Test Edge Cases

Consider boundary conditions and edge cases:

```
[Theory]
[InlineData(int.MinValue)]
[InlineData(0)]
[InlineData(int.MaxValue)]
public async Task TestBoundaryValues(int value)
{
    var module = new MyModule();
    var flow = module.SetupFlow();

    await flow.Start();

    var message = new FlowMessage();
    message.Set("value", value);
    await flow.Receive(message);

    var result = await flow.GetNextResult();
    Assert.NotNull(result);

    await flow.Stop();
}
```

7. Verify Message Properties

Check that your module sets properties correctly:

```
[Fact]
public async Task VerifyOutputProperties()
{
    var module = new RandomNumberModule();
    module.Settings.TargetMin = 50;
    module.Settings.TargetMax = 150;
    module.Settings.TargetName = "sensor.temperature";

    var flow = module.SetupFlow();

    await flow.Start();

    await flow.Receive(new FlowMessage());
    var result = await flow.GetNextResult();

    // Verify the property was set with the configured name
    Assert.True(result.Has("sensor.temperature"));
    var temp = result.Get<int>("sensor.temperature");
    Assert.InRange(temp, 50, 149);

    await flow.Stop();
}
```

8. Test with Realistic Credentials

Use test credentials that match production structure:

```
[Fact]
public async Task TestWithRealisticCredentials()
{
    var module = new MyDatabaseModule();

    var credential = new DatabaseCredential
    {
        Id = Guid.NewGuid(),
        Host = "localhost",
        Port = 5432,
        Database = "testdb",
        Username = "testuser",
        Password = "testpass"
    };

    var flow = module.SetupFlow()
        .WithCredential(credential);
```

```
    await flow.Start();
    // Test database operations
    await flow.Stop();
}
```

9. Add Descriptive Test Names

Use clear, descriptive names that explain what the test verifies:

```
[Fact]
public async Task ProcessMessage_WithValidInput_ReturnsExpectedOutput() { }

[Fact]
public async Task ProcessMessage_WithMissingProperty_ReturnsError() { }

[Fact]
public async Task ProcessMessage_WithLargeDataset_CompletesWithinTimeout() { }
```

10. Document Complex Test Scenarios

Add comments for complex test logic:

```
[Fact]
public async Task TestComplexScenario()
{
    var module = new MyModule();
    var flow = module.SetupFlow();

    await flow.Start();

    // Simulate a sequence of related messages
    // that should trigger aggregation logic
    for (int i = 0; i < 5; i++)
    {
        var message = new FlowMessage();
        message.Set("batch.id", "batch-123");
        message.Set("batch.sequence", i);
        await flow.Receive(message);
    }

    // The module should aggregate after 5 messages
    var result = await flow.GetNextResult();
    Assert.Equal(5, result.Get<int>("batch.count"));
```

```
    await flow.Stop();
}
```

>> Troubleshooting

Troubleshooting

Common issues and solutions when testing modules.

Common Issues

1. Timeout Exceptions

Problem: `OperationCanceledException` when waiting for results

Symptoms:

```
System.OperationCanceledException: The operation was canceled.
```

Solutions:

- Increase timeout in `SetupFlow(timeout: 30000)`
- Check module processing - Verify your module is actually sending output
- Verify message routing - Ensure messages reach the correct output
- Check async operations - Ensure all async operations complete properly

Example Fix:

```
// Increase timeout for slow operations
var flow = module.SetupFlow(timeout: 30000);
```

2. Channel Closed Exceptions

Problem: `ChannelClosedException` when reading results

Symptoms:

```
System.Threading.Channels.ChannelClosedException: The channel has been closed.
```

Solutions:

- Call Start() first - Ensure you call `Start()` before sending messages
- Don't call Stop() early - Don't call `Stop()` before retrieving all expected results
- Check disposal order - Verify module isn't disposed prematurely

Example Fix:

```
[Fact]
public async Task CorrectOrder()
{
    var flow = module.SetupFlow();

    await flow.Start();           // 1. Start first
    await flow.Receive(message);  // 2. Send messages
    var result = await flow.GetNextResult();  // 3. Get results
    await flow.Stop();            // 4. Stop last
}
```

3. Missing Results

Problem: Expected results are not available

Symptoms:

- Test hangs waiting for results
- Fewer results than expected
- Null or empty results

Solutions:

- Verify output routing - Check if your module sends to the correct output index
- Check module initialization - Ensure module requires specific initialization
- Verify dependencies - Ensure all dependencies (credentials, resources) are properly set up
- Check message filters - Verify module isn't filtering out messages

Debugging Steps:

```
[Fact]
public async Task DebugMissingResults()
{
    var module = new MyModule();
    module.Settings.OutputEnabled = true; // Ensure output is enabled

    var flow = module.SetupFlow(timeout: 60000); // Long timeout for debugging

    var startResult = await flow.Start();
    Console.WriteLine($"Start result: {startResult.IsSuccess}");

    await flow.Receive(message);
    Console.WriteLine("Message sent");
```

```
    var result = await flow.GetNextResult();
    Console.WriteLine($"Result received: {result != null}");
}
```

4. Credential/Resource Loading Issues

Problem: Module can't access credentials or resources

Symptoms:

- Null reference exceptions when accessing credentials
- Resources not found
- Authentication failures

Solutions:

- Verify IDs match - Credential/resource IDs must match what your module expects
- Check file paths - Ensure file paths are correct for credential loading
- Verify JSON format - Ensure JSON serialization/deserialization works correctly
- Use correct types - Ensure credential types match module expectations

Example:

```
[Fact]
public async Task TestCredentialLoading()
{
    var module = new MyModule();

    // Ensure ID matches what module expects
    var credentialId = module.Settings.CredentialId;
    var credential = new MyCredential
    {
        Id = credentialId,
        ApiKey = "test-key"
    };

    var flow = module.SetupFlow()
        .WithCredential(credential);

    await flow.Start();
    // Now credential is accessible
}
```

5. Null Reference Exceptions

Problem: Null reference when accessing message properties

Symptoms:

```
System.NullReferenceException: Object reference not set to an instance of an object.
```

Solutions:

- Check property existence - Use `Has()` before `Get()`
- Set required properties - Ensure test messages contain all required properties
- Handle null values - Use `GetOrDefault()` for optional properties

Example:

```
[Fact]
public async Task SafePropertyAccess()
{
    var flow = module.SetupFlow();
    await flow.Start();

    await flow.Receive(new FlowMessage());
    var result = await flow.GetNextResult();

    // Safe property access
    if (result.Has("data.value"))
    {
        var value = result.Get<string>("data.value");
        Assert.NotNull(value);
    }
}
```

Debugging Tips

1. Enable Debug Output

Add logging to see what's happening:

```
[Fact]
public async Task DebugTest()
{
    var module = new MyModule();
    var flow = module.SetupFlow();

    await flow.Start();
```

```
    var message = new FlowMessage();
    message.Set("data", "test");
    Console.WriteLine($"Sending message: {message.ToJSON()}");

    await flow.Receive(message);

    var result = await flow.GetNextResult();
    Console.WriteLine($"Received result: {result.ToJSON()}");

    await flow.Stop();
}
```

2. Check Module State

Verify module configuration before testing:

```
[Fact]
public async Task VerifyModuleState()
{
    var module = new MyModule();
    var flow = module.SetupFlow();

    // Verify settings
    Assert.NotNull(module.Settings);
    Assert.Equal("expected-value", module.Settings.Property);

    // Verify credentials
    Assert.True(module.Settings.Credentials.Any());

    await flow.Start();
    // Continue test
}
```

3. Test Individual Components

Test components separately to isolate issues:

```
[Fact]
public void TestCredentialLoading()
{
    // Test credential loading separately
    var json = File.ReadAllText("test-cred.json");
    var credential = JsonSerializer.Deserialize<MyCredential>(json);
```

```
    Assert.NotNull(credential);
    Assert.NotNull(credential.ApiKey);
}

[Fact]
public async Task TestModuleProcessing()
{
    // Test module in isolation
    var module = new MyModule();
    // ...
}
```

4. Use Breakpoints

Set breakpoints in your module code and step through execution:

```
[Fact]
public async Task DebugWithBreakpoints()
{
    var module = new MyModule();
    var flow = module.SetupFlow(timeout: 300000); // Long timeout for debugging

    await flow.Start();

    // Set breakpoint here, then step into your module
    await flow.Receive(message);

    var result = await flow.GetNextResult();
}
```

5. Verify Async Operations

Ensure all async operations complete:

```
[Fact]
public async Task VerifyAsyncCompletion()
{
    var module = new MyModule();
    var flow = module.SetupFlow();

    var startTask = flow.Start();
    Assert.False(startTask.IsCompleted); // Should be running

    var startResult = await startTask;
    Assert.True(startTask.IsCompleted); // Should be complete
```

```
    Assert.True(startResult.IsSuccess);
}
```

Getting Help

If you continue to experience issues:

1. Check the SDK version compatibility
2. Review the module implementation for common mistakes
3. Consult the Best Practices guide
4. Review similar tests in the SDK examples
5. Contact Crosser support with a minimal reproduction example

Guidelines

This tutorial provides guidelines about how to write modules.

>> FlowMessage

FlowMessage

The FlowMessage is the primary data structure for passing information between modules in a flow. It's a dynamic object that allows you to set and retrieve properties at runtime without needing to define a fixed schema.

Key Concepts

- Dynamic Structure: FlowMessages can hold any property structure, making them flexible for various data scenarios
- Type Safety: While dynamic, FlowMessage provides type-safe methods for getting and setting values
- Immutable Properties: Internally uses immutable dictionaries for thread-safe operations
- JSON Serialization: Seamlessly converts to and from JSON

A FlowMessage is what you receive in the MessageReceived method on your module, and what you pass when calling the Next method.

Creating FlowMessages

There are several ways to create and populate a FlowMessage.

Using Index Operator

You can use the index operator to set properties:

```
var message = new FlowMessage();
message["name"] = "Steven";
message["age"] = 42;
message["isActive"] = true;
```

Using Type-Safe Set Method

For better type safety and control, use the `Set<T>` method:

```
var message = new FlowMessage();
message.Set("name", "Steven");
message.Set("age", 42);
message.Set("isActive", true);
```

Using dynamic Syntax

You can also create a FlowMessage as a dynamic and add properties using dot notation:

```
dynamic message = new FlowMessage();
message.name = "Steven";
message.age = 42;
message.isActive = true;
```

Working with Nested Properties

FlowMessage supports hierarchical data structures using dot notation. This allows you to work with complex, nested objects efficiently.

Reading Nested Properties

When you receive a message with nested structure like this:

```
{
    "data": {
        "sensor": {
            "id": "abc123",
            "celsius": 34.5
        }
    }
}
```

You can access nested values directly using dot notation:

```
// Using Get<T> method (recommended)
var celsius = message.Get<double>("data.sensor.celsius");
// Using GetValue<T> method (alternative)
var sensorId = message.GetValue<string>("data.sensor.id");
// Using index operator (less type-safe)
var id = (string) message["data.sensor.id"];
```

Writing Nested Properties

You can set nested properties, and FlowMessage will automatically create the intermediate objects:

```
// Set a new property in the nested structure
message.Set("data.sensor.fahrenheit", 94.1);
```

After these operations, your message structure will be:

```
{
    "data": {
        "sensor": {
            "id": "abc123",
            "celsius": 34.5,
            "fahrenheit": 94.1        }
    }
}
```

Working with Arrays

FlowMessage also supports array indexing in paths:

```
// Set array elements
message.Set("sensors[0].temperature", 23.5);
message.Set("sensors[1].temperature", 24.2);

// Get array elements
var temp = message.Get<double>("sensors[0].temperature");
```

Checking for Properties

Before accessing properties, it's often useful to check if they exist. FlowMessage provides the Has method for this purpose.

Basic Property Check

Check if a property exists at any level:

```
// Check if a simple property exists
if (message.Has("age"))
{
    Console.WriteLine("The message has an 'age' property");
}

// Check nested properties
if (message.Has("data.sensor.temperature"))
{
    var temp = message.Get<double>("data.sensor.temperature");
}
```

Type-Safe Property Check

You can also verify that a property exists and has a specific type using Has<T>:

```
// Check if 'age' exists and is an integer
if (message.Has<int>("age"))
{
    var age = message.Get<int>("age");
    Console.WriteLine($"Age is: {age}");
}

// Check if a nested property exists with the correct type
if (message.Has<string>("data.sensor.id"))
{
    var sensorId = message.Get<string>("data.sensor.id");
}
```

> TIP
>
> Using `Has<T>` is safer than `Has` alone because it ensures the property can be converted to the expected type before you attempt to retrieve it.

Removing Properties

You can remove properties from a FlowMessage at any level:

```
// Remove a top-level property
message.Remove("age");

// Remove a nested property
message.Remove("data.sensor.humidity");

// The Remove method returns the FlowMessage for method chaining
message.Remove("prop1").Remove("prop2").Set("prop3", "value");
```

Cloning FlowMessages

When you need an independent copy of a FlowMessage, use the `Clone` method:

```
var original = new FlowMessage();
original.Set("name", "John");
original.Set("age", 30);

// Create a deep copy
```

```
var copy = original.Clone();

// Modifying the copy doesn't affect the original
copy.Set("age", 31);

Console.WriteLine(original.Get<int>("age")); // Still 30
Console.WriteLine(copy.Get<int>("age"));     // Now 31
```

Working with JSON

Parsing JSON into FlowMessage

```
// Parse JSON string
var json = "{\"name\":\"Alice\",\"age\":25}";
var message = FlowMessage.Parse(json);

// Safe parsing with TryParse
if (FlowMessage.TryParse(json, out IFlowMessage result))
{
    Console.WriteLine("Successfully parsed");
}

// Parse JSON arrays or complex structures
var arrayJson = "[{\"id\":1},{\"id\":2}]";
var message2 = FlowMessage.Parse("items", arrayJson);
```

Converting FlowMessage to JSON

```
var message = new FlowMessage();
message.Set("name", "Bob");
message.Set("age", 35);

// Convert to JSON string
string json = message.ToJSON();
// or simply
string json2 = message.ToString();
```

Converting Objects to FlowMessage

You can convert any object to a FlowMessage:

```
public class Person
{
```

```
    public string Name { get; set; }
    public int Age { get; set; }
}

var person = new Person { Name = "Charlie", Age = 40 };

// Convert to FlowMessage with extenstion method
var message = FlowMessageExtensions.ToFlowMessage(person);
// Or just assigning the person object to a property
message.Set("person", person);
// Now you can access properties dynamically
var name = message.Get<string>("Name");
var age = message.Get<int>("Age");
var personFromMessage = message.Get<Person>("person");
```

Using Templates

FlowMessage supports template syntax for dynamic value substitution:

```
var message = new FlowMessage();
message.Set("firstName", "John");
message.Set("lastName", "Doe");

// Use template to combine values
var result = message.GetByTemplate("Hello {firstName} {lastName}!");
// Result: "Hello John Doe!"

// Templates work with nested properties too
message.Set("user.name", "Alice");
var greeting = message.GetByTemplate("Welcome {user.name}");
```

Error Handling

FlowMessage provides built-in properties for tracking success/error states:

```
// Mark a message as successful
message.SetSuccess();

// Mark a message as failed with an error message
message.SetError("Connection timeout");

// Check the status
if (message.IsError)
```

```
{
    Console.WriteLine($"Error: {message.ErrorMessage}");
}

if (message.IsSuccess)
{
    Console.WriteLine("Operation completed successfully");
}
```

Best Practices

1. Use type-safe methods: Prefer `Get<T>()` and `Set<T>()` over dynamic access for better type safety
2. Check before accessing: Always use `Has()` or `Has<T>()` before retrieving values to avoid exceptions
3. Clone when needed: If you need to modify a message while preserving the original, use `Clone()`
4. Use templates wisely: Templates are powerful but add processing overhead—use them when dynamic substitution is necessary
5. Handle nested structures: Take advantage of dot notation for cleaner code when working with complex objects

Common Patterns

Safely Getting a Value with Default

```
// Get a value or return a default if it doesn't exist
var age = message.Has<int>("age")
    ? message.Get<int>("age")
    : 0;
```

Transforming Data Between Modules

```
protected override void MessageReceived(IFlowMessage message)
{
    // Read input
    if (message.Has<double>("celsius"))
    {
        var celsius = message.Get<double>("celsius");

        // Transform
        var fahrenheit = (celsius * 9 / 5) + 32;

        // Write output
        message.Set("fahrenheit", fahrenheit);

        // Pass to next module
```

```
        this.Next(message);
    }
}
```

Validating Required Properties

```
private bool ValidateMessage(IFlowMessage message)
{
    var requiredProperties = new[] { "id", "timestamp", "value" };

    foreach (var prop in requiredProperties)
    {
        if (!message.Has(prop))
        {
            message.SetError($"Missing required property: {prop}");
            return false;
        }
    }

    return true;
}
```

Next Steps

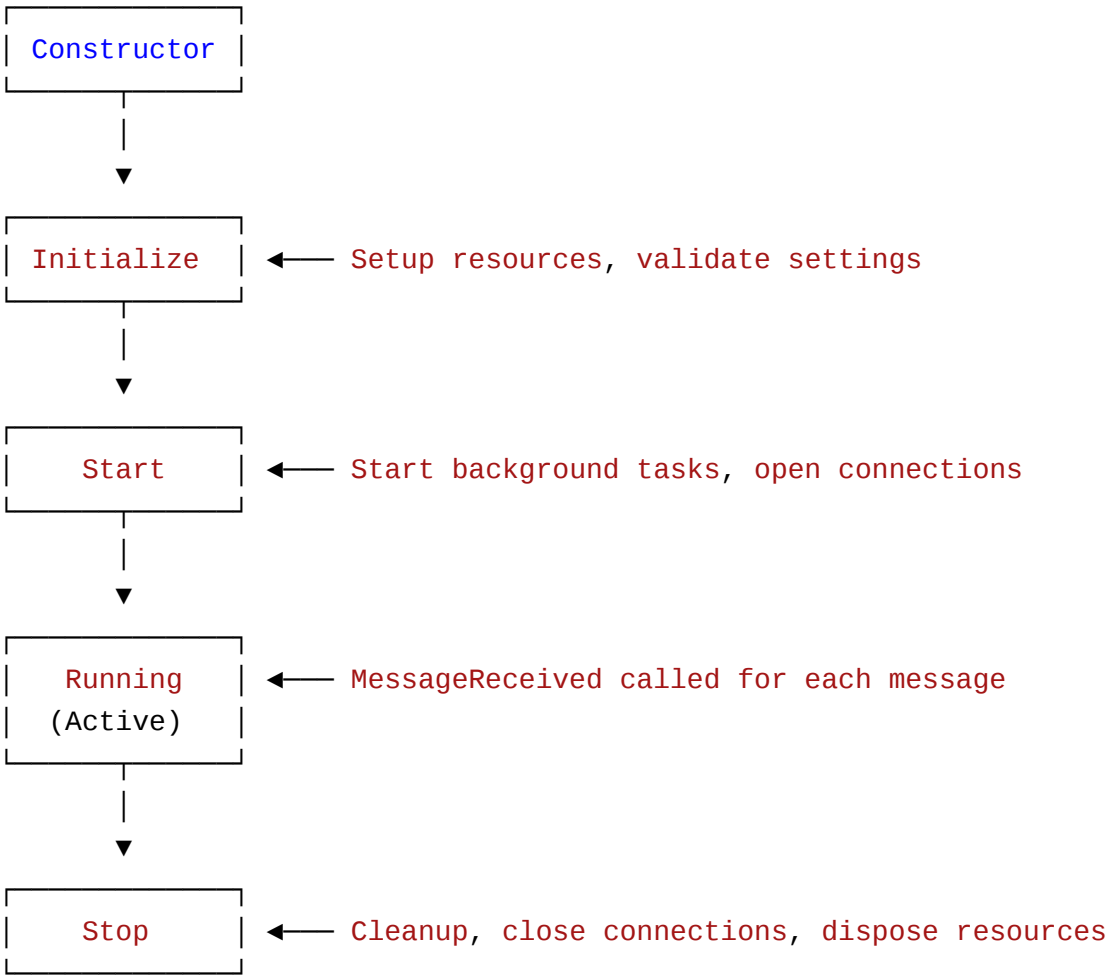
[>> Module Lifetime]

Module Lifetime

The FlowModule base class provides three lifecycle methods: `Initialize`, `Start`, and `Stop`. Override these to manage resources and connections.

Lifecycle Sequence

1. Constructor - Module instance is created
2. Initialize - Called once per module to set up resources
3. Start - Called after all modules are initialized, signals the flow is ready
4. MessageReceived - Called repeatedly as messages flow through the module
5. Stop - Called when the flow is stopping, used for cleanup



Initialize

Initialize is called once when the flow starts. Use it to validate settings and create resources.

Use for: Validate settings, create connection pools, initialize caches, check prerequisites

Return: `await base.Initialize()` on success, or an IError to prevent flow startup

```
public override async Task<IError?> Initialize()
{
    if (string.IsNullOrEmpty(this.Settings.ConnectionString))
        return Error.Create("Connection string is required");

    try
    {
        this.connectionPool = new ConnectionPool(this.Settings.ConnectionString);
    }
    catch (Exception ex)
    {
        return Error.Create($"Failed to initialize: {ex.Message}");
    }

    return await base.Initialize();
}
```

> WARNING
>
> Don't start background tasks here. Use `Start` instead.

Start

Start is called after all modules are initialized. Use it to begin active operations.

Use for: Start timers, open connections, begin listening for events, activate input sources

Return: `await base.Start()` on success, or an IError to stop the flow

```
public override async Task<IError?> Start()
{
    this.cancellationSource = new CancellationTokenSource();
    this.timer = new Timer(_ => PollDataSource(), null,
        TimeSpan.Zero, TimeSpan.FromSeconds(this.Settings.Interval));

    try
    {
        this.mqttClient.Connect();
        this.mqttClient.OnMessageReceived += HandleIncomingMessage;
    }
    catch (Exception ex)
```

```
    {
        return Error.Create($"Failed to connect: {ex.Message}");
    }

    return await base.Start();
}
```

Stop

Stop is called when the flow stops. Use it to cleanup and release resources.

Use for: Stop timers, close connections, dispose resources, cancel operations

```
public override void Stop()
{
    this.cancellationSource?.Cancel();
    this.timer?.Dispose();
    this.timer = null;

    try
    {
        this.mqttClient?.Disconnect();
        this.mqttClient?.Dispose();
    }
    catch (Exception ex)
    {
        this.LogWarning(ex, "Error disconnecting");
    }

    this.cancellationSource?.Dispose();
    this.cancellationSource = null;
}
```

> ⊗ IMPORTANT
>
> Always implement `Stop` if you override `Start` or `Initialize`. Handle exceptions gracefully to avoid blocking other modules' cleanup.

Best Practices

Constructor vs Initialize

- ❌ Don't create resources in constructor (settings not available)

- ✅ Do create resources in Initialize (settings available, can validate)

Resource Management

- Initialize: Create resources (HttpClient, connection pools)
- Start: Activate resources (start timers, open connections)
- Stop: Cleanup in reverse order (cancel, dispose, null)

Error Handling

- Return IError from Initialize/Start to prevent flow startup
- Log and swallow exceptions in Stop to allow other modules to cleanup

Examples

Input Module (Timer-based)

```
public class MyInputModule : InputModule<MySettings>
{
    public override async Task<IError?> Start()
    {
        this.timer = new Timer(_ => GenerateData(), null,
            TimeSpan.Zero, TimeSpan.FromSeconds(this.Settings.Interval));
        return await base.Start();
    }

    private void GenerateData()
    {
        var message = new FlowMessage();
        message.Set("value", GetSensorReading());
        this.Next(message);
    }

    public override void Stop()
    {
        this.timer?.Dispose();
        this.timer = null;
    }
}
```

Output Module (HTTP)

```
public class MyOutputModule : OutputModule<MySettings>
{
    public override async Task<IError?> Initialize()
```

```
    {
        this.httpClient = new HttpClient { BaseAddress = new
Uri(this.Settings.ApiEndpoint) };
        return await base.Initialize();
    }

    protected override void MessageReceived(IFlowMessage message)
    {
        try
        {
            var response = this.httpClient.PostAsync("/data",
                new StringContent(message.ToJSON(), Encoding.UTF8,
"application/json"))
                .GetAwaiter().GetResult();

            message.SetSuccess(response.IsSuccessStatusCode);
        }
        catch (Exception ex)
        {
            message.SetError(ex.Message);
        }
        this.Next(message);
    }

    public override void Stop()
    {
        this.httpClient?.Dispose();
        this.httpClient = null;
    }
}
```

Troubleshooting

Issue	Solution
Module doesn't start	Check Initialize/Start return IError, review logs, verify settings
Resources not cleaned up	Implement Stop, dispose all IDisposable, cancel tokens
Flow startup slow	Move long operations to Start, use async/await, lazy initialization

Next Steps

>> Module Settings

Module Settings

Module settings define configurable properties that users can adjust in the Flow Studio UI. Settings are created by inheriting from FlowModuleSettings.

> TIP
>
> Required namespaces:
>
> - `Crosser.EdgeNode.Common.Abstractions.Utilities.Validation`
> - `Crosser.EdgeNode.Flows.Abstractions`
> - `System.ComponentModel`
> - `System.ComponentModel.DataAnnotations`

Creating Settings Class

Name your settings class with a `Settings` suffix. For example: `RandomNumberModule` → `RandomNumberModuleSettings`

```
public class RandomNumberModuleSettings : FlowModuleSettings
{
    [Display(Name = "Minimum value", Description = "The smallest value allowed")]
    [JsonSchemaExtensionData("x-sortOrder", 1)]
    [DefaultValue(0)]

    public int MinValue { get; set; }

    [Display(Name = "Maximum value", Description = "The highest value allowed")]
    [JsonSchemaExtensionData("x-sortOrder", 2)]
    [DefaultValue(100)]
    public int MaxValue { get; set; }
}
```

In this example, we define two integer settings: `MinValue` and `MaxValue`, each with display names, descriptions, sort order, and default values. The attributes control how the settings appear in the Flow Studio UI. The sort order ensures that `MinValue` appears before `MaxValue` in the Flow Studio UI. The display names and descriptions help users understand the purpose of each setting. The default values provide initial values when the module is added to a flow.

Property Guidelines

Supported Types

- Primitive types: `int`, `string`, `bool`, `double`, `DateTime`, `Guid`
- Enums: For predefined choices
- Collections: `List<T>` for multiple values
- Nested objects: Group related settings into classes

Configuration Attributes

Attributes control how settings appear and behave in Flow Studio:

- Display - Sets labels and tooltips
- Validation - Adds input restrictions
- Default Values - Sets initial values
- Data Types - Specifies special UI behaviors
- Credentials - Marks a setting as a credential.
- Resources - Marks a setting as a resource.

Category	Attribute	Purpose	Example

Display	`[Display]`	Sets label for field	`[Display(Name = "Connection String")]`
			`public string ConnectionStri`
			`get; set; }`

Display	`[Description]`	Provides help text and tooltips for fields	`[Description("Enter the data connection string")]`
			`public string ConnectionStri`
			`get; set; }`

Validation	`[Required]`	Makes a field mandatory	`[Required]`
			`[Display(Name = "API Key")]`
			`public string ApiKey { get;`

Validation	`[Range]`	Sets numeric min/max constraints	`[Range(1, 1000)]`
			`[Display(Name = "Batch Size"`
			`public int BatchSize { get;`
			`= 100;`

Validation	`[MinLength]`	Sets minimum string length	`[MinLength(1)]`
			`[Display(Name = "Queue Name"`
			`public string QueueName { ge`
			`set; }`

Category	Attribute	Purpose	Example
Validation	`[MaxLength]`	Sets maximum string length	`[MaxLength(256)]` `[Display(Name = "Description` `public string Description {` `set; }`
Validation	`[RegularExpression]`	Validates against regex pattern	`[RegularExpression(@"^[a-zA-` `9_-]+$")]` `[Display(Name = "Identifier"` `public string Identifier { g` `set; }`
Default Values	`[DefaultValue]`	Sets default value for a property	`[DefaultValue(true)]` `[Display(Name = "Enable Cach` `public bool EnableCaching {` `set; } = true;`
Data Types	`[DataType]`	Specifies special data types (e.g., multiline text)	`[DataType(DataType.Multiline` `[Display(Name = "SQL Query")` `public string SqlQuery { get` `}`
Credentials	`[JsonSchemaExtensionData]` `("x-credential", type)`	Links to credential storage Types: `"UsernamePassword"`, `"ApiKey"`, `"Certificate"`, `"ConnectionString"`	`[JsonSchemaExtensionData("x-` `credential", "UsernamePasswo` `[Display(Name = "Database` `Credentials")]` `public Guid? DatabaseCredent` `get; set; }`
Resources	`[JsonSchemaExtensionData]` `("x-resource", "File")`	Links to resource files Note: Always use `"File"` for custom modules	`[JsonSchemaExtensionData("x-` `resource", "File")]` `[Display(Name = "Configurati` `File")]` `public Guid? ConfigurationFi` `get; set; }`
Sort Order	`[JsonSchemaExtensionData]` `("x-sortOrder", order)`	Sets the display order in the UI	`[JsonSchemaExtensionData("x-` `sortOrder", 1)]` `[Display(Name = "Min Value")` `public int MinValue { get; s`

TIP

Key Points:

- Combine multiple attributes on a single property for comprehensive configuration
- Never store passwords directly; always use `JsonSchemaExtensionData` for credentials
- Always set both `[DefaultValue]` attribute and property initializer for consistency

Example

```
[Display(Name = "Connection timeout", Description = "Timeout in seconds")]
[DefaultValue(30)]
[Range(1, 300)]
public int TimeoutSeconds { get; set; } = 30;

[Display(Name = "SQL Query", Description = "Query to execute")]
[DataType(DataType.MultilineText)]
[NotNull]
[DefaultValue("Select * from table")]
public string Query { get; set; } = "Select * from table";

[Display(Name = "Allowed IPs", Description = "List of allowed IP addresses")]
public List<string> AllowedIps { get; set; } = new();
```

Validation

Override `Validate` to add custom validation logic:

```
public class RandomNumberModuleSettings : FlowModuleSettings
{
    [Display(Name = "Minimum value")]
    [DefaultValue(0)]
    public int MinValue { get; set; }

    [Display(Name = "Maximum value")]
    [DefaultValue(100)]
    public int MaxValue { get; set; }

    public override void Validate(SettingsValidator validator)
    {
        // Built-in validators
        validator.Validate(nameof(this.MinValue), this.MinValue)
            .MinValue(1)
```

```
            .MaxValue(1000);

        validator.Validate(nameof(this.MaxValue), this.MaxValue)
            .MinValue(1)
            .MaxValue(1000);

        // Custom validation logic
        if (this.MinValue >= this.MaxValue)
        {
            validator.AddError(nameof(this.MinValue),
                "Minimum value must be less than maximum value");
        }
    }
}
```

Available Validators

- `MinValue(int)` / `MaxValue(int)` - Range validation for numbers
- `MinLength(int)` / `MaxLength(int)` - Length validation for strings
- `Required()` - Ensures value is not null/empty
- `AddError(propertyName, message)` - Custom validation errors

Common Patterns

Enums

```
[JsonConverter(typeof(StringEnumConverter))]
public enum ConnectionMode
{
    [Description("Automatic")]
    Auto,
    [Description("Always use SSL")]
    Ssl
}

[Display(Name = "Connection mode")]
[DefaultValue(ConnectionMode.Auto)]
public ConnectionMode Mode { get; set; }
```

Collections

```
[Display(Name = "Allowed IPs", Description = "IP addresses allowed to connect")]
public List<string> AllowedIps { get; set; } = new();
```

```
[Display(Name = "Field Mappings")]
public List<FieldMapping> Mappings { get; set; } = new();

public class FieldMapping
{
    [Display(Name = "Source")]
    public string Source { get; set; }

    [Display(Name = "Target")]
    public string Target { get; set; }
}
```

Nested Settings

```
public class MyModuleSettings : FlowModuleSettings
{
    [Display(Name = "Connection Settings")]
    public ConnectionSettings Connection { get; set; } = new();
}

public class ConnectionSettings
{
    [Display(Name = "Host")]
    [Required]
    public string Host { get; set; }

    [Display(Name = "Port")]
    [DefaultValue(443)]
    [Range(1, 65535)]
    public int Port { get; set; } = 443;
}
```

Conditional Validation

```
[Display(Name = "Enable authentication")]
[DefaultValue(false)]
public bool UseAuth { get; set; }

[Display(Name = "Username")]
public string Username { get; set; }

public override void Validate(SettingsValidator validator)
{
    if (this.UseAuth && string.IsNullOrEmpty(this.Username))
```

```
    {
        validator.AddError(nameof(this.Username),
            "Username required when authentication enabled");
    }
}
```

Accessing Settings

Access settings in your module via the `Settings` property:

```
public class MyModule : FlowModule<MyModuleSettings>
{
    protected override async Task MessageReceived(IFlowMessage message)
    {
        // Direct access
        var timeout = this.Settings.TimeoutSeconds;

        // Nested settings
        var host = this.Settings.Connection.Host;
        var port = this.Settings.Connection.Port;

        // Enum settings
        if (this.Settings.Mode == ProcessingMode.Batch)
        {
            await ProcessBatch(message);
        }

        // Collections
        foreach (var mapping in this.Settings.Mappings)
        {
            var value = message.Get<object>(mapping.Source);
            message.Set(mapping.Target, value);
        }

        await this.Next(message);
    }
}
```

Common Settings (Platform-Managed)

The `CommonSettings` property provides platform-managed configuration that you can read but should NOT modify:

- `MaxItemsInQueue` - Queue size limit

- `ChannelFullMode` - Behavior when queue is full
- `MessagesRetries` - Number of retry attempts
- `MessagesRetryDelay` - Delay between retries
- `PersistentMessages` - Whether messages are persisted
- `DisableQueues` - Queue disable flag

Advanced Features

Credentials

```
[JsonSchemaExtensionData("x-credential", "UsernamePassword")]
[Display(Name = "Database Credentials")]
public Guid? DatabaseCredential { get; set; }

// Other types: "ApiKey", "Certificate", "ConnectionString"
```

Resources

```
[JsonSchemaExtensionData("x-resource", "File")]
[Display(Name = "Configuration File")]
public Guid? ConfigFile { get; set; }
```

> NOTE
>
> See Crosser documentation for how to create credentials and resources in flow studio.

Summary

- Provide defaults: Always set `[DefaultValue]` when possible
- Clear naming: Use descriptive names that users will understand
- Add descriptions: Help users understand what each setting does
- Validate early: Catch configuration errors before the flow starts
- Logical order: Arrange properties in the order users should configure them

Next Steps

>> Module Status

Module Status

Module status alerts users about events within your module. Use SetStatus to communicate warnings or informational messages without stopping the flow.

Available Status Levels

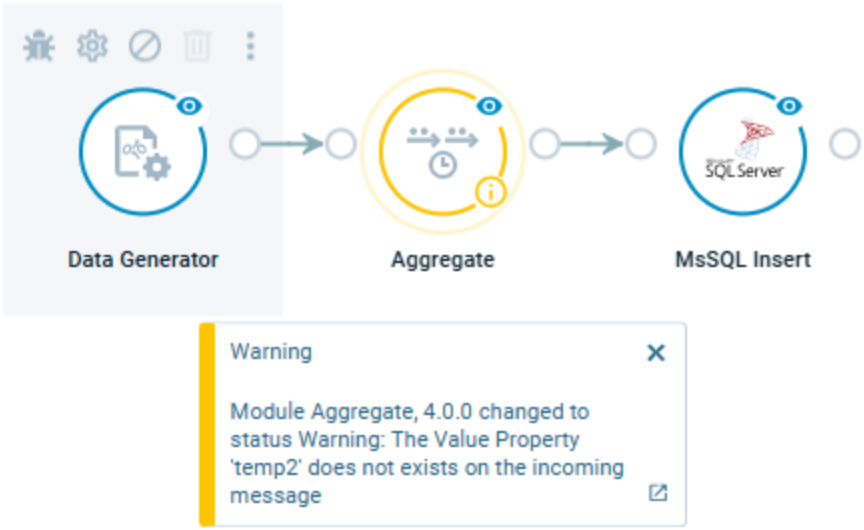
- `Unknown` — Status unknown.
- `Ok` — Successful operation.
- `Warning` — Non-critical issues that need attention; flow can continue.
- `Error` — Critical problems that prevent normal operation; may require handling or escalation and stops the flow if halt on error is enabled.
- `Fatal` — Severe failures that require immediate intervention and typically stop the flow.

Example

If your module expects a property that's missing, set a warning instead of failing:

```
if (!message.Has(this.Settings.SourceName))
{
    this.SetStatus(Status.Warning,
        $"Property '{this.Settings.SourceName}' not found on message");
    // Continue processing with default behavior
    return;
}
```

The status appears in Flow Studio:



Common Patterns

Missing Property

```
if (!message.Has("field"))
{
    this.SetStatus(Status.Information, "Incoming message missing 'field' property");
}
```

Partial Success

```
var processed = 0;
var failed = 0;

foreach (var item in items)
{
    if (ProcessItem(item))
        processed++;
    else
        failed++;
}

if (failed > 0)
{
    this.SetStatus(Status.Warning,
        $"Processed {processed}, failed {failed}");
}
```

Important Notes

> TIP
>
> Status Update Behavior:
>
> - `SetStatus` only fires if the status changes from the current status
> - Use `SetStatus(status, message, force: true)` to always fire the status
> - Clear status with `SetStatus(Status.OK)`

NOTE

Status messages appear in Flow Studio and help users debug flows. Use clear, actionable messages.

Next Steps

>> Error Handling

Error Handling

Since your module will receive a FlowMessage and this is a dynamic object you can´t be sure that the person building the flow will send in the data your module expects. Design modules to handle unexpected or missing data gracefully (use safe getters and ignore unexpected data).

General Principles

- Be fault-tolerant: Don't crash on invalid data
- Validate inputs: Check for required properties before processing
- Use SetStatus: Alert users about issues without stopping the flow (see Module Status)
- Throw only for critical errors: Use FlowModuleException for unrecoverable errors

Retry Strategies

There are two approaches for handling transient errors:

1. Platform-managed retries - Use platform retry settings (configured by the user)
2. Module-level retries - Implement custom retry logic within your module

The recommended approach is to use platform-managed retries when possible since retries will affect other modules in the flow. Sometimes though you might need to implement custom retry logic within your module, for example when running input modules that read from external services or having timer based logic in the module.

Platform-Managed Retries

The platform provides retry settings that users can configure. When using platform retries, always forward messages and let the platform handle retries:

```
protected override async Task MessageReceived(IFlowMessage message)
{
    try
    {
        // Attempt processing
        await this.ProcessMessage(message);
    }
    catch (HttpRequestException ex) when (this.IsTransientError(ex))
    {
        // Log transient error but don't retry here
        Warning(ex, "Transient HTTP error occurred");
        var error = $"Transient error: {ex.Message}";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
        // Platform will retry based on user's retry settings
```

```
    }
    catch (Exception ex)
    {
        // Log permanent errors
        Error(ex, "Processing failed");
        var error = $"Processing failed: {ex.Message}";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    finally
    {
        // Always forward the message - platform handles retries
        await this.Next(message);
    }
}

private bool IsTransientError(Exception ex)
{
    return ex is HttpRequestException httpEx &&
           (httpEx.Message.Contains("timeout", StringComparison.OrdinalIgnoreCase)
||
            httpEx.Message.Contains("connection",
StringComparison.OrdinalIgnoreCase));
}
```

> **TIP**
>
> For transient failures (network, service unavailable), retry logic is implemented for all modules by the SDK. It can be configured in common settings in the UI when the flow is designed. See Messages Queues and Retries for more information.

Module-Level Retries

In general. the platform managed retries should be preferred. However, there are scenarios where you might need custom retry logic within your module. An example is when you have timer-based input modules that read from external services.

```
public class MyInputModule : InputModule<MySettings>
{
    public override async Task<IError?> Start()
    {
        this.timer = new Timer(async _ => await GenerateData(), null,
```

```
            TimeSpan.Zero, TimeSpan.FromSeconds(this.Settings.Interval));
        return await base.Start();
    }

    private async Task GenerateData()
    {
        var message = new FlowMessage();
        int maxRetries = 5;
        int delayMs = 500;
        for (int attempt = 1; attempt <= maxRetries; attempt++)
        {
            try
            {
                var value = await GetSensorReading();
                message.Set("value", value);
                break;
            }
            catch (Exception ex)
            {
                if (attempt == maxRetries)
                {
                    message.SetError($"Sensor read failed after {maxRetries}
attempts: {ex.Message}");
                }
                else
                {
                    await Task.Delay(delayMs);
                    delayMs *= 2; // Backoff
                }
            }
        }

        await this.Next(message);
    }

    public override void Stop()
    {
        this.timer?.Dispose();
        this.timer = null;
    }
}
```

Output Modules Need Special Care

Since output modules represent some kind of communication with the outside world things can go wrong. This means that you need to think about always sending a result

out from your output modules.

Example

Let´s say that you create a module that sends a message to one of the services on Azure. At some point the service might be down or your device might have a failing internet connection. To overcome this possible issue you need to make sure that you always pass a message to the next module and that this message contains the result of the modules operation.

```csharp
protected override async Task MessageReceived(IFlowMessage message)
{
    try
    {
        // Validate input
        if (!message.Has<string>("deviceId"))
        {
            // Handle missing deviceId gracefully
            this.SetStatus(Status.Warning, "Missing deviceId, skipping message");
            return;
        }
        // Perform logic operation
        var isSuccess = await SendToExternalService(message);

        if (isSuccess)
        {
            // If the operation was a success
            message.SetSuccess();
        }
        else
        {
            // If the operation failed
            message.SetError("Failed to send message");
        }
    }
    catch (Exception ex)
    {
        // Set error on exception
        message.SetError(ex.Message);
    }
    finally
    {
        // Always forward the message
        await this.Next(message);
```

```
    }
}
```

[TIP]

> message.SetSuccess() can be omitted. The runtime will set success automatically if no error is set.

By using the logic above your module will always output the result. The methods SetSuccess and SetError takes care of writing a crosser object to the message. For example, if you call message.SetSuccess() your message will look like:

```
{
    "crosser": {
        "success": true
    }
}
```

If you call message.SetError("Failed to connect to API"), your message will look like:

```
{
    "crosser": {
        "success": false,
        "message": "Failed to connect to API"
    }
}
```

> **TIP**
>
> For transient failures (network, service unavailable), retry logic is implemented for all modules by the SDK. It can be configured in common settings in the UI when the flow is designed. See Messages Queues and Retries for more information.

Related Topics

- Security Best Practices - Secure error handling principles
- Performance Optimization - Efficient error handling patterns
- External Integrations - Error handling for external services

>> Module Status

Security Best Practices

Essential security guidelines for developing secure Crosser modules.

Credential Management

Never Store Sensitive Data Directly

❌ Never expose sensitive settings directly:

```
public class UnsafeSettings : FlowModuleSettings
{
    public string Password { get; set; }         // DON'T DO THIS
    public string ApiKey { get; set; }           // DON'T DO THIS
    public string ConnectionString { get; set; } // DON'T DO THIS
}
```

✅ Always use credential references:

```
public class SecureSettings : FlowModuleSettings
{
    [JsonSchemaExtensionData("x-credential", "UsernamePassword")]
    [Display(Name = "Service Credentials")]
    public Guid? ServiceCredential { get; set; }

    [JsonSchemaExtensionData("x-credential", "ApiKey")]
    [Display(Name = "API Key")]
    public Guid? ApiCredential { get; set; }

    [JsonSchemaExtensionData("x-credential", "ConnectionString")]
    [Display(Name = "Database Connection")]
    public Guid? DatabaseCredential { get; set; }
}
```

Secure Credential Access

```
public override async Task<IError?> Initialize()
{
    if (!this.Settings.ApiCredential.HasValue)
    {
        return new Error("API credentials not configured");
    }

    // Retrieve credentials securely
```

```
    var credentials = await this.GetCredentialContentAsync<CredentialWithApiKey>(
        this.Settings.ApiCredential.Value);

    if (credentials.IsError || credentials.Value is null)
    {
        return credentials.Error ?? new Error("Failed to get API credential");
    }

    // Use credentials safely
    this.httpClient.DefaultRequestHeaders.Add("Authorization",
        $"Bearer {credentials.Value.ApiKey}");

    // ❌ NEVER log credentials
    // Information("Using API key: {ApiKey}", credentials.Value.ApiKey);

    // ✅ Log safely without secrets
    Information("API authenticated successfully");

    return await base.Initialize();
}
```

Input Validation and Sanitization

Validate All External Input

Always validate settings in the `Validate` method:

```
public override void Validate(SettingsValidator validator)
{
    // 1. Validate required fields
    if (string.IsNullOrWhiteSpace(this.ApiEndpoint))
    {
        validator.AddError(nameof(this.ApiEndpoint), "API endpoint is required");
        return;
    }

    // 2. Validate URL format and enforce HTTPS
    if (!Uri.TryCreate(this.ApiEndpoint, UriKind.Absolute, out var uri) ||
        !uri.Scheme.Equals("https", StringComparison.OrdinalIgnoreCase))
    {
        validator.AddError(nameof(this.ApiEndpoint),
            "API endpoint must be a valid HTTPS URL");
    }

    // 3. Validate port ranges
```

```
    if (this.Port < 1 || this.Port > 65535)
    {
        validator.AddError(nameof(this.Port),
            $"Port must be between 1 and 65535, got {this.Port}");
    }

    // 4. Validate input patterns
    if (!string.IsNullOrWhiteSpace(this.Identifier) &&
        !Regex.IsMatch(this.Identifier, @"^[a-zA-Z0-9_-]+$"))
    {
        validator.AddError(nameof(this.Identifier),
            "Identifier can only contain letters, numbers, underscores,
and hyphens");
    }
}
```

Sanitize Message Content

Validate and sanitize incoming message data:

```
protected override async Task MessageReceived(IFlowMessage message)
{
    try
    {
        var userInput = message.Get<string>("userInput");

        // Validate input exists
        if (string.IsNullOrWhiteSpace(userInput))
        {
            var error = "Input cannot be empty";
            this.SetStatus(Status.Warning, error);
            message.SetError(error);
            return;
        }

        // Sanitize HTML/XML characters
        var sanitized = userInput.Trim()
            .Replace("<", "&lt;")
            .Replace(">", "&gt;")
            .Replace("\"", "&quot;")
            .Replace("'", "&#39;");

        // Validate length
        if (sanitized.Length > 1000)
        {
```

```
            var error = "Input exceeds maximum length of 1000 characters";
            this.SetStatus(Status.Warning, error);
            message.SetError(error);
            return;
        }

        message.Set("sanitizedInput", sanitized);
    }
    catch (Exception ex)
    {
        Error(ex, "Input validation failed");
        var error = "Input validation failed";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    finally
    {
        await this.Next(message);
    }
}
```

Prevent SQL Injection

When working with databases, always use parameterized queries:

```
// ❌ WRONG - Vulnerable to SQL injection
var query = $"SELECT * FROM users WHERE name = '{userName}'";

// ✅ CORRECT - Use parameterized queries
var query = "SELECT * FROM users WHERE name = @name";
using var command = this.connection.CreateCommand();
command.CommandText = query;

var param = command.CreateParameter();
param.ParameterName = "@name";
param.Value = userName;
command.Parameters.Add(param);
```

Secure Logging

Never Log Sensitive Information

```
protected override async Task MessageReceived(IFlowMessage message)
{
```

```
    try
    {
        var credentials = await
this.GetCredentialContentAsync<CredentialWithUsernamePassword>(
            this.Settings.ServiceCredential.Value);

        // ✅ Safe logging
        Information("Processing message for user {Username}",
            credentials?.Value?.Username);

        // ❌ NEVER log passwords, tokens, or keys
        // Information("Password: {Password}", credentials.Value.Password);
        // Information("Full message: {Content}", message.ToJSON()); // May contain
sensitive data

        var result = await this.ProcessSecurely(message, credentials.Value);

        // ✅ Log results without sensitive data
        Information("Processing completed successfully for user {Username}",
            credentials?.Value?.Username);
    }
    catch (Exception ex)
    {
        // ✅ Log errors without exposing sensitive details
        Error(ex, "Processing failed");
        var error = "Processing failed";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    finally
    {
        await this.Next(message);
    }
}
```

Safe Logging Patterns

```
// ✅ SAFE - Log operation without data
Information("Database query executed successfully");

// ✅ SAFE - Log counts and statistics
Information("Processed {Count} records in {ElapsedMs}ms", count, elapsed);

// ✅ SAFE - Log non-sensitive identifiers
Information("Processing order {OrderId} for customer {CustomerId}",
```

```
orderId, customerId);

// ❌ UNSAFE - Don't log entire messages
// Debug("Received message: {@Message}", message);

// ❌ UNSAFE - Don't log credentials
// Debug("Using connection: {ConnectionString}", connectionString);

// ❌ UNSAFE - Don't log API keys
// Debug("API Key: {ApiKey}", apiKey);
```

Error Handling Security

Avoid Information Disclosure

Never expose internal system details in error messages:

```
protected override async Task MessageReceived(IFlowMessage message)
{
    try
    {
        await this.ProcessMessage(message);
    }
    catch (SqlException ex)
    {
        // ❌ Don't expose internal details
        // message.SetError($"SQL Error: {ex.Message}");
        // message.SetError($"Query failed: {query}");

        // ✅ Use generic error messages
        var error = "Database operation failed";
        Error(ex, "Database operation failed");
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    catch (UnauthorizedAccessException ex)
    {
        // ✅ Handle security exceptions appropriately
        Warning(ex, "Access denied for operation");
        var error = "Access denied";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    catch (HttpRequestException ex)
    {
```

```
        // ❌ Don't expose URLs or endpoints
        // message.SetError($"Failed to connect to {url}: {ex.Message}");

        // ✅ Generic external service error
        Error(ex, "External service request failed");
        var error = "External service unavailable";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    catch (Exception ex)
    {
        // ✅ Generic handling for unexpected errors
        Error(ex, "Unexpected error occurred");
        var error = "An error occurred while processing the message";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    finally
    {
        await this.Next(message);
    }
}
```

Secure Communication

Enforce HTTPS

Always use HTTPS for external communications:

```
public override void Validate(SettingsValidator validator)
{
    if (!string.IsNullOrWhiteSpace(this.ApiEndpoint))
    {
        if (Uri.TryCreate(this.ApiEndpoint, UriKind.Absolute, out var uri))
        {
            if (!uri.Scheme.Equals("https", StringComparison.OrdinalIgnoreCase))
            {
                validator.AddError(nameof(this.ApiEndpoint),
                    "Only HTTPS endpoints are allowed for security");
            }
        }
    }
}
```

Certificate Validation

```
// ❌ NEVER disable certificate validation in production
// ServicePointManager.ServerCertificateValidationCallback =
//     (sender, cert, chain, sslPolicyErrors) => true;

// ✅ Use proper certificate handling
var handler = new HttpClientHandler();
if (this.Settings.AllowSelfSignedCerts)
{
    // Only allow in non-production environments
    if (this.Settings.Environment.Equals("Development",
        StringComparison.OrdinalIgnoreCase))
    {
        handler.ServerCertificateCustomValidationCallback =
            HttpClientHandler.DangerousAcceptAnyServerCertificateValidator;
    }
}
```

Configuration Security

Use Secure Defaults

```
public class SecureModuleSettings : FlowModuleSettings
{
    [Display(Name = "Enable SSL")]
    [DefaultValue(true)]
    public bool EnableSsl { get; set; } = true; // Secure by default

    [Range(1, 300)]
    [Display(Name = "Timeout (seconds)")]
    [DefaultValue(30)]
    public int TimeoutSeconds { get; set; } = 30;

    [Display(Name = "Allow Self-Signed Certificates")]
    [DefaultValue(false)]
    public bool AllowSelfSignedCerts { get; set; } = false; // Secure default

    [Display(Name = "Environment")]
    [DefaultValue("Production")]
    public string Environment { get; set; } = "Production";

    public override void Validate(SettingsValidator validator)
    {
        // Warn about insecure configurations
        if (!this.EnableSsl &&
            this.Environment.Equals("Production",
```

```
StringComparison.OrdinalIgnoreCase))
        {
            validator.AddError(nameof(this.EnableSsl),
                "SSL is disabled - connection will not be encrypted");
        }

        if (this.AllowSelfSignedCerts &&
            this.Environment.Equals("Production",
StringComparison.OrdinalIgnoreCase))
        {
            validator.AddError(nameof(this.AllowSelfSignedCerts),
                "Self-signed certificates should not be allowed in production");
        }

        base.Validate(validator);
    }
}
```

Resource Management Security

Secure File Handling

When working with resources:

```
public override async Task<IError?> Initialize()
{
    if (!this.Settings.ConfigFile.HasValue)
    {
        return new Error("Configuration file not specified");
    }

    // Get resource securely
    var resource = await this.GetResourceContentAsync<string>
(this.Settings.ConfigFile.Value);
    if (!resource.IsSuccess)
    {
        return new Error("Failed to load configuration file");
    }

    var fileContent = resource.Result;

    // Validate file content before parsing
    if (string.IsNullOrEmpty(fileContent))
    {
        return new Error("Configuration file is empty");
```

```
    }

    // Limit file size to prevent DoS
    if (fileContent.Length > 1000000) // 1MB limit
    {
        return new Error("Configuration file exceeds maximum size of 1MB");
    }

    try
    {
        // Parse with error handling
        this.configuration = JsonSerializer.Deserialize<ModuleConfiguration>
(fileContent);

        if (this.configuration == null)
        {
            return new Error("Invalid configuration format");
        }
    }
    catch (JsonException ex)
    {
        Error(ex, "Failed to parse configuration file");
        return new Error("Configuration file format is invalid");
    }

    return await base.Initialize();
}
```

Security Checklist

Use this checklist when developing modules:

Credentials

- [ ] No passwords or keys in settings properties
- [ ] All sensitive data uses credential references
- [ ] Credentials never logged
- [ ] Credential access includes null checks

Input Validation

- [ ] All settings validated in `Validate` method
- [ ] Message properties validated before use
- [ ] String lengths checked
- [ ] Input sanitized for injection attacks

- [ ] HTTPS enforced for URLs

Error Handling

- [ ] Generic error messages to users
- [ ] Detailed errors only in logs
- [ ] No internal implementation details exposed
- [ ] Exception types handled appropriately

Logging

- [ ] No credentials in logs
- [ ] No sensitive user data in logs
- [ ] No full message content in logs
- [ ] Error messages don't expose internals

Communication

- [ ] HTTPS used for external APIs
- [ ] Certificate validation enabled
- [ ] Timeouts configured
- [ ] Secure defaults used

Best Practices

- [ ] Secure by default configuration
- [ ] Validation warnings for insecure settings
- [ ] Production environment checks
- [ ] Resource size limits enforced

Common Security Mistakes

1. Logging Credentials

```
// ❌ NEVER DO THIS
Information("Connecting with password: {Password}", password);
Information("API Key: {Key}", apiKey);
Debug("Full config: {@Settings}", Settings);
```

2. Exposing Internal Errors

```
// ❌ NEVER DO THIS
catch (Exception ex)
{
```

```
    message.SetError(ex.ToString()); // Exposes stack traces
    message.SetError($"Query: {sqlQuery}"); // Exposes SQL
}
```

3. Weak Input Validation

```
// ❌ NEVER DO THIS
var id = message.Get<string>("id");
var query = $"SELECT * FROM users WHERE id = {id}"; // SQL injection!
```

4. Disabling Security Features

```
// ❌ NEVER DO THIS
ServicePointManager.ServerCertificateValidationCallback =
    (sender, cert, chain, sslPolicyErrors) => true; // Accepts any certificate!
```

Summary

Security is not optional. Follow these principles:

1. Never expose credentials - Use credential references
2. Validate all input - Trust nothing from external sources
3. Log safely - Never log sensitive information
4. Use generic error messages - Don't expose internals
5. Enforce HTTPS - Secure communication by default
6. Secure defaults - Make the safe choice the default choice

Remember: A security vulnerability in your module can compromise the entire flow and potentially the entire system. Always code defensively and assume all input is malicious until proven otherwise.

>> Performance Optimization

Performance Optimization

Guidelines and patterns for building high-performance Crosser modules.

General Principles

1. Async/Await properly - Don't block threads
2. Initialize resources once - Avoid recreation per message
3. Batch when appropriate - Reduce external call overhead
4. Use appropriate collections - Thread-safe when needed
5. Dispose resources properly - Prevent memory leaks

Resource Initialization

Initialize Once, Use Many Times

❌ Wrong - Recreates resources per message:

```
protected override async Task MessageReceived(IFlowMessage message)
{
    // Creating new instances for every message is expensive!
    var httpClient = new HttpClient();
    var result = await httpClient.GetAsync("https://api.example.com/data");
    httpClient.Dispose();

    await this.Next(message);
}
```

✅ Correct - Initialize once, reuse:

```
private HttpClient httpClient;

public override async Task<IError?> Initialize()
{
    // Create once during initialization
    this.httpClient = new HttpClient
    {
        BaseAddress = new Uri(this.Settings.ApiBaseUrl),
        Timeout = TimeSpan.FromSeconds(this.Settings.TimeoutSeconds)
    };

    this.httpClient.DefaultRequestHeaders.Add("User-Agent", "CrosserModule/1.0");

    return await base.Initialize();
```

```
}

protected override async Task MessageReceived(IFlowMessage message)
{
    // Reuse the same instance
    var result = await this.httpClient.GetAsync("/data");
    await this.Next(message);
}

public override async Task Stop()
{
    // Cleanup when module stops
    this.httpClient?.Dispose();
    await base.Stop();
}
```

Async/Await Patterns

Non-Blocking Operations

❌ Wrong - Blocking calls:

```
protected override async Task MessageReceived(IFlowMessage message)
{
    // .Result blocks the thread!
    var data = this.httpClient.GetAsync("/data").Result;

    // .Wait() blocks the thread!
    this.ProcessDataAsync(message).Wait();

    await this.Next(message);
}
```

✅ Correct - Proper async/await:

```
protected override async Task MessageReceived(IFlowMessage message)
{
    // Non-blocking awaits
    var data = await this.httpClient.GetAsync("/data");
    await this.ProcessDataAsync(message);
    await this.Next(message);
}
```

CPU-Intensive Operations

For CPU-bound work, use `Task.Run` to avoid blocking when message processing order is not critical:

```
protected override async Task MessageReceived(IFlowMessage message)
{
    try
    {
        var data = message.Get<byte[]>("imageData");

        // Offload CPU-intensive work to thread pool
        var processed = await Task.Run(() => this.ProcessImage(data));

        message.Set("processedImage", processed);
    }
    catch (Exception ex)
    {
        Error(ex, "Image processing failed");
        var error = "Image processing failed";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    finally
    {
        await this.Next(message);
    }
}

private byte[] ProcessImage(byte[] imageData)
{
    // CPU-intensive synchronous processing
    // This runs on a thread pool thread, not blocking the module
    return ImageProcessor.Resize(imageData, 800, 600);
}
```

Batching Patterns

One-Shot Timer Batching

This pattern collects messages and processes them in batches, using a one-shot timer that only fires when needed:

```
public class BatchProcessorModule : FlowModule<BatchProcessorModuleSettings>
{
    private readonly ConcurrentQueue<IFlowMessage> messageQueue = new();
    private readonly SemaphoreSlim semaphore = new(1, 1);
    private Timer batchTimer;
```

```
private volatile bool isProcessing;

public override string UserFriendlyName => "Batch Processor";

public BatchProcessorModule() : base(FlowModuleType.Function) { }

protected override async Task MessageReceived(IFlowMessage message)
{
    await this.semaphore.WaitAsync();
    try
    {
        var wasEmpty = this.messageQueue.IsEmpty;

        // Add message to queue (clone to avoid reference issues)
        this.messageQueue.Enqueue(message.Clone());

        if (wasEmpty)
        {
            // First message - start the timer
            this.StartBatchTimer();
        }

        // Size threshold - process immediately if batch is full
        if (this.messageQueue.Count >= this.Settings.BatchSize)
        {
            this.StopBatchTimer();
            await this.ProcessBatch();
        }
    }
    finally
    {
        this.semaphore.Release();
        // Always forward original message
        await this.Next(message);
    }
}

private void StartBatchTimer()
{
    if (this.batchTimer == null)
    {
        this.batchTimer = new Timer(
            async _ => await this.ProcessBatchFromTimer(),
            null,
            Timeout.InfiniteTimeSpan,  // Start disabled
            Timeout.InfiniteTimeSpan   // One-shot timer
```

```
        );
    }

    // Set timer to fire once after timeout period
    this.batchTimer.Change(
        TimeSpan.FromSeconds(this.Settings.BatchTimeoutSeconds),
        Timeout.InfiniteTimeSpan
    );
}

private void StopBatchTimer()
{
    this.batchTimer?.Change(Timeout.InfiniteTimeSpan, Timeout.InfiniteTimeSpan);
}

private async Task ProcessBatchFromTimer()
{
    await this.ProcessBatch();
}

private async Task ProcessBatch()
{
    if (this.isProcessing || this.messageQueue.IsEmpty)
    {
        return;
    }

    this.isProcessing = true;
    try
    {
        var batch = new List<IFlowMessage>();

        // Dequeue up to BatchSize messages
        while (batch.Count < this.Settings.BatchSize &&
               this.messageQueue.TryDequeue(out var msg))
        {
            batch.Add(msg);
        }

        if (batch.Any())
        {
            Information("Processing batch of {Count} messages", batch.Count);

            // Extract only needed data for bulk operation
            var ids = batch.Select(m => m.Get<string>("id")).ToList();
```

```
            // Process batch in single external call
            var result = await this.ProcessBulkAsync(ids);

            // Send batch result
            var batchMsg = new FlowMessage();
            batchMsg.Set("batchSize", batch.Count);
            batchMsg.Set("successCount", result.SuccessCount);
            batchMsg.Set("failedCount", result.FailedCount);
            batchMsg.Set("processedAt", DateTime.UtcNow);

            await this.Next(batchMsg);

            // Restart timer if messages remain
            if (!this.messageQueue.IsEmpty)
            {
                this.StartBatchTimer();
            }
        }
    }
    catch (Exception ex)
    {
        Error(ex, "Batch processing failed");
        this.SetStatus(Status.Warning, "Batch processing failed");
    }
    finally
    {
        this.isProcessing = false;
    }
}

private async Task<BatchResult> ProcessBulkAsync(List<string> ids)
{
    // Simulate bulk API call
    await Task.Delay(100);
    return new BatchResult { SuccessCount = ids.Count, FailedCount = 0 };
}

public override async Task Stop()
{
    this.StopBatchTimer();
    this.batchTimer?.Dispose();

    // Process remaining messages
    if (!this.messageQueue.IsEmpty)
    {
        await this.ProcessBatch();
```

```
        }

        this.semaphore?.Dispose();
        await base.Stop();
    }
}

public class BatchProcessorModuleSettings : FlowModuleSettings
{
    [Display(Name = "Batch Size")]
    [Description("Maximum number of messages to process in a batch")]
    [Range(1, 1000)]
    [DefaultValue(100)]
    public int BatchSize { get; set; } = 100;

    [Display(Name = "Batch Timeout (seconds)")]
    [Description("Maximum time to wait before processing incomplete batch")]
    [Range(1, 300)]
    [DefaultValue(5)]
    public int BatchTimeoutSeconds { get; set; } = 5;
}

public class BatchResult
{
    public int SuccessCount { get; set; }
    public int FailedCount { get; set; }
}
```

Key Design Points

- One-Shot Timer: Fires exactly once per batch period, no unnecessary checks
- Smart Reset: Timer restarts only when messages remain after processing
- Efficient Triggers: Size threshold stops timer and processes immediately
- ConcurrentQueue: Lock-free thread-safe operations
- Minimal Processing: Extract only required fields for bulk operations
- SemaphoreSlim: Prevents race conditions when starting/stopping timer

Thread-Safe Collections

Use Concurrent Collections for Shared State

❌ Wrong - Not thread-safe:

```
private readonly List<IFlowMessage> messageBuffer = new();
private readonly object lockObject = new();
```

```
protected override async Task MessageReceived(IFlowMessage message)
{
    lock (this.lockObject)
    {
        this.messageBuffer.Add(message);
    }

    await this.Next(message);
}
```

✅ Correct - Use concurrent collections:

```
private readonly ConcurrentQueue<IFlowMessage> messageBuffer = new();

protected override async Task MessageReceived(IFlowMessage message)
{
    // Lock-free, thread-safe enqueue
    this.messageBuffer.Enqueue(message);

    await this.Next(message);
}
```

Available Concurrent Collections

```
// Queue: First-in, first-out
private readonly ConcurrentQueue<T> queue = new();

// Stack: Last-in, first-out
private readonly ConcurrentStack<T> stack = new();

// Dictionary: Thread-safe key-value pairs
private readonly ConcurrentDictionary<TKey, TValue> dictionary = new();

// Bag: Unordered collection
private readonly ConcurrentBag<T> bag = new();
```

Memory Management

Dispose Resources Properly

Implement `IDisposable` for modules with unmanaged resources:

```
public class DatabaseModule : FlowModule<DatabaseModuleSettings>, IDisposable
{
    private SqlConnection connection;
    private Timer heartbeatTimer;
    private bool disposed = false;

    public override async Task<IError?> Initialize()
    {
        this.connection = new SqlConnection(this.Settings.ConnectionString);
        await this.connection.OpenAsync();

        this.heartbeatTimer = new Timer(
            _ => this.CheckConnection(),
            null,
            TimeSpan.FromMinutes(1),
            TimeSpan.FromMinutes(1)
        );

        return await base.Initialize();
    }

    protected override async Task MessageReceived(IFlowMessage message)
    {
        // Use connection...
        await this.Next(message);
    }

    public override async Task Stop()
    {
        // Dispose resources
        this.heartbeatTimer?.Dispose();
        this.connection?.Close();
        this.connection?.Dispose();

        await base.Stop();
    }

    protected virtual void Dispose(bool disposing)
    {
        if (!this.disposed)
        {
            if (disposing)
            {
                // Dispose managed resources
                this.heartbeatTimer?.Dispose();
                this.connection?.Dispose();
```

```
            }

            this.disposed = true;
        }
    }

    public void Dispose()
    {
        this.Dispose(true);
        GC.SuppressFinalize(this);
    }
}
```

Avoid Memory Leaks

Common memory leak patterns to avoid:

```
// ❌ WRONG - Unbounded growth
private readonly List<IFlowMessage> allMessages = new();

protected override async Task MessageReceived(IFlowMessage message)
{
    this.allMessages.Add(message); // Grows forever!
    await this.Next(message);
}

// ✅ CORRECT - Bounded collection
private readonly Queue<IFlowMessage> recentMessages = new();
private const int MaxBufferSize = 1000;

protected override async Task MessageReceived(IFlowMessage message)
{
    this.recentMessages.Enqueue(message);

    // Maintain size limit
    while (this.recentMessages.Count > MaxBufferSize)
    {
        this.recentMessages.Dequeue();
    }

    await this.Next(message);
}
```

Unsubscribe from Events

```
// ❌ WRONG - Event handler leak
public override async Task<IError?> Start()
{
    ExternalService.OnDataReceived += this.HandleData; // Never unsubscribed!
    return await base.Start();
}

// ✅ CORRECT - Proper cleanup
public override async Task<IError?> Start()
{
    ExternalService.OnDataReceived += this.HandleData;
    return await base.Start();
}

public override async Task Stop()
{
    ExternalService.OnDataReceived -= this.HandleData;
    await base.Stop();
}
```

Connection Pooling

Database Connection Pooling

Most ADO.NET providers handle connection pooling automatically:

```
public class DatabaseModule : FlowModule<DatabaseModuleSettings>
{
    private string connectionString;

    public override async Task<IError?> Initialize()
    {
        // Connection string with pooling enabled (default for most providers)
        this.connectionString = this.Settings.ConnectionString;

        // Test connection
        using var connection = new SqlConnection(this.connectionString);
        await connection.OpenAsync();

        return await base.Initialize();
    }

    protected override async Task MessageReceived(IFlowMessage message)
    {
        // Create connection per message - pooling handles efficiency
```

```
        using var connection = new SqlConnection(this.connectionString);
        await connection.OpenAsync();

        // Use connection
        using var command = connection.CreateCommand();
        command.CommandText = "SELECT * FROM table WHERE id = @id";
        command.Parameters.AddWithValue("@id", message.Get<int>("id"));

        using var reader = await command.ExecuteReaderAsync();
        // Process results...

        await this.Next(message);

        // Connection automatically returned to pool when disposed
    }
}
```

HttpClient Reuse

```
// ✅ CORRECT - Single HttpClient instance
private HttpClient httpClient;

public override async Task<IError?> Initialize()
{
    this.httpClient = new HttpClient
    {
        BaseAddress = new Uri(this.Settings.ApiBaseUrl),
        Timeout = TimeSpan.FromSeconds(30)
    };

    // HttpClient handles connection pooling internally
    return await base.Initialize();
}

protected override async Task MessageReceived(IFlowMessage message)
{
    // Reuse same HttpClient - it handles connection pooling
    var response = await this.httpClient.GetAsync("/endpoint");
    await this.Next(message);
}
```

Performance Monitoring

Built-in Timing

```
private readonly Stopwatch processingTimer = new();

protected override async Task MessageReceived(IFlowMessage message)
{
    this.processingTimer.Restart();

    try
    {
        await this.ProcessMessage(message);
    }
    finally
    {
        this.processingTimer.Stop();

        // Log slow operations
        if (this.processingTimer.ElapsedMilliseconds > 1000)
        {
            Warning("Slow processing: {ElapsedMs}ms for message",
                this.processingTimer.ElapsedMilliseconds);
        }

        await this.Next(message);
    }
}
```

Message Rate Tracking

```
private int messageCount;
private DateTime lastResetTime = DateTime.UtcNow;

protected override async Task MessageReceived(IFlowMessage message)
{
    Interlocked.Increment(ref this.messageCount);

    var elapsed = DateTime.UtcNow - this.lastResetTime;
    if (elapsed.TotalSeconds >= 60)
    {
        var rate = this.messageCount / elapsed.TotalSeconds;
        Information("Processing rate: {Rate:F2} messages/second", rate);

        this.messageCount = 0;
        this.lastResetTime = DateTime.UtcNow;
    }

    await this.ProcessMessage(message);
```

```
    await this.Next(message);
}
```

Performance Checklist

- [ ] Resources initialized once, not per message
- [ ] HttpClient/database connections reused
- [ ] Async/await used correctly (no blocking)
- [ ] CPU-intensive work offloaded with Task.Run
- [ ] Concurrent collections used for shared state
- [ ] Resources disposed properly in Stop()
- [ ] Event handlers unsubscribed
- [ ] Collections have size limits
- [ ] Batching used for bulk operations
- [ ] Timers disposed correctly

Common Performance Mistakes

1. Creating Resources Per Message

```
// ❌ Creates 1000 HttpClients for 1000 messages!
protected override async Task MessageReceived(IFlowMessage message)
{
    var client = new HttpClient(); // Don't do this!
    await client.GetAsync("...");
    client.Dispose();
}
```

2. Blocking on Async Operations

```
// ❌ Blocks thread pool thread
protected override async Task MessageReceived(IFlowMessage message)
{
    var result = SomeAsyncOperation().Result; // Blocking!
}
```

3. Unbounded Collections

```
// ❌ Grows without limit
private readonly List<IFlowMessage> buffer = new();

protected override async Task MessageReceived(IFlowMessage message)
```

```
{
    this.buffer.Add(message); // Memory leak!
}
```

4. Unnecessary Synchronization

```
// ❌ Lock on every message when concurrent collection would work
private readonly List<int> values = new();
private readonly object lockObj = new();

protected override async Task MessageReceived(IFlowMessage message)
{
    lock (this.lockObj) // Unnecessary lock!
    {
        this.values.Add(message.Get<int>("value"));
    }
}

// ✅ Use concurrent collection instead
private readonly ConcurrentBag<int> values = new();

protected override async Task MessageReceived(IFlowMessage message)
{
    this.values.Add(message.Get<int>("value")); // Lock-free!
}
```

Summary

Key principles for high-performance modules:

1. Initialize once - Create resources during Initialize(), not per message
2. Don't block - Use async/await properly, never use .Result or .Wait()
3. Batch efficiently - Use one-shot timers and concurrent collections
4. Manage memory - Dispose resources, limit collection sizes, unsubscribe from events
5. Monitor performance - Track processing times and message rates
6. Use thread-safe collections - ConcurrentQueue, ConcurrentDictionary when needed
7. Pool connections - Reuse HttpClient and rely on connection pooling

Remember: Performance issues in your module can affect the entire flow. Design for efficiency from the start.

>> External Integrations

External Integrations

This guide covers how to integrate your modules with external services, manage credentials securely, and access resources.

HTTP/REST API Integration

Basic Setup

```
public class RestApiModule : FlowModule<RestApiModuleSettings>
{
    private HttpClient httpClient;

    public override string UserFriendlyName => "REST API Client";

    public RestApiModule() : base(FlowModuleType.Output) { }

    public override async Task<IError?> Initialize()
    {
        // Get API credentials
        if (!this.Settings.ApiCredential.HasValue)
        {
            return new Error("API credential not configured");
        }

        var apiCredential = await
this.GetCredentialContentAsync<CredentialWithApiKey>(
            this.Settings.ApiCredential.Value);

        if (apiCredential.IsError || apiCredential.Value is null)
        {
            return apiCredential.Error ?? new Error("Failed to get API credential");
        }

        // Configure HTTP client
        this.httpClient = new HttpClient
        {
            BaseAddress = new Uri(this.Settings.ApiBaseUrl),
            Timeout = TimeSpan.FromSeconds(this.Settings.TimeoutSeconds)
        };

        this.httpClient.DefaultRequestHeaders.Add("Authorization",
            $"Bearer {apiCredential.Value.ApiKey}");
        this.httpClient.DefaultRequestHeaders.Add("User-Agent",
            "CrosserModule/1.0");
```

```
    // Test connection
    try
    {
        var response = await this.httpClient.GetAsync("/health");
        if (!response.IsSuccessStatusCode)
        {
            return new Error($"API health check failed: {response.StatusCode}");
        }

        Information("Successfully connected to API at {BaseUrl}",
            this.Settings.ApiBaseUrl);
    }
    catch (HttpRequestException ex)
    {
        return new Error($"Failed to connect to API: {ex.Message}");
    }

    return await base.Initialize();
}

protected override async Task MessageReceived(IFlowMessage message)
{
    try
    {
        var endpoint = message.Get<string>("endpoint") ?? "/data";
        var method = message.Get<string>("method") ?? "GET";
        var body = message.Get<object>("body");

        HttpResponseMessage response;

        switch (method.ToUpper())
        {
            case "GET":
                response = await this.httpClient.GetAsync(endpoint);
                break;

            case "POST":
                var content = new StringContent(
                    JsonSerializer.Serialize(body),
                    Encoding.UTF8,
                    "application/json"
                );
                response = await this.httpClient.PostAsync(endpoint, content);
                break;
```

```
            case "PUT":
                content = new StringContent(
                    JsonSerializer.Serialize(body),
                    Encoding.UTF8,
                    "application/json"
                );
                response = await this.httpClient.PutAsync(endpoint, content);
                break;

            case "DELETE":
                response = await this.httpClient.DeleteAsync(endpoint);
                break;

            default:
                var error = $"Unsupported HTTP method: {method}";
                this.SetStatus(Status.Warning, error);
                message.SetError(error);
                return;
        }

        if (response.IsSuccessStatusCode)
        {
            var responseData = await response.Content.ReadAsStringAsync();
            message.Set("response", responseData);
            message.Set("statusCode", (int)response.StatusCode);
            message.Set("success", true);
        }
        else
        {
            var error = $"API request failed: {response.StatusCode}";
            this.SetStatus(Status.Warning, error);
            message.SetError(error);
            message.Set("statusCode", (int)response.StatusCode);
        }
    }
    catch (HttpRequestException ex)
    {
        Error(ex, "HTTP request failed");
        var error = $"HTTP request failed: {ex.Message}";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    catch (Exception ex)
    {
        Error(ex, "Unexpected error");
        this.SetStatus(Status.Warning, ex.Message);
```

```
            message.SetError(ex.Message);
        }
        finally
        {
            // Always forward the message
            await this.Next(message);
        }
    }

    public override async Task Stop()
    {
        this.httpClient?.Dispose();
        await base.Stop();
    }
}

public class RestApiModuleSettings : FlowModuleSettings
{
    [Required]
    [Display(Name = "API Base URL")]
    [Description("Base URL of the REST API")]
    public string ApiBaseUrl { get; set; }

    [JsonSchemaExtensionData("x-credential", "ApiKey")]
    [Display(Name = "API Key")]
    [Description("API key for authentication")]
    public Guid? ApiCredential { get; set; }

    [Range(1, 300)]
    [Display(Name = "Timeout (seconds)")]
    [DefaultValue(30)]
    public int TimeoutSeconds { get; set; } = 30;

    public override void Validate(SettingsValidator validator)
    {
        if (!string.IsNullOrWhiteSpace(this.ApiBaseUrl))
        {
            if (!Uri.TryCreate(this.ApiBaseUrl, UriKind.Absolute, out var uri))
            {
                validator.AddError(nameof(this.ApiBaseUrl),
                    "API Base URL must be a valid URL");
            }
        }

        base.Validate(validator);
```

```
    }
}
```

Database Integration

SQL Server Example

```
public class DatabaseModule : FlowModule<DatabaseModuleSettings>
{
    private string connectionString;

    public override string UserFriendlyName => "Database Writer";

    public DatabaseModule() : base(FlowModuleType.Output) { }

    public override async Task<IError?> Initialize()
    {
        // Get database credentials
        if (!this.Settings.DbCredential.HasValue)
        {
            return new Error("Database credential not configured");
        }

        var credentials = await
this.GetCredentialContentAsync<CredentialWithConnectionString>(
            this.Settings.DbCredential.Value);

        if (credentials.IsError || credentials.Value is null)
        {
            return credentials.Error ?? new Error("Failed to get
database credential");
        }

        this.connectionString = credentials.Value.ConnectionString;

        // Test connection
        try
        {
            using var connection = new SqlConnection(this.connectionString);
            await connection.OpenAsync();

            Information("Successfully connected to database");
        }
        catch (SqlException ex)
        {
```

```
            return new Error($"Database connection failed: {ex.Message}");
        }

        return await base.Initialize();
    }

    protected override async Task MessageReceived(IFlowMessage message)
    {
        try
        {
            // Create new connection per message (uses connection pooling)
            using var connection = new SqlConnection(this.connectionString);
            await connection.OpenAsync();

            var tableName = this.Settings.TableName;
            var data = message.Get<Dictionary<string, object>>("data");

            if (data == null || !data.Any())
            {
                var error = "No data to insert";
                this.SetStatus(Status.Warning, error);
                message.SetError(error);
                return;
            }

            // Build parameterized INSERT query
            var columns = string.Join(", ", data.Keys);
            var parameters = string.Join(", ", data.Keys.Select(k => $"@{k}"));
            var query = $"INSERT INTO {tableName} ({columns})
VALUES ({parameters})";

            using var command = connection.CreateCommand();
            command.CommandText = query;
            command.CommandTimeout = this.Settings.QueryTimeout;

            // Add parameters safely
            foreach (var kvp in data)
            {
                var param = command.CreateParameter();
                param.ParameterName = $"@{kvp.Key}";
                param.Value = kvp.Value ?? DBNull.Value;
                command.Parameters.Add(param);
            }

            // Execute query
            var rowsAffected = await command.ExecuteNonQueryAsync();
```

```
            message.Set("rowsAffected", rowsAffected);
            message.Set("success", true);
            message.Set("insertedAt", DateTime.UtcNow);
        }
        catch (SqlException ex)
        {
            Error(ex, "Database operation failed");
            var error = $"Database operation failed: {ex.Message}";
            this.SetStatus(Status.Warning, error);
            message.SetError(error);
        }
        catch (Exception ex)
        {
            Error(ex, "Unexpected error");
            this.SetStatus(Status.Warning, ex.Message);
            message.SetError(ex.Message);
        }
        finally
        {
            // Always forward the message
            await this.Next(message);
        }
    }
}

public class DatabaseModuleSettings : FlowModuleSettings
{
    [JsonSchemaExtensionData("x-credential", "ConnectionString")]
    [Display(Name = "Database Credentials")]
    [Description("Database connection string")]
    public Guid? DbCredential { get; set; }

    [Required]
    [Display(Name = "Table Name")]
    [Description("Name of the database table")]
    public string TableName { get; set; }

    [Range(1, 300)]
    [Display(Name = "Query Timeout (seconds)")]
    [DefaultValue(30)]
    public int QueryTimeout { get; set; } = 30;
}
```

Multiple Database Providers

```
public class MultiDbModule : FlowModule<MultiDbModuleSettings>
{
    private IDbConnection connection;

    public override async Task<IError?> Initialize()
    {
        var credentials = await
this.GetCredentialContentAsync<CredentialWithConnectionString>(
            this.Settings.DbCredential.Value);

        if (credentials.IsError || credentials.Value is null)
        {
            return credentials.Error ?? new Error("Failed to get
database credential");
        }

        var connString = credentials.Value.ConnectionString;

        // Create connection based on provider
        this.connection = this.Settings.DatabaseProvider.ToLower() switch
        {
            "sqlserver" => new SqlConnection(connString),
            "postgresql" => new NpgsqlConnection(connString),
            "mysql" => new MySqlConnection(connString),
            _ => null
        };

        if (this.connection == null)
        {
            return new Error($"Unsupported database provider:
{this.Settings.DatabaseProvider}");
        }

        // Test connection
        try
        {
            await ((DbConnection)this.connection).OpenAsync();
            Information("Successfully connected to {Provider} database",
                this.Settings.DatabaseProvider);
        }
        catch (DbException ex)
        {
            return new Error($"Database connection failed: {ex.Message}");
        }

        return await base.Initialize();
```

```
    }

    public override async Task Stop()
    {
        this.connection?.Close();
        this.connection?.Dispose();
        await base.Stop();
    }
}

public class MultiDbModuleSettings : FlowModuleSettings
{
    [Required]
    [Display(Name = "Database Provider")]
    [Description("Type of database")]
    [DefaultValue("SqlServer")]
    public string DatabaseProvider { get; set; } = "SqlServer";

    [JsonSchemaExtensionData("x-credential", "ConnectionString")]
    [Display(Name = "Database Credentials")]
    public Guid? DbCredential { get; set; }
}
```

Message Queue Integration

RabbitMQ Example

```
public class RabbitMqModule : FlowModule<RabbitMqModuleSettings>
{
    private IConnection connection;
    private IModel channel;

    public override string UserFriendlyName => "RabbitMQ Publisher";

    public RabbitMqModule() : base(FlowModuleType.Output) { }

    public override async Task<IError?> Initialize()
    {
        // Get credentials
        if (!this.Settings.QueueCredential.HasValue)
        {
            return new Error("Queue credential not configured");
        }

        var credentials = await
```

```
this.GetCredentialContentAsync<CredentialWithUsernamePassword>(
            this.Settings.QueueCredential.Value);

        if (credentials.IsError || credentials.Value is null)
        {
            return credentials.Error ?? new Error("Failed to get queue credential");
        }

        // Setup connection
        var factory = new ConnectionFactory
        {
            HostName = this.Settings.Host,
            Port = this.Settings.Port,
            UserName = credentials.Value.Username,
            Password = credentials.Value.Password,
            VirtualHost = this.Settings.VirtualHost,
            RequestedHeartbeat = TimeSpan.FromSeconds(60),
            NetworkRecoveryInterval = TimeSpan.FromSeconds(10),
            AutomaticRecoveryEnabled = true
        };

        try
        {
            this.connection = factory.CreateConnection($"{this.UserFriendlyName}-
{this.Id}");
            this.channel = this.connection.CreateModel();

            // Declare queue
            this.channel.QueueDeclare(
                queue: this.Settings.QueueName,
                durable: this.Settings.DurableQueue,
                exclusive: false,
                autoDelete: false,
                arguments: null
            );

            Information("Connected to RabbitMQ queue {Queue} at {Host}:{Port}",
                this.Settings.QueueName, this.Settings.Host, this.Settings.Port);
        }
        catch (Exception ex)
        {
            return new Error($"Failed to connect to RabbitMQ: {ex.Message}");
        }

        return await base.Initialize();
    }
```

```
protected override async Task MessageReceived(IFlowMessage message)
{
    try
    {
        var messageBody = message.ToJSON();
        var body = Encoding.UTF8.GetBytes(messageBody);

        var properties = this.channel.CreateBasicProperties();
        properties.Persistent = this.Settings.PersistentMessages;
        properties.ContentType = "application/json";
        properties.MessageId = Guid.NewGuid().ToString();
        properties.Timestamp = new AmqpTimestamp(
            DateTimeOffset.UtcNow.ToUnixTimeSeconds());

        // Optional routing key from message
        var routingKey = message.Get<string>("routingKey")
            ?? this.Settings.RoutingKey
            ?? this.Settings.QueueName;

        this.channel.BasicPublish(
            exchange: this.Settings.Exchange ?? "",
            routingKey: routingKey,
            basicProperties: properties,
            body: body
        );

        message.Set("published", true);
        message.Set("messageId", properties.MessageId);
        message.Set("queue", this.Settings.QueueName);
    }
    catch (Exception ex)
    {
        Error(ex, "Failed to publish message to RabbitMQ");
        var error = $"Publish failed: {ex.Message}";
        this.SetStatus(Status.Warning, error);
        message.SetError(error);
    }
    finally
    {
        // Always forward the message
        await this.Next(message);
    }
}

public override async Task Stop()
```

```
    {
        try
        {
            this.channel?.Close();
            this.channel?.Dispose();
            this.connection?.Close();
            this.connection?.Dispose();

            Information("Disconnected from RabbitMQ");
        }
        catch (Exception ex)
        {
            Warning(ex, "Error closing RabbitMQ connection");
        }

        await base.Stop();
    }
}

public class RabbitMqModuleSettings : FlowModuleSettings
{
    [Required]
    [Display(Name = "Host")]
    [Description("RabbitMQ server hostname")]
    public string Host { get; set; }

    [Range(1, 65535)]
    [Display(Name = "Port")]
    [DefaultValue(5672)]
    public int Port { get; set; } = 5672;

    [Display(Name = "Virtual Host")]
    [DefaultValue("/")]
    public string VirtualHost { get; set; } = "/";

    [Required]
    [Display(Name = "Queue Name")]
    [Description("Name of the queue to publish to")]
    public string QueueName { get; set; }

    [Display(Name = "Exchange")]
    [Description("Optional exchange name (leave empty for default)")]
    public string Exchange { get; set; }

    [Display(Name = "Routing Key")]
    [Description("Optional routing key (defaults to queue name)")]
```

```
    public string RoutingKey { get; set; }

    [Display(Name = "Durable Queue")]
    [Description("Queue survives broker restart")]
    [DefaultValue(true)]
    public bool DurableQueue { get; set; } = true;

    [Display(Name = "Persistent Messages")]
    [Description("Messages survive broker restart")]
    [DefaultValue(true)]
    public bool PersistentMessages { get; set; } = true;

    [JsonSchemaExtensionData("x-credential", "UsernamePassword")]
    [Display(Name = "Queue Credentials")]
    public Guid? QueueCredential { get; set; }
}
```

Resource File Handling

Text File Resources

```
public override async Task<IError?> Initialize()
{
    if (!this.Settings.ConfigFile.HasValue)
    {
        return new Error("Configuration file not specified");
    }

    // Get resource as string (for text files like JSON, XML, CSV)
    var resource = await this.GetResourceContentAsync<string>(
        this.Settings.ConfigFile.Value);

    if (!resource.IsSuccess)
    {
        return new Error("Failed to load configuration file");
    }

    var fileContent = resource.Result;

    if (string.IsNullOrEmpty(fileContent))
    {
        return new Error("Configuration file is empty");
    }

    // Parse based on format
```

```
    try
    {
        if (this.IsJson(fileContent))
        {
            this.configuration = JsonSerializer.Deserialize<ModuleConfiguration>
(fileContent);
        }
        else if (this.IsXml(fileContent))
        {
            var doc = new XmlDocument();
            doc.LoadXml(fileContent);
            this.ProcessXmlDocument(doc);
        }
        else
        {
            return new Error("Unsupported file format");
        }
    }
    catch (Exception ex)
    {
        return new Error($"Failed to parse configuration file: {ex.Message}");
    }

    return await base.Initialize();
}

private bool IsJson(string content)
{
    content = content.Trim();
    return (content.StartsWith("{") && content.EndsWith("}")) ||
           (content.StartsWith("[") && content.EndsWith("]"));
}

private bool IsXml(string content)
{
    content = content.Trim();
    return content.StartsWith("<?xml") || content.StartsWith("<");
}
```

Binary File Resources

```
public override async Task<IError?> Initialize()
{
    if (!this.Settings.DataFile.HasValue)
    {
```

```
            return new Error("Data file not specified");
        }

        // Get resource as byte array (for binary files)
        var resource = await this.GetResourceContentAsync<byte[]>(
            this.Settings.DataFile.Value);

        if (!resource.IsSuccess)
        {
            return new Error("Failed to load data file");
        }

        var resourceBytes = resource.Result;

        if (resourceBytes == null || resourceBytes.Length == 0)
        {
            return new Error("Data file is empty");
        }

        // Process binary data
        try
        {
            // Example: Save to temp file for processing
            var tempPath = Path.Combine(Path.GetTempPath(),
                $"module_{Guid.NewGuid()}.dat");
            await File.WriteAllBytesAsync(tempPath, resourceBytes);

            // Process the file...
            this.ProcessBinaryFile(tempPath);

            // Clean up
            File.Delete(tempPath);
        }
        catch (Exception ex)
        {
            return new Error($"Failed to process data file: {ex.Message}");
        }

        return await base.Initialize();
    }
```

Integration Best Practices

1. Connection Health Checks

```
public override async Task<IError?> Start()
{
    // Start background health check
    this.healthCheckTimer = new Timer(
        async _ => await this.CheckConnectionHealth(),
        null,
        TimeSpan.FromMinutes(1),
        TimeSpan.FromMinutes(1)
    );

    return await base.Start();
}

private async Task CheckConnectionHealth()
{
    try
    {
        var response = await this.httpClient.GetAsync("/health");
        if (!response.IsSuccessStatusCode)
        {
            Warning("Health check failed: {StatusCode}", response.StatusCode);
            this.SetStatus(Status.Warning, "External service health check failed");
        }
    }
    catch (Exception ex)
    {
        Warning(ex, "Health check failed");
        this.SetStatus(Status.Warning, "Cannot reach external service");
    }
}
```

2. Retry Logic

See Error Handling for complete retry strategies.

3. Timeout Configuration

Always configure timeouts for external calls:

```
this.httpClient.Timeout = TimeSpan.FromSeconds(this.Settings.TimeoutSeconds);

command.CommandTimeout = this.Settings.QueryTimeout;
```

4. Secure Credential Handling

See Security Best Practices for complete guidance.

Summary

Key principles for external integrations:

1. Always use credentials - Never hardcode sensitive information
2. Test connections - Validate connectivity during Initialize()
3. Handle errors gracefully - Always forward messages, even on failure
4. Configure timeouts - Prevent hanging operations
5. Reuse connections - Initialize clients once, use many times
6. Monitor health - Implement health checks for long-running connections
7. Log appropriately - Log operations but never log credentials

For more details on specific topics:

- Security Best Practices - Secure credential handling
- Error Handling - Retry strategies and error patterns
- Performance Optimization - Connection pooling and efficiency

>> Logging

Logging

You can use the built-in static `ILog` instance to log messages at different levels (Verbose, Debug, Information, Warning, Error, Fatal). The logs will appear in the Crosser Node local log file when the flow is running.

```
using static Crosser.EdgeNode.Common.Utilities.Logger.Log
namespace YourOrg.FlowModule.RandomNumberModule;

public class RandomNumberModule  : FlowModule<RandomNumberModuleSettings>
{

    public RandomNumberModule () : base(FlowModuleType.Function) {}

    protected override async Task MessageReceived(IFlowMessage message)
    {
        Logger?.Debug($"Module received message: {message}");

        try
        {
            // Log input validation
            if (!ValidateInput(message))
            {
                Logger?.Warning($"Invalid input received: {message}");
                return;
            }

            // Log processing steps
            Logger?.Debug($"Processing message with settings: {Settings}");

            var result = await ProcessMessage(message);

            Logger?.Debug($"Module produced output: {result}");

            await Send(result);

            Logger?.Information("Message processed successfully");
        }
        catch (Exception ex)
        {
            Logger?.Error(ex, $"Failed to process message");
            throw;
        }
    }
```

```
    private bool ValidateInput(IFlowMessage message)
    {
        // Add input validation logic
        // Log validation failures
        return true;
    }
}
```

See the ILog interface for a full list of methods

> WARNING
>
> Be careful with what level you log at. You do not want to log Verbose data into higher levels such as Information.

Namespace Crosser.EdgeNode.Flows

Classes

BaseSchemaExtensionAttribute

CredentialTypeAttribute
: This property will be flagged as credential. Should of type Guid or Guid?. The attribute value should be a comma separated string with the credential types the setting accept.

Flow

FlowError

FlowError<T>

FlowException

FlowMessagePersistence

FlowModule
: Base class for all modules, but you will probably use FlowModule<TSettings> for your implementation

FlowModuleJsonConverter

FlowModuleMetadata

FlowModuleWithoutSettings
: Use this when a module have no specific settings. This will populate the module with the default settings for filters etc

FlowModule<TSettings>
: Base class for all modules with settings.

KeyDescriptionAttribute
: Used for dictionaries The value is used as a placeholder in the input field for new entry keys

MethodInformation

ModuleHelpers

ModuleInfo

ModuleInteractivity

PathHelpers

Contains paths that might be useful when writing custom modules

ResourceTypeAttribute

This property will be flagged as resource. Should of type Guid or Guid?. The attribute value should be a comma separated string with the resource types the setting accept

SettingInteractivity

TaskHelpers

UIEditorAttribute

UISortOrderAttribute

What index the setting should be displayed in the UI. The setting with the lowest index will be displayed first (negative is allowed). Settings with no sort order will be placed last in the order they are defined in the C# module settings class

ValueDescriptionAttribute

Used for dictionaries and lists of simple types. The value is used as a placeholder in the input field for new entry values

Enums

FlowErrorCode

FlowMessageRetry

FlowModuleStatistics

FlowModuleType

FlowStatistics

QueueState

Status

Class BaseSchemaExtensionAttribute

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public abstract class BaseSchemaExtensionAttribute : Attribute
```

Inheritance

object ← Attribute ← BaseSchemaExtensionAttribute

Derived

CredentialTypeAttribute, KeyDescriptionAttribute, ResourceTypeAttribute, UIEditorAttribute, UISortOrderAttribute, ValueDescriptionAttribute

Inherited Members

Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) , Attribute.GetCustomAttribute(Assembly, Type, bool) , Attribute.GetCustomAttribute(MemberInfo, Type) , Attribute.GetCustomAttribute(MemberInfo, Type, bool) , Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) , Attribute.GetCustomAttribute(ParameterInfo, Type) , Attribute.GetCustomAttribute(ParameterInfo, Type, bool) , Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) , Attribute.GetCustomAttributes(Assembly, Type) , Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) , Attribute.GetCustomAttributes(MemberInfo, bool) , Attribute.GetCustomAttributes(MemberInfo, Type) , Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) , Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) , Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) , Attribute.GetCustomAttributes(ParameterInfo, bool) , Attribute.GetCustomAttributes(ParameterInfo, Type) , Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() , Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) , Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) , Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) , Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) , Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

BaseSchemaExtensionAttribute(string, object)

```
protected BaseSchemaExtensionAttribute(string name, object value)
```

Parameters

`name` string

`value` object

Properties

Name

```
public string Name { get; }
```

Property Value

string

Value

```
public object Value { get; }
```

Property Value

object

Class CredentialTypeAttribute

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

This property will be flagged as credential. Should of type Guid or Guid?. The attribute value should be a comma separated string with the credential types the setting accept.

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class CredentialTypeAttribute : BaseSchemaExtensionAttribute
```

Inheritance

object ← Attribute ← BaseSchemaExtensionAttribute ← CredentialTypeAttribute

Inherited Members

BaseSchemaExtensionAttribute.Name , BaseSchemaExtensionAttribute.Value , Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) , Attribute.GetCustomAttribute(Assembly, Type, bool) , Attribute.GetCustomAttribute(MemberInfo, Type) , Attribute.GetCustomAttribute(MemberInfo, Type, bool) , Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) , Attribute.GetCustomAttribute(ParameterInfo, Type) , Attribute.GetCustomAttribute(ParameterInfo, Type, bool) , Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) , Attribute.GetCustomAttributes(Assembly, Type) , Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) , Attribute.GetCustomAttributes(MemberInfo, bool) , Attribute.GetCustomAttributes(MemberInfo, Type) , Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) , Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) , Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) , Attribute.GetCustomAttributes(ParameterInfo, bool) , Attribute.GetCustomAttributes(ParameterInfo, Type) , Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() , Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) , Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) , Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) , Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) , Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CredentialTypeAttribute(string)

```
public CredentialTypeAttribute(string type)
```

Parameters

`type` string

Class Flow

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public class Flow : FlowBase, IDisposable
```

Inheritance

object ← FlowBase ← Flow

Implements

IDisposable

Inherited Members

FlowBase.Name , FlowBase.FlowVersion , FlowBase.Description , FlowBase.Id , FlowBase.FlowDefinitionId , FlowBase.FlowDeploymentId , FlowBase.ResolveFlowDeploymentId , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

Flow()

Default constructor

```
public Flow()
```

Properties

CalculateTotalTimeoutForStartup

```
public int CalculateTotalTimeoutForStartup { get; }
```

Property Value

int

DisableQueues

If true no channels will be used for queueing messages internally in the module Persistence and retry will also be disabled if queues are disabled

```
public bool DisableQueues { get; set; }
```

Property Value

bool

EnabledModules

```
[JsonIgnore]
public IEnumerable<FlowModule> EnabledModules { get; }
```

Property Value

IEnumerable<FlowModule>

FlowContext

Context for the flow

```
[JsonIgnore]
public RepositoryInstance<string, dynamic> FlowContext { get; }
```

Property Value

RepositoryInstance<string, dynamic>

FlowState

Current FlowState of the flow

```
public virtual FlowState FlowState { get; set; }
```

Property Value

FlowState

HaltOnError

If true the flow will be stopped if an error occurs

```
public bool HaltOnError { get; set; }
```

Property Value

bool

IsCancellationRequested

```
public bool IsCancellationRequested { get; }
```

Property Value

bool

Modules

```
[JsonConverter(typeof(FlowModuleJsonConverter))]
public List<FlowModule> Modules { get; set; }
```

Property Value

List<FlowModule>

OnFlowException

```
[JsonIgnore]
public EventHandler<FlowException>? OnFlowException { get; set; }
```

Property Value

EventHandler<FlowException>

OnMessageDeadLetter

```
[JsonIgnore]
public EventHandler<MessageEvent>? OnMessageDeadLetter { get; set; }
```

Property Value

EventHandler<MessageEvent>

OnMessageDropped

```
[JsonIgnore]
public EventHandler<MessageEvent>? OnMessageDropped { get; set; }
```

Property Value

EventHandler<MessageEvent>

OnModuleDebugMessage

```
[JsonIgnore]
public EventHandler<ModuleDebugMessage>? OnModuleDebugMessage { get; set; }
```

Property Value

EventHandler<ModuleDebugMessage>

OnModuleStatusChanged

```
[JsonIgnore]
public EventHandler<ModuleStatusEvent>? OnModuleStatusChanged { get; set; }
```

Property Value

EventHandler<ModuleStatusEvent>

RemoteSessionEnabled

True when debug data should be sent to cloud

```
[JsonIgnore]
public bool RemoteSessionEnabled { get; }
```

Property Value

bool

Resources

The resources used on the flow

```
public List<ResourceBase> Resources { get; set; }
```

Property Value

List<ResourceBase>

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Dispose(bool)

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

EnableRemoteSession()

```
public void EnableRemoteSession()
```

Start()

```
public Task<IResult> Start()
```

Returns

Task<IResult>

StartInteractive()

```
public Task<IResult> StartInteractive()
```

Returns

Task<IResult>

Stop()

```
public Task<IResult> Stop()
```

Returns

Task<IResult>

ValidateFlow()

```
public IResult<IList<IValidationError>> ValidateFlow()
```

Returns

IResult<IList<IValidationError>>

Class FlowError

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public record FlowError : IEquatable<FlowError>
```

Inheritance

object ← FlowError

Implements

IEquatable<FlowError>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowError(object?, FlowErrorCode, string?)

```
public FlowError(object? data, FlowErrorCode errorCode, string? message)
```

Parameters

`data` object

`errorCode` FlowErrorCode

`message` string

Properties

ErrorText

```
public string ErrorText { get; }
```

Property Value

string

data

```
public object? data { get; init; }
```

Property Value

object

errorCode

```
public FlowErrorCode errorCode { get; init; }
```

Property Value

FlowErrorCode

message

```
public string? message { get; init; }
```

Property Value

string

Enum FlowErrorCode

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
[JsonConverter(typeof(JsonStringEnumConverter))]
public enum FlowErrorCode
```

Fields

BadRequest = 3

GenericError = 0

InvalidOperation = 4

MethodNotFound = 2

ModuleException = 6

ModuleValidationError = 7

ProcessError = 5

UnknownError = -1

UriNotFound = 1

Class FlowError<T>

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public record FlowError<T> : IEquatable<FlowError<T>>
```

Type Parameters

`T`

Inheritance

object ← FlowError<T>

Implements

IEquatable<FlowError<T>>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowError(T?, FlowErrorCode, string?)

```
public FlowError(T? data, FlowErrorCode errorCode, string? message)
```

Parameters

`data` T

`errorCode` FlowErrorCode

`message` string

Properties

ErrorText

```
public string ErrorText { get; }
```

Property Value

string

data

```
public T? data { get; init; }
```

Property Value

T

errorCode

```
public FlowErrorCode errorCode { get; init; }
```

Property Value

FlowErrorCode

message

```
public string? message { get; init; }
```

Property Value

string

Class FlowException

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public class FlowException
```

Inheritance

object ← FlowException

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowException(Guid, Exception)

```
public FlowException(Guid flowId, Exception exception)
```

Parameters

`flowId` Guid

`exception` Exception

FlowException(Guid, Exception, IFlowModule)

```
public FlowException(Guid flowId, Exception exception, IFlowModule module)
```

Parameters

`flowId` Guid

`exception` Exception

module IFlowModule

Properties

Exception

```
public Exception Exception { get; set; }
```

Property Value

Exception

FlowId

```
public Guid FlowId { get; set; }
```

Property Value

Guid

Module

```
public IFlowModule? Module { get; set; }
```

Property Value

IFlowModule

Class FlowMessagePersistence

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public class FlowMessagePersistence
```

Inheritance

object ← FlowMessagePersistence

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowMessagePersistence(FlowModule)

```
public FlowMessagePersistence(FlowModule module)
```

Parameters

`module` FlowModule

Methods

CreatePersistentMessage(IFlowMessage)

```
public Task<IResult> CreatePersistentMessage(IFlowMessage message)
```

Parameters

`message` IFlowMessage

Returns

Task<IResult>

DeleteFromStorage(Guid)

```
public Task<IResult> DeleteFromStorage(Guid messageId)
```

Parameters

`messageId` Guid

Returns

Task<IResult>

GetAllMessagesForModule()

```
public Task<IResult<IEnumerable<PersistentMessage>>> GetAllMessagesForModule()
```

Returns

Task<IResult<IEnumerable<PersistentMessage>>>

TryAgain(Guid)

```
public Task<FlowMessageRetry> TryAgain(Guid messageId)
```

Parameters

`messageId` Guid

Returns

Task<FlowMessageRetry>

Enum FlowMessageRetry

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public enum FlowMessageRetry
```

Fields

`No = 0`

`Yes = 1`

Class FlowModule

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

Base class for all modules, but you will probably use FlowModule<TSettings> for your implementation

```
public abstract class FlowModule : IFlowModule, IDisposable
```

Inheritance

object ← FlowModule

Implements

IFlowModule, IDisposable

Derived

FlowModule<TSettings>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowModule(FlowModuleType)

Constructor for FlowModule

```
protected FlowModule(FlowModuleType moduleType)
```

Parameters

`moduleType` FlowModuleType

Fields

InteractivityJsonOptions

```
public static readonly JsonSerializerOptions InteractivityJsonOptions
```

Field Value

JsonSerializerOptions

Properties

AssemblyVersion

```
public string AssemblyVersion { get; set; }
```

Property Value

string

Categories

Sets the tags to be able to group modules together representing similar functionality

```
[JsonIgnore]
public List<string> Categories { get; set; }
```

Property Value

List<string>

Debug

If true the module will log messages in/out from the module.

```
public bool Debug { get; set; }
```

Property Value

bool

Description

```
[JsonIgnore]
public string Description { get; }
```

Property Value

string

Disabled

If true the module will ignore messages coming in

```
public bool Disabled { get; set; }
```

Property Value

bool

FlowContext

Reference to the flow specific context that the module belongs to

```
[JsonIgnore]
public RepositoryInstance<string, dynamic> FlowContext { get; }
```

Property Value

RepositoryInstance<string, dynamic>

FlowId

Id of the flow that the module belongs to

```
public Guid FlowId { get; set; }
```

Property Value

Guid

FlowName

```
public string FlowName { get; }
```

Property Value

string

FlowStatistics

```
[Obsolete("This property will be removed in the next major version")]
[JsonIgnore]
public FlowStatistics FlowStatistics { get; set; }
```

Property Value

FlowStatistics

HaltOnError

If true the module and flow will be stopped when OnError is called

```
public bool HaltOnError { get; set; }
```

Property Value

bool

HasInput

If true you can pass data to this module

```
public bool HasInput { get; set; }
```

Property Value

bool

HasInteractiveSupport

```
public virtual bool HasInteractiveSupport { get; }
```

Property Value

bool

HasOutput

if Outputs Count is > 0 this will be true

```
public bool HasOutput { get; }
```

Property Value

bool

Id

Unique Id for the module instance

```
public Guid Id { get; set; }
```

Property Value

Guid

Index

```
[JsonIgnore]
public int Index { get; set; }
```

Property Value

int

InteractiveMethods

Get all currently available interactive methods from the module. Can be overridden in derived classes i.e. lets derived class have another way of storing methods.

```
public virtual IEnumerable<KeyValuePair<(string Uri, string Method),
Func<JsonElement, ILog?, Task<(object?, FlowError?)>>>> InteractiveMethods { get; }
```

Property Value

IEnumerable<KeyValuePair<(string Uri, string Method), Func<JsonElement, ILog, Task<(object, FlowError)>>>>

InteractivityTopics

Get all registered methods from the module

```
public IEnumerable<(string Uri, string Method)> InteractivityTopics { get; }
```

Property Value

IEnumerable<(string Uri, string Method)>

Logger

```
public ILog? Logger { get; }
```

Property Value

ILog

MaxNrOfOutputs

The maximum number of outputs for the module, defaults to 1 so set this in your custom module if needed

```
[Obsolete("This property will be removed in the next major version")]
public int MaxNrOfOutputs { get; set; }
```

Property Value

int

ModuleProtocol

If the module uses a specific protocol for communication is should be registered here to make routers able to locate the module

```
[Obsolete("Modules that communicate via a specific protocol will solve the
communication on their own")]
[JsonIgnore]
public FlowModuleProtocol ModuleProtocol { get; set; }
```

Property Value

FlowModuleProtocol

ModuleStatistics

```
[Obsolete("This property will be removed in the next major version")]
[JsonIgnore]
```

```
public FlowModuleStatistics ModuleStatistics { get; set; }
```

Property Value

FlowModuleStatistics

ModuleType

The module type implemented. See FlowModuleType

```
[JsonIgnore]
public FlowModuleType ModuleType { get; set; }
```

Property Value

FlowModuleType

ModuleVersion

```
public string ModuleVersion { get; set; }
```

Property Value

string

Name

Name of the module at runtime (set by users using the module in the flow)

```
public string Name { get; set; }
```

Property Value

string

NoError

```
protected static Task<IError?> NoError { get; }
```

Property Value

Task<IError>

NrOfOutputs

The available number of outputs from the module

```
public int NrOfOutputs { get; }
```

Property Value

int

Remarks

Each output can have several modules connected to it

OnDebug

```
[JsonIgnore]
public EventHandler<ModuleDebugMessage>? OnDebug { get; set; }
```

Property Value

EventHandler<ModuleDebugMessage>

OnError

```
[Obsolete("Use SetStatus instead")]
[JsonIgnore]
public EventHandler<FlowModuleException>? OnError { get; set; }
```

Property Value

EventHandler<FlowModuleException>

OnFlowDataReceived

```
[JsonIgnore]
[Obsolete("This will be removed in the next major version")]
public EventHandler<FlowStatisticsData>? OnFlowDataReceived { get; set; }
```

Property Value

EventHandler<FlowStatisticsData>

OnFlowDataSent

```
[JsonIgnore]
[Obsolete("This will be removed in the next major version")]
public EventHandler<FlowStatisticsData>? OnFlowDataSent { get; set; }
```

Property Value

EventHandler<FlowStatisticsData>

OnMessageDeadLetter

```
[JsonIgnore]
public EventHandler<MessageEvent>? OnMessageDeadLetter { get; set; }
```

Property Value

EventHandler<MessageEvent>

OnMessageDropped

```
[JsonIgnore]
public EventHandler<MessageEvent>? OnMessageDropped { get; set; }
```

Property Value

EventHandler<MessageEvent>

OnModuleDataReceived

```
[JsonIgnore]
[Obsolete("This will be removed in the next major version")]
public EventHandler<FlowModuleStatisticsData>? OnModuleDataReceived { get; set; }
```

Property Value

EventHandler<FlowModuleStatisticsData>

OnModuleDataSent

```
[JsonIgnore]
[Obsolete("This will be removed in the next major version")]
public EventHandler<FlowModuleStatisticsData>? OnModuleDataSent { get; set; }
```

Property Value

EventHandler<FlowModuleStatisticsData>

OnStatusChanged

```
[JsonIgnore]
public EventHandler<ModuleStatusEvent>? OnStatusChanged { get; set; }
```

Property Value

EventHandler<ModuleStatusEvent>

OnWarning

```
[Obsolete("Use SetStatus instead")]
[JsonIgnore]
public EventHandler<FlowModuleException>? OnWarning { get; set; }
```

Property Value

EventHandler<FlowModuleException>

Outputs

The actual outputs from the module

```
public List<RepositoryInstance<Guid, IFlowModule?>> Outputs { get; set; }
```

Property Value

List<RepositoryInstance<Guid, IFlowModule>>

ReceivesInputFromMultipleModules

```
public bool ReceivesInputFromMultipleModules { get; set; }
```

Property Value

bool

RemoteSessionEnabled

```
[JsonIgnore]
public bool RemoteSessionEnabled { get; set; }
```

Property Value

bool

Resources

```
public IList<ResourceBase> Resources { get; set; }
```

Property Value

IList<ResourceBase>

Settings

```
public FlowModuleSettings Settings { get; protected set; }
```

Property Value

FlowModuleSettings

SettingsUri

```
public string SettingsUri { get; }
```

Property Value

string

Status

```
[JsonIgnore]
public Status Status { get; }
```

Property Value

Status

Topic

To get the topic for routing modules. This setting should be refactored with the ModuleType later on.

```
[Obsolete("Modules that communicate via a specific protocol will solve the
communication on their own")]
[JsonIgnore]
public string? Topic { get; set; }
```

Property Value

string

Type

```
[JsonIgnore]
public string Type { get; }
```

Property Value

string

Uri

```
public string Uri { get; }
```

Property Value

string

UserFriendlyName

Name to present to users

```
public abstract string UserFriendlyName { get; }
```

Property Value

string

Methods

AddMethod<TParameters, TResult>(string, Func<TParameters, Task<(TResult, FlowError?)>>)

```
protected void AddMethod<TParameters, TResult>(string name, Func<TParameters,
Task<(TResult, FlowError?)>> method)
```

Parameters

`name` string

`method` Func<TParameters, Task<(TResult, FlowError)>>

Type Parameters

`TParameters`

`TResult`

AddMethod<TParameters, TResult>(string, string, Func<TParameters, Task<(TResult, FlowError?)>>)

```
protected void AddMethod<TParameters, TResult>(string uri, string name,
Func<TParameters, Task<(TResult, FlowError?)>> method)
```

Parameters

`uri` string

`name` string

`method` Func<TParameters, Task<(TResult, FlowError)>>

Type Parameters

`TParameters`

`TResult`

AddModuleMethod<TParameters, TResult>(string, Func<TParameters, Task<(TResult, FlowError?)>>)

```
protected void AddModuleMethod<TParameters, TResult>(string name, Func<TParameters,
Task<(TResult, FlowError?)>> method)
```

Parameters

`name` string

`method` Func<TParameters, Task<(TResult, FlowError)>>

Type Parameters

`TParameters`

`TResult`

AddSettingMethod<TParameters, TResult>(string, Func<TParameters, Task<(TResult, FlowError?)>>)

```
protected void AddSettingMethod<TParameters, TResult>(string name, Func<TParameters,
Task<(TResult, FlowError?)>> method)
```

Parameters

`name` string

`method` Func<TParameters, Task<(TResult, FlowError)>>

Type Parameters

`TParameters`

`TResult`

AllowMessageToEnterModule(IFlowMessage)

```
protected abstract bool AllowMessageToEnterModule(IFlowMessage message)
```

Parameters

`message` IFlowMessage

Returns

bool

BypassMessageNotMatchingRules()

```
protected abstract bool BypassMessageNotMatchingRules()
```

Returns

bool

CanStart()

```
[Obsolete("This method will be removed in the next major version")]
public virtual IError? CanStart()
```

Returns

IError

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public virtual void Dispose()
```

Dispose(bool)

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

FlowDefinitionId()

```
public Guid FlowDefinitionId()
```

Returns

Guid

GetCredential(Guid)

This will return the Credential loaded into the Settings.Credential dictionary. If you need to download the resource use the IFlowCredentialsManagerService

```
[Obsolete("Use GetCredentialAsync instead")]
public IResult<Credential> GetCredential(Guid credentialId)
```

Parameters

`credentialId` Guid

Returns

IResult<Credential>

GetCredentialAsync(Guid)

If the credential is not found locally it will be downloaded. This call need all required services to be registered in the ServiceCollection. This is done by the runtime, but if you are using the method in a unit test you need to handle these requirements.

```
public Task<IResult<Credential>> GetCredentialAsync(Guid credentialId)
```

Parameters

`credentialId` Guid

Returns

Task<IResult<Credential>>

GetCredentialContentAs<T>(Guid)

This will return the Credential loaded into the Settings.Credential dictionary. If you need to download the resource use the IFlowCredentialsManagerService

```
[Obsolete("Use GetCredentialContentAsync instead")]
public IResult<T> GetCredentialContentAs<T>(Guid credentialId) where T :
class, new()
```

Parameters

`credentialId` Guid

Returns

IResult<T>

Type Parameters

`T`

Exceptions

NullReferenceException

GetCredentialContentAsync<T>(Guid)

If the credential is not found locally it will be downloaded. This call need all required services to be registered in the ServiceCollection. This is done by the runtime, but if you are using the method in a unit test you need to handle these requirements.

```
public Task<IResult<T>> GetCredentialContentAsync<T>(Guid credentialId) where T :
class, new()
```

Parameters

`credentialId` Guid

Returns

Task<IResult<T>>

Type Parameters

`T`

Exceptions

NullReferenceException

GetResource(Guid)

This will return the ResourceItem loaded into the Settings.Resources dictionary. If you need to download the resource use the IFlowResourceManagerService

```
[Obsolete("Use GetResourceAsync instead")]
public IResult<ResourceItem> GetResource(Guid resourceId)
```

Parameters

`resourceId` Guid

Returns

IResult<ResourceItem>

GetResourceAsync(Guid)

If the resource is not found locally it will be downloaded. This call need all required services to be registered in the ServiceCollection. This is done by the runtime, but if you are using the method in a unit test you need to handle these requirements.

```
public Task<IResult<ResourceItem>> GetResourceAsync(Guid resourceId)
```

Parameters

`resourceId` Guid

Returns

Task<IResult<ResourceItem>>

GetResourceContentAs<T>(Guid)

This will return the ResourceItem loaded into the Settings.Resources dictionary. If you need to download the resource use the IFlowResourceManagerService

```
[Obsolete("Use GetResourceContentAsync instead")]
public IResult<T> GetResourceContentAs<T>(Guid resourceId) where T : class, new()
```

Parameters

`resourceId` Guid

Returns

IResult<T>

Type Parameters

`T`

Exceptions

NullReferenceException

GetResourceContentAsync<T>(Guid)

If the resource is not found locally it will be downloaded. This call need all required services to be registered in the ServiceCollection. This is done by the runtime, but if you are using the method in a unit test you need to handle these requirements.

```
public Task<IResult<T>> GetResourceContentAsync<T>(Guid resourceId) where T :
class, new()
```

Parameters

`resourceId` Guid

Returns

Task <IResult<T>>

Type Parameters

`T`

Exceptions

NullReferenceException

Initialize()

This method will be called when the flow is about to be started, but before the call to Start()

```
public virtual Task<IError?> Initialize()
```

Returns

Task<IError>

An awaitable Task

InitializeInteractivity()

Override this method and register your module interactivity methods

```
public virtual void InitializeInteractivity()
```

InteractiveSessionRequest(string, string, JsonElement)

```
public Task<(object? result, FlowError? error)> InteractiveSessionRequest(string
uri, string method, JsonElement parameters)
```

Parameters

`uri` string

`method` string

`parameters` JsonElement

Returns

Task<(object result, FlowError error)>

InteractiveSessionUri(string)

Obsolete method to get the interactive session URI. This method is deprecated and should be replaced with Uri and interpolated string. Uri

```
[Obsolete("Use Uri and interpolated string instead")]
public string InteractiveSessionUri(string uri)
```

Parameters

`uri` string

Returns

string

Interactivity()

```
public virtual ModuleInteractivity Interactivity()
```

Returns

ModuleInteractivity

LoadSettings(string)

```
public abstract void LoadSettings(string settingsAsJson)
```

Parameters

`settingsAsJson` string

MessageReceived(IFlowMessage)

Implement this method in your custom modules to implement logic for the FlowMessage passing through the module. If you want to pass data to the next IFlowModule you should call Next(params IFlowMessage[]) in this method

```
protected abstract Task MessageReceived(IFlowMessage message)
```

Parameters

`message` IFlowMessage

The FlowMessage passed into the module

Returns

Task

An awaitable Task

Next(params IFlowMessage[])

When you want to pass the message(s) to the next module(s) in the. This should be called from the MessageReceived(IFlowMessage) method.

```
protected Task Next(params IFlowMessage[] messages)
```

Parameters

`messages` IFlowMessage[]

The parameter is 0 to n FlowMessage"

Returns

Task

An awaitable Task

Receive(IFlowMessage)

Passing data to this will flag the module to process the message. In most cases this will be done by the flow-engine or the flow but in special cases you can call this manually to trigger the processing of a message

```
public Task Receive(IFlowMessage message)
```

Parameters

`message` IFlowMessage

The FlowMessage passed into the module

Returns

Task

An awaitable Task

ReceiveFrom(IFlowModule, int)

Will register the module passed in as a receiver from this module. The module will will be registered at output index 0 by default

```
public IFlowModule ReceiveFrom(IFlowModule module, int ix = 0)
```

Parameters

`module` IFlowModule

The module to receive data

`ix` int

The output-index to register the module on

Returns

IFlowModule

SendTo(IFlowModule, int)

Will pass messages this module to the module passed in

```
public IFlowModule SendTo(IFlowModule module, int ix = 0)
```

Parameters

`module` IFlowModule

The module that will receive the data

`ix` int

The output index to add the module to

Returns

IFlowModule

The module that will receive the data (the module passed in)

SetNrOfOutputs(int)

```
protected void SetNrOfOutputs(int value)
```

Parameters

`value` int

SetStatus(Status)

Set the Status of the module. If the current status is different the OnStatusChanged event will be triggered.

```
public void SetStatus(Status status)
```

Parameters

`status` Status

The new Status

SetStatus(Status, string)

Set the Status of the module. If the current status is different the OnStatusChanged event will be triggered.

```
public void SetStatus(Status status, string message = "")
```

Parameters

`status` Status

The new Status

`message` string

The message to attach to the ModuleStatusEvent

SetStatus(Status, string, bool)

```
public void SetStatus(Status status, string message = "", bool force = false)
```

Parameters

`status` Status

`message` string

`force` bool

SetStatus(Status, string, bool, Exception?)

Set the Status of the module. If the current status is different the OnStatusChanged event will be triggered.

```
public void SetStatus(Status status, string message = "", bool force = false,
Exception? exception = null)
```

Parameters

`status` Status

The new Status

`message` string

The message to attach to the ModuleStatusEvent

`force` bool

Pass true to force the OnStatusChanged event to be triggered regardless of current status

`exception` Exception

Start()

This method will be called when the flow is about to be started, but after the call to Initialize()

```
public virtual Task<IError?> Start()
```

Returns

Task<IError>

A Crosser.EdgeNode.Common.Abstractions.Utilities.Errors.IError describing the error. If null there was no error.

Stop()

This method will be called when the flow is about to be stopped.

```
public virtual Task Stop()
```

Returns

Task

An awaitable Task

UpdateFlowReceivedStatistics(long)

```
[Obsolete("This method will be removed in the next major version")]
protected void UpdateFlowReceivedStatistics(long size = 0)
```

Parameters

`size` long

UpdateFlowSentStatistics(long)

```
[Obsolete("This method will be removed in the next major version")]
protected void UpdateFlowSentStatistics(long size = 0)
```

Parameters

`size` long

UpdateModuleReceivedStatistics(long)

```
[Obsolete("This method will be removed in the next major version")]
protected void UpdateModuleReceivedStatistics(long size = 0)
```

Parameters

`size` long

UpdateModuleSentStatistics(long)

```
[Obsolete("This method will be removed in the next major version")]
protected void UpdateModuleSentStatistics(long size = 0)
```

Parameters

`size` long

ValidateSettings(SettingsValidator)

```
public virtual void ValidateSettings(SettingsValidator validator)
```

Parameters

`validator` SettingsValidator

Class FlowModuleJsonConverter

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public class FlowModuleJsonConverter : JsonConverter<List<FlowModule>>
```

Inheritance

object ← JsonConverter ← JsonConverter<List<FlowModule>> ← FlowModuleJsonConverter

Inherited Members

JsonConverter<List<FlowModule>>.CanConvert(Type) ,
JsonConverter<List<FlowModule>>.ReadAsPropertyName(ref Utf8JsonReader, Type, JsonSerializerOptions) ,
JsonConverter<List<FlowModule>>.WriteAsPropertyName(Utf8JsonWriter, List<FlowModule>, JsonSerializerOptions) ,
JsonConverter<List<FlowModule>>.HandleNull , JsonConverter<List<FlowModule>>.Type ,
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetFlowModuleMetadata(JsonElement)

```
public static FlowModuleMetadata GetFlowModuleMetadata(JsonElement jsonElement)
```

Parameters

`jsonElement` JsonElement

Returns

FlowModuleMetadata

Read(ref Utf8JsonReader, Type, JsonSerializerOptions)

Reads and converts the JSON to type List<FlowModule>.

```
public override List<FlowModule> Read(ref Utf8JsonReader reader, Type typeToConvert,
JsonSerializerOptions options)
```

Parameters

`reader` Utf8JsonReader

The reader.

`typeToConvert` Type

The type to convert.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Returns

List<FlowModule>

The converted value.

Write(Utf8JsonWriter, List<FlowModule>, JsonSerializerOptions)

Writes a specified value as JSON.

```
public override void Write(Utf8JsonWriter writer, List<FlowModule> value,
JsonSerializerOptions options)
```

Parameters

`writer` Utf8JsonWriter

The writer to write to.

`value` List<FlowModule>

The value to convert to JSON.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Class FlowModuleMetadata

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public record FlowModuleMetadata : IEquatable<FlowModuleMetadata>
```

Inheritance

object ← FlowModuleMetadata

Implements

IEquatable<FlowModuleMetadata>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowModuleMetadata(string, bool, bool, string, Guid, Guid, string, bool, bool, string, string?, string)

```
public FlowModuleMetadata(string Type, bool Debug, bool HaltOnError, string
ModuleVersion, Guid Id, Guid FlowId, string Outputs, bool HasInput, bool Disabled,
string Name, string? CustomName, string Settings)
```

Parameters

`Type` string

`Debug` bool

`HaltOnError` bool

`ModuleVersion` string

`Id` Guid

`FlowId` Guid

`Outputs` string

`HasInput` bool

`Disabled` bool

`Name` string

`CustomName` string

`Settings` string

Properties

CustomName

```
public string? CustomName { get; init; }
```

Property Value

string

Debug

```
public bool Debug { get; init; }
```

Property Value

bool

Disabled

```
public bool Disabled { get; init; }
```

Property Value

bool

FlowId

```
public Guid FlowId { get; init; }
```

Property Value

Guid

HaltOnError

```
public bool HaltOnError { get; init; }
```

Property Value

bool

HasInput

```
public bool HasInput { get; init; }
```

Property Value

bool

Id

```
public Guid Id { get; init; }
```

Property Value

Guid

ModuleVersion

```
public string ModuleVersion { get; init; }
```

Property Value

string

Name

```
public string Name { get; init; }
```

Property Value

string

Outputs

```
public string Outputs { get; init; }
```

Property Value

string

Settings

```
public string Settings { get; init; }
```

Property Value

string

Type

```
public string Type { get; init; }
```

Property Value

string

Enum FlowModuleStatistics

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
[Obsolete("Will be removed in the next major version")]
[Flags]
public enum FlowModuleStatistics
```

Fields

`In = 1`

`Off = 0`

`Out = 2`

Enum FlowModuleType

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public enum FlowModuleType
```

Fields

`Function = 4`

This type receives data and acts on it and pass it on to the next module in the flow

`Input = 1`

This type receives the first message or generate messages

`Output = 2`

This type receives the last message in the flow

Class FlowModuleWithoutSettings

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

Use this when a module have no specific settings. This will populate the module with the default settings for filters etc

```
public abstract class FlowModuleWithoutSettings : FlowModule<FlowModuleSettings>,
IFlowModule, IDisposable
```

Inheritance

object ← FlowModule ← FlowModule<FlowModuleSettings> ← FlowModuleWithoutSettings

Implements

IFlowModule, IDisposable

Inherited Members

FlowModule<FlowModuleSettings>.Settings , FlowModule<FlowModuleSettings>.LoadSettings(string) , FlowModule<FlowModuleSettings>.AllowMessageToEnterModule(IFlowMessage) , FlowModule<FlowModuleSettings>.BypassMessageNotMatchingRules() , FlowModule<FlowModuleSettings>.ValidateSettings(SettingsValidator) , FlowModule.Logger , FlowModule.ReceivesInputFromMultipleModules , FlowModule.SetNrOfOutputs(int) , FlowModule.Receive(IFlowMessage) , FlowModule.ReceiveFrom(IFlowModule, int) , FlowModule.SendTo(IFlowModule, int) , FlowModule.SetStatus(Status) , FlowModule.SetStatus(Status, string) , FlowModule.SetStatus(Status, string, bool) , FlowModule.SetStatus(Status, string, bool, Exception) , FlowModule.MessageReceived(IFlowMessage) , FlowModule.Next(params IFlowMessage[]) , FlowModule.NoError , FlowModule.Initialize() , FlowModule.Start() , FlowModule.Stop() , FlowModule.Dispose(bool) , FlowModule.Dispose() , FlowModule.CanStart() , FlowModule.UpdateModuleReceivedStatistics(long) , FlowModule.UpdateFlowReceivedStatistics(long) , FlowModule.UpdateModuleSentStatistics(long) , FlowModule.UpdateFlowSentStatistics(long) , FlowModule.OnStatusChanged , FlowModule.OnError , FlowModule.OnWarning , FlowModule.OnDebug , FlowModule.OnModuleDataReceived , FlowModule.OnFlowDataReceived , FlowModule.OnModuleDataSent , FlowModule.OnFlowDataSent , FlowModule.OnMessageDropped , FlowModule.OnMessageDeadLetter , FlowModule.GetResource(Guid) , FlowModule.GetResourceAsync(Guid) , FlowModule.GetCredential(Guid) , FlowModule.GetCredentialAsync(Guid) , FlowModule.GetResourceContentAs<T>(Guid) , FlowModule.GetResourceContentAsync<T>(Guid) , FlowModule.GetCredentialContentAs<T>(Guid) , FlowModule.GetCredentialContentAsync<T>(Guid) , FlowModule.Uri , FlowModule.SettingsUri , FlowModule.InteractivityJsonOptions , FlowModule.HasInteractiveSupport ,

FlowModule.InteractiveMethods , FlowModule.InteractivityTopics , FlowModule.InitializeInteractivity() , FlowModule.Interactivity() , FlowModule.InteractiveSessionRequest(string, string, JsonElement) , FlowModule.InteractiveSessionUri(string) , FlowModule.AddSettingMethod<TParameters, TResult>(string, Func<TParameters, Task<(TResult, FlowError)>>) , FlowModule.AddModuleMethod<TParameters, TResult>(string, Func<TParameters, Task<(TResult, FlowError)>>) , FlowModule.AddMethod<TParameters, TResult>(string, Func<TParameters, Task<(TResult, FlowError)>>) , FlowModule.AddMethod<TParameters, TResult>(string, string, Func<TParameters, Task<(TResult, FlowError)>>) , FlowModule.ModuleVersion , FlowModule.AssemblyVersion , FlowModule.Status , FlowModule.Resources , FlowModule.RemoteSessionEnabled , FlowModule.FlowContext , FlowModule.ModuleType , FlowModule.Type , FlowModule.Description , FlowModule.Name , FlowModule.UserFriendlyName , FlowModule.Categories , FlowModule.HasInput , FlowModule.HasOutput , FlowModule.HaltOnError , FlowModule.Disabled , FlowModule.NrOfOutputs , FlowModule.Index , FlowModule.Debug , FlowModule.Outputs , FlowModule.FlowId , FlowModule.FlowDefinitionId() , FlowModule.FlowName , FlowModule.Id , FlowModule.ModuleStatistics , FlowModule.FlowStatistics , FlowModule.MaxNrOfOutputs , FlowModule.ModuleProtocol , FlowModule.Topic , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowModuleWithoutSettings(FlowModuleType)

```
public FlowModuleWithoutSettings(FlowModuleType moduleType)
```

Parameters

`moduleType` FlowModuleType

Class FlowModule<TSettings>

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

Base class for all modules with settings.

```
public abstract class FlowModule<TSettings> : FlowModule, IFlowModule, IDisposable
where TSettings : FlowModuleSettings, new()
```

Type Parameters

`TSettings`

The class to use as settings for the module

Inheritance

object ← FlowModule ← FlowModule<TSettings>

Implements

IFlowModule, IDisposable

Derived

FlowModuleWithoutSettings

Inherited Members

FlowModule.Logger , FlowModule.ReceivesInputFromMultipleModules , FlowModule.SetNrOfOutputs(int) , FlowModule.Receive(IFlowMessage) , FlowModule.ReceiveFrom(IFlowModule, int) , FlowModule.SendTo(IFlowModule, int) , FlowModule.SetStatus(Status) , FlowModule.SetStatus(Status, string) , FlowModule.SetStatus(Status, string, bool) , FlowModule.SetStatus(Status, string, bool, Exception) , FlowModule.MessageReceived(IFlowMessage) , FlowModule.Next(params IFlowMessage[]) , FlowModule.NoError , FlowModule.Initialize() , FlowModule.Start() , FlowModule.Stop() , FlowModule.Dispose(bool) , FlowModule.Dispose() , FlowModule.CanStart() , FlowModule.UpdateModuleReceivedStatistics(long) , FlowModule.UpdateFlowReceivedStatistics(long) , FlowModule.UpdateModuleSentStatistics(long) , FlowModule.UpdateFlowSentStatistics(long) , FlowModule.OnStatusChanged , FlowModule.OnError , FlowModule.OnWarning , FlowModule.OnDebug , FlowModule.OnModuleDataReceived , FlowModule.OnFlowDataReceived , FlowModule.OnModuleDataSent , FlowModule.OnFlowDataSent , FlowModule.OnMessageDropped , FlowModule.OnMessageDeadLetter , FlowModule.GetResource(Guid) , FlowModule.GetResourceAsync(Guid) , FlowModule.GetCredential(Guid) ,

FlowModule.GetCredentialAsync(Guid) , FlowModule.GetResourceContentAs<T>(Guid) , FlowModule.GetResourceContentAsync<T>(Guid) , FlowModule.GetCredentialContentAs<T>(Guid) , FlowModule.GetCredentialContentAsync<T>(Guid) , FlowModule.Uri , FlowModule.SettingsUri , FlowModule.InteractivityJsonOptions , FlowModule.HasInteractiveSupport , FlowModule.InteractiveMethods , FlowModule.InteractivityTopics , FlowModule.InitializeInteractivity() , FlowModule.Interactivity() , FlowModule.InteractiveSessionRequest(string, string, JsonElement) , FlowModule.InteractiveSessionUri(string) , FlowModule.AddSettingMethod<TParameters, TResult>(string, Func<TParameters, Task<(TResult, FlowError)>>) , FlowModule.AddModuleMethod<TParameters, TResult>(string, Func<TParameters, Task<(TResult, FlowError)>>) , FlowModule.AddMethod<TParameters, TResult>(string, Func<TParameters, Task<(TResult, FlowError)>>) , FlowModule.AddMethod<TParameters, TResult>(string, string, Func<TParameters, Task<(TResult, FlowError)>>) , FlowModule.ModuleVersion , FlowModule.AssemblyVersion , FlowModule.Status , FlowModule.Resources , FlowModule.RemoteSessionEnabled , FlowModule.FlowContext , FlowModule.ModuleType , FlowModule.Type , FlowModule.Description , FlowModule.Name , FlowModule.UserFriendlyName , FlowModule.Categories , FlowModule.HasInput , FlowModule.HasOutput , FlowModule.HaltOnError , FlowModule.Disabled , FlowModule.NrOfOutputs , FlowModule.Index , FlowModule.Debug , FlowModule.Outputs , FlowModule.FlowId , FlowModule.FlowDefinitionId() , FlowModule.FlowName , FlowModule.Id , FlowModule.ModuleStatistics , FlowModule.FlowStatistics , FlowModule.MaxNrOfOutputs , FlowModule.ModuleProtocol , FlowModule.Topic , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowModule(FlowModuleType)

Constructor for FlowModule<TSettings>

```
public FlowModule(FlowModuleType moduleType)
```

Parameters

`moduleType` FlowModuleType

Properties

Settings

```
public TSettings Settings { get; protected set; }
```

Property Value

TSettings

Methods

AllowMessageToEnterModule(IFlowMessage)

Get the setting class for the module

```
protected override bool AllowMessageToEnterModule(IFlowMessage message)
```

Parameters

`message` IFlowMessage

Returns

bool

The settings class of the type passed in as generic FlowModule<TSettings>

BypassMessageNotMatchingRules()

```
protected override bool BypassMessageNotMatchingRules()
```

Returns

bool

LoadSettings(string)

```
public override sealed void LoadSettings(string settingsAsJson)
```

Parameters

`settingsAsJson` string

ValidateSettings(SettingsValidator)

```
public override sealed void ValidateSettings(SettingsValidator validator)
```

Parameters

`validator` SettingsValidator

Enum FlowStatistics

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
[Obsolete("Will be removed in the next major version")]
[Flags]
public enum FlowStatistics
```

Fields

`In = 1`

`Off = 0`

`Out = 2`

Class KeyDescriptionAttribute

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

Used for dictionaries The value is used as a placeholder in the input field for new entry keys

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class KeyDescriptionAttribute : BaseSchemaExtensionAttribute
```

Inheritance

object ← Attribute ← BaseSchemaExtensionAttribute ← KeyDescriptionAttribute

Inherited Members

BaseSchemaExtensionAttribute.Name , BaseSchemaExtensionAttribute.Value , Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) , Attribute.GetCustomAttribute(Assembly, Type, bool) , Attribute.GetCustomAttribute(MemberInfo, Type) , Attribute.GetCustomAttribute(MemberInfo, Type, bool) , Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) , Attribute.GetCustomAttribute(ParameterInfo, Type) , Attribute.GetCustomAttribute(ParameterInfo, Type, bool) , Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) , Attribute.GetCustomAttributes(Assembly, Type) , Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) , Attribute.GetCustomAttributes(MemberInfo, bool) , Attribute.GetCustomAttributes(MemberInfo, Type) , Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) , Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) , Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) , Attribute.GetCustomAttributes(ParameterInfo, bool) , Attribute.GetCustomAttributes(ParameterInfo, Type) , Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() , Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) , Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) , Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) , Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) , Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

KeyDescriptionAttribute(string)

```
public KeyDescriptionAttribute(string text)
```

Parameters

`text` string

Class MethodInformation

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public record MethodInformation : IEquatable<MethodInformation>
```

Inheritance

object ← MethodInformation

Implements

IEquatable<MethodInformation>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Description

```
public required string Description { get; init; }
```

Property Value

string

Id

```
public required string Id { get; init; }
```

Property Value

string

Type

```
public string Type { get; init; }
```

Property Value

string

Class ModuleHelpers

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public static class ModuleHelpers
```

Inheritance

object ← ModuleHelpers

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

FlowModuleVersionsCatalog

```
public static RepositoryInstance<string, ModuleInfo> FlowModuleVersionsCatalog {
get; set; }
```

Property Value

RepositoryInstance<string, ModuleInfo>

Methods

CreateInstance(TypeInfo)

```
public static IFlowModule? CreateInstance(TypeInfo type)
```

Parameters

`type` TypeInfo

Returns

IFlowModule

CreateInstance<T>(Type, FlowModuleMetadata)

```
public static T? CreateInstance<T>(Type type, FlowModuleMetadata moduleMetadata)
where T : IFlowModule
```

Parameters

`type` Type

`moduleMetadata` FlowModuleMetadata

Returns

T

Type Parameters

`T`

GetFlowModule(string)

```
public static IResult<ModuleInfo> GetFlowModule(string type)
```

Parameters

`type` string

Returns

IResult<ModuleInfo>

RegisterModuleType(ModuleInfo)

```
public static void RegisterModuleType(ModuleInfo moduleInfo)
```

Parameters

`moduleInfo` ModuleInfo

Class ModuleInfo

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public class ModuleInfo
```

Inheritance

object ← ModuleInfo

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ModuleInfo(Type)

```
public ModuleInfo(Type type)
```

Parameters

`type` Type

Properties

Type

```
public Type Type { get; set; }
```

Property Value

Type

TypeString

```
public string TypeString { get; set; }
```

Property Value

string

Class ModuleInteractivity

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public record ModuleInteractivity : IEquatable<ModuleInteractivity>
```

Inheritance

object ← ModuleInteractivity

Implements

IEquatable<ModuleInteractivity>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Id

```
public required Guid Id { get; set; }
```

Property Value

Guid

Methods

```
public required IEnumerable<MethodInformation> Methods { get; init; }
```

Property Value

IEnumerable<MethodInformation>

Settings

```
public required IEnumerable<SettingInteractivity> Settings { get; init; }
```

Property Value

IEnumerable <SettingInteractivity>

Uri

```
public required string Uri { get; init; }
```

Property Value

string

Class PathHelpers

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

Contains paths that might be useful when writing custom modules

```
public static class PathHelpers
```

Inheritance

object ← PathHelpers

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

FLOW_RESOURCE_DIRECTORY

```
public static string FLOW_RESOURCE_DIRECTORY { get; }
```

Property Value

string

Enum QueueState

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public enum QueueState
```

Fields

`Full = 1`

`Ok = 0`

Class ResourceTypeAttribute

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

This property will be flagged as resource. Should of type Guid or Guid?. The attribute value should be a comma separated string with the resource types the setting accept

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class ResourceTypeAttribute : BaseSchemaExtensionAttribute
```

Inheritance

object ← Attribute ← BaseSchemaExtensionAttribute ← ResourceTypeAttribute

Inherited Members

BaseSchemaExtensionAttribute.Name , BaseSchemaExtensionAttribute.Value ,
Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) , Attribute.GetCustomAttribute(Module, Type) ,
Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) , Attribute.GetCustomAttributes(Assembly) ,
Attribute.GetCustomAttributes(Assembly, bool) , Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) , Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,
object.Equals(object, object) , object.GetType() , object.MemberwiseClone() ,
object.ReferenceEquals(object, object) , object.ToString()

Constructors

ResourceTypeAttribute(string)

```
public ResourceTypeAttribute(string type)
```

Parameters

`type` string

Class SettingInteractivity

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public record SettingInteractivity : IEquatable<SettingInteractivity>
```

Inheritance

object ← SettingInteractivity

Implements

IEquatable<SettingInteractivity>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Id

```
public required string Id { get; init; }
```

Property Value

string

Methods

```
public required IEnumerable<MethodInformation> Methods { get; init; }
```

Property Value

IEnumerable<MethodInformation>

Uri

```
public required string Uri { get; init; }
```

Property Value

string

Enum Status

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public enum Status
```

Fields

`Error = 3`

`Fatal = 4`

`Ok = 1`

`Unknown = 0`

`Warning = 2`

Class TaskHelpers

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
public static class TaskHelpers
```

Inheritance

object ← TaskHelpers

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

TimeoutAfter(Task, TimeSpan)

```
public static Task TimeoutAfter(this Task task, TimeSpan timeout)
```

Parameters

`task` Task

`timeout` TimeSpan

Returns

Task

Exceptions

TimeoutException

TimeoutAfter<TResult>(Task<TResult>, TimeSpan)

```
public static Task<TResult> TimeoutAfter<TResult>(this Task<TResult> task,
TimeSpan timeout)
```

Parameters

`task` Task<TResult>

`timeout` TimeSpan

Returns

Task<TResult>

Type Parameters

`TResult`

Exceptions

TimeoutException

Class UIEditorAttribute

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class UIEditorAttribute : BaseSchemaExtensionAttribute
```

Inheritance

object ← Attribute ← BaseSchemaExtensionAttribute ← UIEditorAttribute

Inherited Members

BaseSchemaExtensionAttribute.Name , BaseSchemaExtensionAttribute.Value ,
Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) , Attribute.GetCustomAttribute(Module, Type) ,
Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) , Attribute.GetCustomAttributes(Assembly) ,
Attribute.GetCustomAttributes(Assembly, bool) , Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) , Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,
object.Equals(object, object) , object.GetType() , object.MemberwiseClone() ,
object.ReferenceEquals(object, object) , object.ToString()

Constructors

UIEditorAttribute(string)

```
public UIEditorAttribute(string editor)
```

Parameters

`editor` string

Class UISortOrderAttribute

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

What index the setting should be displayed in the UI. The setting with the lowest index will be displayed first (negative is allowed). Settings with no sort order will be placed last in the order they are defined in the C# module settings class

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class UISortOrderAttribute : BaseSchemaExtensionAttribute
```

Inheritance

object ← Attribute ← BaseSchemaExtensionAttribute ← UISortOrderAttribute

Inherited Members

BaseSchemaExtensionAttribute.Name , BaseSchemaExtensionAttribute.Value ,
Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) ,
Attribute.GetCustomAttribute(Assembly, Type, bool) ,
Attribute.GetCustomAttribute(MemberInfo, Type) ,
Attribute.GetCustomAttribute(MemberInfo, Type, bool) , Attribute.GetCustomAttribute(Module, Type) ,
Attribute.GetCustomAttribute(Module, Type, bool) ,
Attribute.GetCustomAttribute(ParameterInfo, Type) ,
Attribute.GetCustomAttribute(ParameterInfo, Type, bool) , Attribute.GetCustomAttributes(Assembly) ,
Attribute.GetCustomAttributes(Assembly, bool) , Attribute.GetCustomAttributes(Assembly, Type) ,
Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) ,
Attribute.GetCustomAttributes(MemberInfo, bool) , Attribute.GetCustomAttributes(MemberInfo, Type) ,
Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) ,
Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) ,
Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) ,
Attribute.GetCustomAttributes(ParameterInfo, bool) ,
Attribute.GetCustomAttributes(ParameterInfo, Type) ,
Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() ,
Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) ,
Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) ,
Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) ,
Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) ,
Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId ,

object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

UISortOrderAttribute(int)

```
public UISortOrderAttribute(int order)
```

Parameters

`order` int

Class ValueDescriptionAttribute

Namespace: Crosser.EdgeNode.Flows

Assembly: Crosser.EdgeNode.Flows.dll

Used for dictionaries and lists of simple types. The value is used as a placeholder in the input field for new entry values

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class ValueDescriptionAttribute : BaseSchemaExtensionAttribute
```

Inheritance

object ← Attribute ← BaseSchemaExtensionAttribute ← ValueDescriptionAttribute

Inherited Members

BaseSchemaExtensionAttribute.Name , BaseSchemaExtensionAttribute.Value , Attribute.Equals(object) , Attribute.GetCustomAttribute(Assembly, Type) , Attribute.GetCustomAttribute(Assembly, Type, bool) , Attribute.GetCustomAttribute(MemberInfo, Type) , Attribute.GetCustomAttribute(MemberInfo, Type, bool) , Attribute.GetCustomAttribute(Module, Type) , Attribute.GetCustomAttribute(Module, Type, bool) , Attribute.GetCustomAttribute(ParameterInfo, Type) , Attribute.GetCustomAttribute(ParameterInfo, Type, bool) , Attribute.GetCustomAttributes(Assembly) , Attribute.GetCustomAttributes(Assembly, bool) , Attribute.GetCustomAttributes(Assembly, Type) , Attribute.GetCustomAttributes(Assembly, Type, bool) , Attribute.GetCustomAttributes(MemberInfo) , Attribute.GetCustomAttributes(MemberInfo, bool) , Attribute.GetCustomAttributes(MemberInfo, Type) , Attribute.GetCustomAttributes(MemberInfo, Type, bool) , Attribute.GetCustomAttributes(Module) , Attribute.GetCustomAttributes(Module, bool) , Attribute.GetCustomAttributes(Module, Type) , Attribute.GetCustomAttributes(Module, Type, bool) , Attribute.GetCustomAttributes(ParameterInfo) , Attribute.GetCustomAttributes(ParameterInfo, bool) , Attribute.GetCustomAttributes(ParameterInfo, Type) , Attribute.GetCustomAttributes(ParameterInfo, Type, bool) , Attribute.GetHashCode() , Attribute.IsDefaultAttribute() , Attribute.IsDefined(Assembly, Type) , Attribute.IsDefined(Assembly, Type, bool) , Attribute.IsDefined(MemberInfo, Type) , Attribute.IsDefined(MemberInfo, Type, bool) , Attribute.IsDefined(Module, Type) , Attribute.IsDefined(Module, Type, bool) , Attribute.IsDefined(ParameterInfo, Type) , Attribute.IsDefined(ParameterInfo, Type, bool) , Attribute.Match(object) , Attribute.TypeId , object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ValueDescriptionAttribute(string)

```
public ValueDescriptionAttribute(string text)
```

Parameters

`text` string

Namespace Crosser.EdgeNode.Flows.Abstractions

Classes

Credential

Credential.Types

CredentialBase

CredentialWithApiKey

CredentialWithCertificate

CredentialWithConnectionString

CredentialWithData

CredentialWithUsernamePassword

FlowCredentials

FlowEndpointsConfiguration

FlowLoginResponse

FlowModuleCommonSettings
Common settings for all modules

FlowModuleException

FlowModuleSettings

FlowSecurityConfiguration

MessageEvent

ModuleStatusEvent

ResourceBase

ResourceItem

Interfaces

ICloudCommunicationService

IFlowAuthenticationService

IFlowConfigurationSection<T>

IFlowConfigurationService

IFlowCredentialsManagerService

IFlowCredentialsStoreService

IFlowModule

Base class for all modules.

IFlowResourceManagerService

IFlowResourceStoreService

Enums

FlowLoginResult

FlowModuleProtocol

Class Credential

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class Credential : CredentialBase
```

Inheritance

object ← CredentialBase ← Credential

Derived

CredentialWithApiKey, CredentialWithCertificate, CredentialWithConnectionString, CredentialWithData, CredentialWithUsernamePassword

Inherited Members

CredentialBase.Id , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

Credential()

```
public Credential()
```

Credential(Guid, byte[], string, Guid?, string)

```
[JsonConstructor]
public Credential(Guid id, byte[] data, string algorithmId, Guid? encryptionKeyId,
string credentialType)
```

Parameters

`id` Guid

`data` byte[]

`algorithmId` string

`encryptionKeyId` Guid?

`credentialType` string

Fields

ATTRIBUTE

```
public const string ATTRIBUTE = "x-credentials"
```

Field Value

string

Properties

AlgorithmId

```
public string AlgorithmId { get; set; }
```

Property Value

string

CredentialType

```
[JsonPropertyName("type")]
public string CredentialType { get; set; }
```

Property Value

string

Data

```
public byte[] Data { get; set; }
```

Property Value

byte[]

EncryptionKeyId

```
public Guid? EncryptionKeyId { get; set; }
```

Property Value

Guid?

Methods

ToCredential<T>()

```
public T? ToCredential<T>() where T : class, new()
```

Returns

T

Type Parameters

T

ToCredential<T>(Credential)

```
public static T? ToCredential<T>(Credential credential) where T : class, new()
```

Parameters

`credential` Credential

Returns

T

Type Parameters

`T`

ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

string

A string that represents the current object.

Class Credential.Types

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public static class Credential.Types
```

Inheritance

object ← Credential.Types

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

ApiKey

```
public const string ApiKey = "API-Key"
```

Field Value

string

Certificate

```
public const string Certificate = "Certificate"
```

Field Value

string

ConnectionString

```
public const string ConnectionString = "ConnectionString"
```

Field Value

string

Data

```
public const string Data = "Data"
```

Field Value

string

UsernameAndPassword

```
public const string UsernameAndPassword = "UsernameAndPassword"
```

Field Value

string

Class CredentialBase

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class CredentialBase
```

Inheritance

object ← CredentialBase

Derived

Credential

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Id

```
[Required]
public Guid Id { get; set; }
```

Property Value

Guid

Class CredentialWithApiKey

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class CredentialWithApiKey : Credential
```

Inheritance

object ← CredentialBase ← Credential ← CredentialWithApiKey

Inherited Members

Credential.ATTRIBUTE , Credential.Data , Credential.AlgorithmId , Credential.EncryptionKeyId , Credential.CredentialType , Credential.ToString() , Credential.ToCredential<T>() , Credential.ToCredential<T>(Credential) , CredentialBase.Id , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Properties

ApiKey

```
public string? ApiKey { get; set; }
```

Property Value

string

Class CredentialWithCertificate

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class CredentialWithCertificate : Credential
```

Inheritance

object ← CredentialBase ← Credential ← CredentialWithCertificate

Inherited Members

Credential.ATTRIBUTE , Credential.Data , Credential.AlgorithmId , Credential.EncryptionKeyId , Credential.CredentialType , Credential.ToString() , Credential.ToCredential<T>() , Credential.ToCredential<T>(Credential) , CredentialBase.Id , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

CredentialWithCertificate()

```
public CredentialWithCertificate()
```

CredentialWithCertificate(byte[], string)

```
[JsonConstructor]
public CredentialWithCertificate(byte[] certificate, string password)
```

Parameters

`certificate` byte[]

`password` string

Properties

Certificate

```
public byte[] Certificate { get; set; }
```

Property Value

byte[]

Password

```
public string? Password { get; set; }
```

Property Value

string

X509Certificate2

```
[JsonIgnore]
public X509Certificate2 X509Certificate2 { get; }
```

Property Value

X509Certificate2

Class CredentialWithConnectionString

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class CredentialWithConnectionString : Credential
```

Inheritance

object ← CredentialBase ← Credential ← CredentialWithConnectionString

Inherited Members

Credential.ATTRIBUTE , Credential.Data , Credential.AlgorithmId , Credential.EncryptionKeyId , Credential.CredentialType , Credential.ToString() , Credential.ToCredential<T>() , Credential.ToCredential<T>(Credential) , CredentialBase.Id , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Properties

ConnectionString

```
public string? ConnectionString { get; set; }
```

Property Value

string

Class CredentialWithData

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class CredentialWithData : Credential
```

Inheritance

object ← CredentialBase ← Credential ← CredentialWithData

Inherited Members

Credential.ATTRIBUTE , Credential.Data , Credential.AlgorithmId , Credential.EncryptionKeyId , Credential.CredentialType , Credential.ToString() , Credential.ToCredential<T>() , Credential.ToCredential<T>(Credential) , CredentialBase.Id , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Class CredentialWithUsernamePassword

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class CredentialWithUsernamePassword : Credential
```

Inheritance

object ← CredentialBase ← Credential ← CredentialWithUsernamePassword

Inherited Members

Credential.ATTRIBUTE , Credential.Data , Credential.AlgorithmId , Credential.EncryptionKeyId , Credential.CredentialType , Credential.ToString() , Credential.ToCredential<T>() , Credential.ToCredential<T>(Credential) , CredentialBase.Id , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Properties

Password

```
public string? Password { get; set; }
```

Property Value

string

Username

```
public string? Username { get; set; }
```

Property Value

string

Class FlowCredentials

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowCredentials
```

Inheritance

object ← FlowCredentials

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

AccessKey

```
public string AccessKey { get; set; }
```

Property Value

string

JwtId

```
[JsonIgnore]
public Guid? JwtId { get; set; }
```

Property Value

Guid?

NodeId

```
public Guid NodeId { get; set; }
```

Property Value

Guid

OrganizationId

```
public Guid? OrganizationId { get; set; }
```

Property Value

Guid?

Class FlowEndpointsConfiguration

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowEndpointsConfiguration :
IFlowConfigurationSection<FlowEndpointsConfiguration>
```

Inheritance

object ← FlowEndpointsConfiguration

Implements

IFlowConfigurationSection<FlowEndpointsConfiguration>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

API

```
public const string API = "/api"
```

Field Value

string

Properties

AlwaysTrustRemoteCertificate

```
public bool? AlwaysTrustRemoteCertificate { get; set; }
```

Property Value

bool?

BaseUrl

```
public string BaseUrl { get; set; }
```

Property Value

string

HttpScheme

```
[JsonIgnore]
public string HttpScheme { get; }
```

Property Value

string

Port

```
public int Port { get; set; }
```

Property Value

int

Secure

```
public bool? Secure { get; set; }
```

Property Value

bool?

Methods

Default()

```
public static FlowEndpointsConfiguration Default()
```

Returns

FlowEndpointsConfiguration

GetDownloadCredentialUrl(FlowCredentials, Guid)

```
public string GetDownloadCredentialUrl(FlowCredentials credentials,
Guid credentialId)
```

Parameters

`credentials` FlowCredentials

`credentialId` Guid

Returns

string

GetLoginUrl()

```
public string GetLoginUrl()
```

Returns

string

GetResourceDataUrl(FlowCredentials, Guid)

```
public string GetResourceDataUrl(FlowCredentials credentials, Guid id)
```

Parameters

`credentials` FlowCredentials

`id` Guid

Returns

string

GetResourceMetadataUrl(FlowCredentials, Guid)

```
public string GetResourceMetadataUrl(FlowCredentials credentials, Guid id)
```

Parameters

`credentials` FlowCredentials

`id` Guid

Returns

string

Instance()

```
public FlowEndpointsConfiguration Instance()
```

Returns

FlowEndpointsConfiguration

Class FlowLoginResponse

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowLoginResponse
```

Inheritance

object ← FlowLoginResponse

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Jwt

```
public string Jwt { get; set; }
```

Property Value

string

Result

```
public FlowLoginResult Result { get; set; }
```

Property Value

FlowLoginResult

Enum FlowLoginResult

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public enum FlowLoginResult
```

Fields

`MissingOrInvalidCredentials = 1`

`Ok = 0`

`Timeout = 3`

`Unauthorized = 2`

Class FlowModuleCommonSettings

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

Common settings for all modules

```
public class FlowModuleCommonSettings
```

Inheritance

object ← FlowModuleCommonSettings

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowModuleCommonSettings()

```
public FlowModuleCommonSettings()
```

Fields

MAX_QUEUE_SIZE

```
public const int MAX_QUEUE_SIZE = 2147483647
```

Field Value

int

MIN_QUEUE_SIZE

```
public const int MIN_QUEUE_SIZE = 100
```

Field Value

int

Properties

BypassMessagesNotMatchingRules

```
public bool BypassMessagesNotMatchingRules { get; set; }
```

Property Value

bool

ChannelFullMode

```
public BoundedChannelFullMode ChannelFullMode { get; set; }
```

Property Value

BoundedChannelFullMode

DisableQueues

```
public bool DisableQueues { get; set; }
```

Property Value

bool

EnableBackoffForRetries

Added in SDK 4.0.0

```
public bool EnableBackoffForRetries { get; set; }
```

Property Value

bool

InitializeTimeout

```
public int InitializeTimeout { get; set; }
```

Property Value

int

MaxItemsInQueue

```
[Range(100, 2147483647)]
public int MaxItemsInQueue { get; set; }
```

Property Value

int

MaxMessagesRetryDelay

Added in SDK 4.0.0 Will be used when EnableBackOffForRetries is true Time is specified in seconds

```
public int MaxMessagesRetryDelay { get; set; }
```

Property Value

int

MessagesDelay

```
public int MessagesDelay { get; set; }
```

Property Value

int

MessagesRetries

```
public int MessagesRetries { get; set; }
```

Property Value

int

MessagesRetryDelay

```
public int MessagesRetryDelay { get; set; }
```

Property Value

int

PersistentMessages

```
public bool PersistentMessages { get; set; }
```

Property Value

bool

Rules

```
public List<MessageFilter> Rules { get; set; }
```

Property Value

List<MessageFilter>

StartTimeout

```
public int StartTimeout { get; set; }
```

Property Value

int

StopTimeout

```
public int StopTimeout { get; set; }
```

Property Value

int

Methods

Validate(SettingsValidator)

If validation is needed overwrite this method

```
public virtual void Validate(SettingsValidator validator)
```

Parameters

`validator` SettingsValidator

Class FlowModuleException

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowModuleException : Exception, ISerializable
```

Inheritance

object ← Exception ← FlowModuleException

Implements

ISerializable

Inherited Members

Exception.GetBaseException() , Exception.GetType() , Exception.ToString() , Exception.Data , Exception.HelpLink , Exception.HResult , Exception.InnerException , Exception.Message , Exception.Source , Exception.StackTrace , Exception.TargetSite , Exception.SerializeObjectState , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.MemberwiseClone() , object.ReferenceEquals(object, object)

Constructors

FlowModuleException(IFlowModule, IError)

```
public FlowModuleException(IFlowModule module, IError error)
```

Parameters

`module` IFlowModule

`error` IError

FlowModuleException(IFlowModule, string)

```
public FlowModuleException(IFlowModule module, string message)
```

Parameters

`module` IFlowModule

`message` string

FlowModuleException(IFlowModule, string, Exception)

```
public FlowModuleException(IFlowModule module, string message,
Exception innerException)
```

Parameters

`module` IFlowModule

`message` string

`innerException` Exception

Properties

Error

```
public IError Error { get; }
```

Property Value

IError

Module

```
public IFlowModule Module { get; }
```

Property Value

IFlowModule

Enum FlowModuleProtocol

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
[Obsolete("Will be removed when modules are responsible for their
own communication")]
public enum FlowModuleProtocol
```

Fields

HTTP = 3

MQTT = 1

NONE = 0

OPC = 2

WEBSOCKET = 4

Class FlowModuleSettings

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowModuleSettings
```

Inheritance

object ← FlowModuleSettings

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowModuleSettings()

```
public FlowModuleSettings()
```

Properties

CommonSettings

```
public FlowModuleCommonSettings CommonSettings { get; set; }
```

Property Value

FlowModuleCommonSettings

Credentials

```
[JsonIgnore]
```

```
public IDictionary<Guid, Credential> Credentials { get; set; }
```

Property Value

IDictionary<Guid, Credential>

Resources

```
[JsonIgnore]
public IDictionary<Guid, ResourceItem> Resources { get; set; }
```

Property Value

IDictionary<Guid, ResourceItem>

Methods

GetCredentialById(Guid)

```
[Obsolete("Use GetCredential or GetCredentialContentAs<T> on the FlowModule")]
public Credential? GetCredentialById(Guid id)
```

Parameters

`id` Guid

Returns

Credential

GetCredentialProperties()

```
public IEnumerable<(Guid credentialId, string[] credentialTypes)>
GetCredentialProperties()
```

Returns

IEnumerable<(Guid credentialId, string[] credentialTypes)>

GetResourceById(Guid)

```
[Obsolete("Use GetResource or GetResourceContentAs<T> on the FlowModule")]
public ResourceItem? GetResourceById(Guid id)
```

Parameters

`id` Guid

Returns

ResourceItem

GetResourceProperties()

```
public IEnumerable<(Guid resourceId, string[] resourceTypes)>
GetResourceProperties()
```

Returns

IEnumerable<(Guid credentialId, string[] credentialTypes)>

Validate(SettingsValidator)

If validation is needed overwrite this method

```
public virtual void Validate(SettingsValidator validator)
```

Parameters

`validator` SettingsValidator

Class FlowSecurityConfiguration

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowSecurityConfiguration :
IFlowConfigurationSection<FlowSecurityConfiguration>
```

Inheritance

object ← FlowSecurityConfiguration

Implements

IFlowConfigurationSection<FlowSecurityConfiguration>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Credentials

```
public FlowCredentials Credentials { get; set; }
```

Property Value

FlowCredentials

JWT

```
[JsonIgnore]
public string JWT { get; set; }
```

Property Value

string

Methods

Default()

```
public static FlowSecurityConfiguration Default()
```

Returns

FlowSecurityConfiguration

HasCredentials()

```
public bool HasCredentials()
```

Returns

bool

Instance()

```
public FlowSecurityConfiguration Instance()
```

Returns

FlowSecurityConfiguration

JwtHasExpired()

```
public bool JwtHasExpired()
```

Returns

bool

RegisterJwt(string)

```
public void RegisterJwt(string jwt)
```

Parameters

`jwt` string

Interface ICloudCommunicationService

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public interface ICloudCommunicationService
```

Methods

DownloadCredential(Guid, int)

```
Task<IHttpResult<Credential>> DownloadCredential(Guid credentialId, int retry = 3)
```

Parameters

`credentialId` Guid

`retry` int

Returns

Task<IHttpResult<Credential>>

DownloadResource(Guid, int)

```
Task<IHttpResult<ResourceItem>> DownloadResource(Guid resourceId, int retry = 3)
```

Parameters

`resourceId` Guid

`retry` int

Returns

Task<IHttpResult<ResourceItem>>

DownloadResourceData(ResourceItem, int)

```
Task<IHttpResult<Stream>> DownloadResourceData(ResourceItem resourceItem, int retry
= 3)
```

Parameters

`resourceItem` ResourceItem

`retry` int

Returns

Task<IHttpResult<Stream>>

Interface IFlowAuthenticationService

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public interface IFlowAuthenticationService : IDisposable
```

Inherited Members
IDisposable.Dispose()

Methods

EnsureAuthentication()

```
Task<FlowLoginResult> EnsureAuthentication()
```

Returns

Task<FlowLoginResult>

IsAuthenticated()

```
Task<bool> IsAuthenticated()
```

Returns

Task<bool>

Login(FlowCredentials)

```
Task<FlowLoginResponse> Login(FlowCredentials credentials)
```

Parameters

`credentials` FlowCredentials

Returns

Task<FlowLoginResponse>

Reset()

```
void Reset()
```

Interface IFlowConfigurationSection<T>

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public interface IFlowConfigurationSection<T>
```

Type Parameters

T

Methods

Instance()

```
T Instance()
```

Returns

T

Interface IFlowConfigurationService

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public interface IFlowConfigurationService
```

Properties

EndpointsConfiguration

```
FlowEndpointsConfiguration EndpointsConfiguration { get; }
```

Property Value

FlowEndpointsConfiguration

SecurityConfiguration

```
FlowSecurityConfiguration SecurityConfiguration { get; }
```

Property Value

FlowSecurityConfiguration

UserAgent

```
string UserAgent { get; }
```

Property Value

string

Methods

RegisterJwt(string)

```
void RegisterJwt(string jwt)
```

Parameters

`jwt` string

UpdateCredentialsJson(FlowSecurityConfiguration)

```
void UpdateCredentialsJson(FlowSecurityConfiguration securityConfiguration)
```

Parameters

`securityConfiguration` FlowSecurityConfiguration

Interface IFlowCredentialsManagerService

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public interface IFlowCredentialsManagerService
```

Methods

GetCredential(Guid)

```
Task<IResult<Credential>> GetCredential(Guid credentialId)
```

Parameters

`credentialId` Guid

Returns

Task<IResult<Credential>>

Interface IFlowCredentialsStoreService

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public interface IFlowCredentialsStoreService
```

Methods

CreateCredential(Credential)

```
IResult<Credential> CreateCredential(Credential credential)
```

Parameters

`credential` Credential

Returns

IResult<Credential>

CredentialExist(Guid)

```
bool CredentialExist(Guid credentialId)
```

Parameters

`credentialId` Guid

Returns

bool

GetCredential(Guid)

```
IResult<Credential> GetCredential(Guid credentialId)
```

Parameters

`credentialId` Guid

Returns

IResult<Credential>

Interface IFlowModule

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

Base class for all modules.

```
public interface IFlowModule : IDisposable
```

Inherited Members
IDisposable.Dispose()

Properties

Categories

```
List<string> Categories { get; set; }
```

Property Value

List<string>

Debug

```
bool Debug { get; set; }
```

Property Value

bool

Description

```
string Description { get; }
```

Property Value

string

Disabled

```
bool Disabled { get; set; }
```

Property Value

bool

FlowContext

```
[JsonIgnore]
RepositoryInstance<string, dynamic> FlowContext { get; }
```

Property Value

RepositoryInstance<string, dynamic>

FlowId

```
Guid FlowId { get; set; }
```

Property Value

Guid

FlowName

```
string FlowName { get; }
```

Property Value

string

FlowStatistics

```
[Obsolete("Will be removed in the next major version")]
[JsonIgnore]
FlowStatistics FlowStatistics { get; set; }
```

Property Value

FlowStatistics

HaltOnError

```
bool HaltOnError { get; set; }
```

Property Value

bool

HasInput

```
bool HasInput { get; set; }
```

Property Value

bool

HasOutput

```
bool HasOutput { get; }
```

Property Value

bool

Id

```
Guid Id { get; set; }
```

Property Value

Guid

Index

```
int Index { get; set; }
```

Property Value

int

Logger

```
ILog? Logger { get; }
```

Property Value

ILog

ModuleProtocol

```
[Obsolete("Will be removed in the next major version")]
FlowModuleProtocol ModuleProtocol { get; set; }
```

Property Value

FlowModuleProtocol

ModuleStatistics

```
[Obsolete("Will be removed in the next major version")]
[JsonIgnore]
FlowModuleStatistics ModuleStatistics { get; set; }
```

Property Value

FlowModuleStatistics

ModuleType

```
FlowModuleType ModuleType { get; set; }
```

Property Value

FlowModuleType

ModuleVersion

```
string ModuleVersion { get; set; }
```

Property Value

string

Name

```
string Name { get; set; }
```

Property Value

string

OnDebug

```
[JsonIgnore]
EventHandler<ModuleDebugMessage>? OnDebug { get; set; }
```

Property Value

EventHandler<ModuleDebugMessage>

OnError

```
[JsonIgnore]
EventHandler<FlowModuleException>? OnError { get; set; }
```

Property Value

EventHandler<FlowModuleException>

OnFlowDataReceived

```
[Obsolete("Will be removed in the next major version")]
[JsonIgnore]
EventHandler<FlowStatisticsData>? OnFlowDataReceived { get; set; }
```

Property Value

EventHandler<FlowStatisticsData>

OnFlowDataSent

```
[Obsolete("Will be removed in the next major version")]
[JsonIgnore]
EventHandler<FlowStatisticsData>? OnFlowDataSent { get; set; }
```

Property Value

EventHandler<FlowStatisticsData>

OnMessageDeadLetter

```
[JsonIgnore]
EventHandler<MessageEvent>? OnMessageDeadLetter { get; set; }
```

Property Value

EventHandler<MessageEvent>

OnMessageDropped

```
[JsonIgnore]
EventHandler<MessageEvent>? OnMessageDropped { get; set; }
```

Property Value

EventHandler<MessageEvent>

OnModuleDataReceived

```
[Obsolete("Will be removed in the next major version")]
[JsonIgnore]
EventHandler<FlowModuleStatisticsData>? OnModuleDataReceived { get; set; }
```

Property Value

EventHandler<FlowModuleStatisticsData>

OnModuleDataSent

```
[Obsolete("Will be removed in the next major version")]
[JsonIgnore]
EventHandler<FlowModuleStatisticsData>? OnModuleDataSent { get; set; }
```

Property Value

EventHandler<FlowModuleStatisticsData>

OnStatusChanged

```
[JsonIgnore]
EventHandler<ModuleStatusEvent>? OnStatusChanged { get; set; }
```

Property Value

EventHandler<ModuleStatusEvent>

OnWarning

```
[JsonIgnore]
EventHandler<FlowModuleException>? OnWarning { get; set; }
```

Property Value

EventHandler<FlowModuleException>

Outputs

```
List<RepositoryInstance<Guid, IFlowModule?>> Outputs { get; set; }
```

Property Value

List<RepositoryInstance<Guid, IFlowModule>>

ReceivesInputFromMultipleModules

```
bool ReceivesInputFromMultipleModules { get; set; }
```

Property Value

bool

RemoteSessionEnabled

```
bool RemoteSessionEnabled { get; set; }
```

Property Value

bool

Resources

```
IList<ResourceBase> Resources { get; set; }
```

Property Value

IList<ResourceBase>

Settings

```
FlowModuleSettings Settings { get; }
```

Property Value

FlowModuleSettings

Status

```
Status Status { get; }
```

Property Value

Status

Topic

```
[Obsolete("Will be removed in the next major version")]
string? Topic { get; set; }
```

Property Value

string

Type

```
string Type { get; }
```

Property Value

string

UserFriendlyName

```
string UserFriendlyName { get; }
```

Property Value

string

Methods

CanStart()

```
[Obsolete("Use Initialize and Start to handle and return errors")]
IError? CanStart()
```

Returns

IError

FlowDefinitionId()

```
Guid FlowDefinitionId()
```

Returns

Guid

Initialize()

```
Task<IError?> Initialize()
```

Returns

Task<IError>

InitializeInteractivity()

Override this method and register your module interactivity methods

```
void InitializeInteractivity()
```

LoadSettings(string)

```
void LoadSettings(string settingsAsJson)
```

Parameters

`settingsAsJson` string

Receive(IFlowMessage)

```
Task Receive(IFlowMessage message)
```

Parameters

`message` IFlowMessage

Returns

Task

ReceiveFrom(IFlowModule, int)

```
IFlowModule ReceiveFrom(IFlowModule module, int ix = 0)
```

Parameters

`module` IFlowModule

`ix` int

Returns

IFlowModule

SendTo(IFlowModule, int)

```
IFlowModule SendTo(IFlowModule module, int ix = 0)
```

Parameters

`module` IFlowModule

`ix` int

Returns

IFlowModule

SetStatus(Status)

```
void SetStatus(Status status)
```

Parameters

`status` Status

SetStatus(Status, string)

```
void SetStatus(Status status, string message)
```

Parameters

`status` Status

`message` string

SetStatus(Status, string, bool, Exception)

```
void SetStatus(Status status, string message, bool force, Exception exception)
```

Parameters

`status` Status

`message` string

`force` bool

`exception` Exception

Start()

```
Task<IError?> Start()
```

Returns

Task<IError>

Stop()

```
Task Stop()
```

Returns

Task

ValidateSettings(SettingsValidator)

```
void ValidateSettings(SettingsValidator validator)
```

Parameters

`validator` SettingsValidator

Interface IFlowResourceManagerService

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public interface IFlowResourceManagerService
```

Methods

GetResource(Guid, int)

```
Task<IResult<(string file, ResourceItem resourceItem)>> GetResource(Guid resourceId,
int retry = 3)
```

Parameters

`resourceId` Guid

`retry` int

Returns

Task<IResult<(string file, ResourceItem resourceItem)>>

Interface IFlowResourceStoreService

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public interface IFlowResourceStoreService
```

Methods

Create(ResourceItem, Stream)

```
bool Create(ResourceItem resourceItem, Stream data)
```

Parameters

`resourceItem` ResourceItem

`data` Stream

Returns

bool

Exists(ResourceItem)

```
bool Exists(ResourceItem resourceItem)
```

Parameters

`resourceItem` ResourceItem

Returns

bool

GetById(Guid)

```
IResult<(string file, ResourceItem resourceItem)> GetById(Guid resourceId)
```

Parameters

`resourceId` Guid

Returns

IResult<(string file, ResourceItem resourceItem)>

Class MessageEvent

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public record MessageEvent : IEquatable<MessageEvent>
```

Inheritance

object ← MessageEvent

Implements

IEquatable<MessageEvent>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MessageEvent(Guid, string, string, string, Status, string, BoundedChannelFullMode, int, int, float, float, bool, Guid, Guid, string, Guid?, DateTime)

```
public MessageEvent(Guid ModuleId, string ModuleName, string ModuleType, string
Version, Status Status, string Reason, BoundedChannelFullMode Mode, int QueueSize,
int NumberOfRetries, float RetryDelay, float Delay, bool Persistent, Guid FlowId,
Guid FlowDefinitionId, string FlowName, Guid? MessageId, DateTime Timestamp)
```

Parameters

`ModuleId` Guid

`ModuleName` string

`ModuleType` string

`Version` string

`Status` Status

`Reason` string

`Mode` BoundedChannelFullMode

`QueueSize` int

`NumberOfRetries` int

`RetryDelay` float

`Delay` float

`Persistent` bool

`FlowId` Guid

`FlowDefinitionId` Guid

`FlowName` string

`MessageId` Guid?

`Timestamp` DateTime

Properties

Delay

```
public float Delay { get; init; }
```

Property Value

float

FlowDefinitionId

```
public Guid FlowDefinitionId { get; init; }
```

Property Value

Guid

FlowId

```
public Guid FlowId { get; init; }
```

Property Value

Guid

FlowName

```
public string FlowName { get; init; }
```

Property Value

string

MessageId

```
public Guid? MessageId { get; init; }
```

Property Value

Guid?

Mode

```
public BoundedChannelFullMode Mode { get; init; }
```

Property Value

BoundedChannelFullMode

ModuleId

```
public Guid ModuleId { get; init; }
```

Property Value

Guid

ModuleName

```
public string ModuleName { get; init; }
```

Property Value

string

ModuleType

```
public string ModuleType { get; init; }
```

Property Value

string

NumberOfRetries

```
public int NumberOfRetries { get; init; }
```

Property Value

int

Persistent

```
public bool Persistent { get; init; }
```

Property Value

bool

QueueSize

```
public int QueueSize { get; init; }
```

Property Value

int

Reason

```
public string Reason { get; init; }
```

Property Value

string

RetryDelay

```
public float RetryDelay { get; init; }
```

Property Value

float

Status

```
public Status Status { get; init; }
```

Property Value

Status

Timestamp

```
public DateTime Timestamp { get; init; }
```

Property Value

DateTime

Version

```
public string Version { get; init; }
```

Property Value

string

Methods

Create(IFlowModule, string, Guid?)

```
public static MessageEvent Create(IFlowModule flowModule, string reason, Guid?
messageId = null)
```

Parameters

`flowModule` IFlowModule

`reason` string

`messageId` Guid?

Returns

MessageEvent

Class ModuleStatusEvent

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class ModuleStatusEvent : EventArgs
```

Inheritance

object ← EventArgs ← ModuleStatusEvent

Inherited Members

EventArgs.Empty , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

ModuleStatusEvent(Status, Status, Guid, string, string, string, bool, Exception?)

```
public ModuleStatusEvent(Status oldStatus, Status newStatus, Guid id, string name,
string moduleVersion, string message, bool haltOnError, Exception? exception)
```

Parameters

`oldStatus` Status

`newStatus` Status

`id` Guid

`name` string

`moduleVersion` string

`message` string

`haltOnError` bool

`exception` Exception

Properties

Exception

```
public Exception? Exception { get; }
```

Property Value

Exception

HaltOnError

```
public bool HaltOnError { get; }
```

Property Value

bool

Id

```
public Guid Id { get; }
```

Property Value

Guid

Message

```
public string Message { get; set; }
```

Property Value

string

ModuleVersion

```
public string ModuleVersion { get; }
```

Property Value

string

Name

```
public string Name { get; }
```

Property Value

string

NewStatus

```
public Status NewStatus { get; }
```

Property Value

Status

OldStatus

```
public Status OldStatus { get; }
```

Property Value

Status

Class ResourceBase

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class ResourceBase
```

Inheritance

object ← ResourceBase

Derived

ResourceItem

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Id

This is still used by EN and will be used until ED stops sending the property and instead only send the new property 'VersionId'

```
[Required]
public Guid Id { get; set; }
```

Property Value

Guid

Class ResourceItem

Namespace: Crosser.EdgeNode.Flows.Abstractions

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class ResourceItem : ResourceBase, IValidatableObject
```

Inheritance

object ← ResourceBase ← ResourceItem

Implements

IValidatableObject

Inherited Members

ResourceBase.Id , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

ATTRIBUTE

```
public const string ATTRIBUTE = "x-resources"
```

Field Value

string

Properties

Md5Sum

```
public string Md5Sum { get; set; }
```

Property Value

string

Name

```
public string Name { get; set; }
```

Property Value

string

OrganizationId

```
public Guid? OrganizationId { get; set; }
```

Property Value

Guid?

Path

```
[Required]
public string Path { get; set; }
```

Property Value

string

SizeInBytes

```
public long SizeInBytes { get; set; }
```

Property Value

long

Type

```
public string Type { get; set; }
```

Property Value

string

Url

```
public string Url { get; set; }
```

Property Value

string

Methods

ToResource<T>()

```
public T? ToResource<T>() where T : class, new()
```

Returns

T

Type Parameters

T

ToResource<T>(ResourceItem)

```
public static T? ToResource<T>(ResourceItem resource) where T : class, new()
```

Parameters

`resource` ResourceItem

Returns

T

Type Parameters

`T`

Validate(ValidationContext)

Determines whether the specified object is valid.

```
public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
```

Parameters

`validationContext` ValidationContext

The validation context.

Returns

IEnumerable<ValidationResult>

A collection that holds failed-validation information.

ValidatePath(string, string)

```
public static ValidationResult? ValidatePath(string value, string name)
```

Parameters

`value` string

`name` string

Returns

ValidationResult

Namespace Crosser.EdgeNode.Flows.Abstractions.Filter

Classes

MessageFilter

MessageFilterExtensions

Enums

ComparisonType

ExpressionType

Enum ComparisonType

Namespace: Crosser.EdgeNode.Flows.Abstractions.Filter

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public enum ComparisonType
```

Fields

`Any = 0`

`Boolean = 3`

`Number = 2`

`Property = 4`

`String = 1`

Enum ExpressionType

Namespace: Crosser.EdgeNode.Flows.Abstractions.Filter

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public enum ExpressionType
```

Fields

Contains = 10

DoesNotContain = 11

Empty = 17

Endswith = 13

EqualTo = 4

GreaterThan = 8

GreaterThanOrEqualTo = 9

IsFalse = 1

IsNotNumeric = 3

IsNumeric = 2

IsTrue = 0

LessThan = 6

LessThanOrEqualTo = 7

NotEqualTo = 5

NotNull = 15

Null = 14

Regex = 16

```
Startswith = 12
```

Class MessageFilter

Namespace: Crosser.EdgeNode.Flows.Abstractions.Filter

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class MessageFilter
```

Inheritance

object ← MessageFilter

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

MessageFilter(string, ExpressionType, string?, ComparisonType, bool)

```
public MessageFilter(string propertyName, ExpressionType @operator, string? value,
ComparisonType comparisonType = ComparisonType.Any, bool missingPropertyIsValid
= false)
```

Parameters

`propertyName` string

`operator` ExpressionType

`value` string

`comparisonType` ComparisonType

`missingPropertyIsValid` bool

Properties

ComparisonType

```
public ComparisonType ComparisonType { get; set; }
```

Property Value

ComparisonType

MissingPropertyIsValid

```
public bool MissingPropertyIsValid { get; set; }
```

Property Value

bool

Operator

```
public ExpressionType Operator { get; set; }
```

Property Value

ExpressionType

PropertyName

```
public string PropertyName { get; set; }
```

Property Value

string

Value

```
public string Value { get; set; }
```

Property Value

string

Class MessageFilterExtensions

Namespace: Crosser.EdgeNode.Flows.Abstractions.Filter

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public static class MessageFilterExtensions
```

Inheritance

object ← MessageFilterExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

FilterApplyAll(IFlowMessage, List<MessageFilter>)

```
public static bool FilterApplyAll(this IFlowMessage o, List<MessageFilter> rules)
```

Parameters

`o` IFlowMessage

`rules` List<MessageFilter>

Returns

bool

FilterApplyAny(IFlowMessage, List<MessageFilter>)

```
public static bool FilterApplyAny(this IFlowMessage o, List<MessageFilter> rules)
```

Parameters

o IFlowMessage

rules List<MessageFilter>

Returns

bool

IsNumericRegEx(string?)

```
public static bool IsNumericRegEx(string? str)
```

Parameters

str string

Returns

bool

IsTrueString(string?)

```
public static bool IsTrueString(string? v)
```

Parameters

v string

Returns

bool

Namespace Crosser.EdgeNode.Flows.Communication

Classes

CloudCommunicationService

CloudRequestService

EncryptionHelper

FlowAuthenticationService

FlowConfigurationService

FlowCredentialsManagerService

FlowCredentialsStoreService

FlowPathHelpers

FlowResourceManagerService

FlowResourceStoreService

FlowUrlResolverService

Class CloudCommunicationService

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class CloudCommunicationService : ICloudCommunicationService
```

Inheritance

object ← CloudCommunicationService

Implements

ICloudCommunicationService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CloudCommunicationService(IFlowAuthenticationService, CloudRequestService, FlowUrlResolverService)

```
public CloudCommunicationService(IFlowAuthenticationService authenticationService,
CloudRequestService httpRequestService, FlowUrlResolverService urlResolverService)
```

Parameters

`authenticationService` IFlowAuthenticationService

`httpRequestService` CloudRequestService

`urlResolverService` FlowUrlResolverService

Methods

DownloadCredential(Guid, int)

```
public Task<IHttpResult<Credential>> DownloadCredential(Guid credentialId, int retry
= 3)
```

Parameters

`credentialId` Guid

`retry` int

Returns

Task<IHttpResult<Credential>>

DownloadResource(Guid, int)

```
public Task<IHttpResult<ResourceItem>> DownloadResource(Guid resourceId, int retry
= 3)
```

Parameters

`resourceId` Guid

`retry` int

Returns

Task<IHttpResult<ResourceItem>>

DownloadResourceData(ResourceItem, int)

```
public Task<IHttpResult<Stream>> DownloadResourceData(ResourceItem resourceItem, int
retry = 3)
```

Parameters

`resourceItem` ResourceItem

`retry` int

Returns

Task<IHttpResult<Stream>>

Class CloudRequestService

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class CloudRequestService
```

Inheritance

object ← CloudRequestService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

CloudRequestService(IFlowConfigurationService, IHttpClientFactory)

```
public CloudRequestService(IFlowConfigurationService configurationService,
IHttpClientFactory httpClientFactory)
```

Parameters

`configurationService` IFlowConfigurationService

`httpClientFactory` IHttpClientFactory

Methods

GetHttpErrorDescription(HttpResponseMessage, string, string)

```
public static Task<string> GetHttpErrorDescription(HttpResponseMessage response,
string uri, string method)
```

Parameters

`response` HttpResponseMessage

`uri` string

`method` string

Returns

Task<string>

MakeHttpGetRequest<T>(string, bool)

```
public Task<IHttpResult<T>> MakeHttpGetRequest<T>(string url, bool
requireAuthentication = true)
```

Parameters

`url` string

`requireAuthentication` bool

Returns

Task<IHttpResult<T>>

Type Parameters

`T`

MakeHttpGetStreamRequest(string, bool, int)

```
public Task<IHttpResult<Stream>> MakeHttpGetStreamRequest(string url, bool
requireAuthentication = true, int timeout = -1)
```

Parameters

`url` string

`requireAuthentication` bool

`timeout` int

Returns

Task<IHttpResult<Stream>>

MakeHttpPostRequest(string, object, bool)

```
public Task<IHttpResult<HttpResponseMessage>> MakeHttpPostRequest(string url, object
data, bool requireAuthentication = true)
```

Parameters

`url` string

`data` object

`requireAuthentication` bool

Returns

Task<IHttpResult<HttpResponseMessage>>

Class EncryptionHelper

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class EncryptionHelper
```

Inheritance

object ← EncryptionHelper

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

EncryptionHelper(string)

```
public EncryptionHelper(string secret)
```

Parameters

`secret` string

Methods

AesDecrypt(string)

```
public string AesDecrypt(string cipherText)
```

Parameters

`cipherText` string

Returns

string

AesEncrypt(string)

```
public string AesEncrypt(string text)
```

Parameters

`text` string

Returns

string

Class FlowAuthenticationService

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class FlowAuthenticationService : IFlowAuthenticationService, IDisposable
```

Inheritance

object ← FlowAuthenticationService

Implements

IFlowAuthenticationService, IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowAuthenticationService(IFlowConfigurationService, CloudRequestService, FlowUrlResolverService)

```
public FlowAuthenticationService(IFlowConfigurationService configurationService,
CloudRequestService httpRequestService, FlowUrlResolverService urlResolverService)
```

Parameters

`configurationService` IFlowConfigurationService

`httpRequestService` CloudRequestService

`urlResolverService` FlowUrlResolverService

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Dispose(bool)

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

EnsureAuthentication()

```
public Task<FlowLoginResult> EnsureAuthentication()
```

Returns

Task<FlowLoginResult>

IsAuthenticated()

```
public Task<bool> IsAuthenticated()
```

Returns

Task<bool>

Login(FlowCredentials)

```
public Task<FlowLoginResponse> Login(FlowCredentials credentials)
```

Parameters

`credentials` FlowCredentials

Returns

Task<FlowLoginResponse>

Reset()

```
public void Reset()
```

Class FlowConfigurationService

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class FlowConfigurationService : IFlowConfigurationService
```

Inheritance

object ← FlowConfigurationService

Implements

IFlowConfigurationService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowConfigurationService()

```
public FlowConfigurationService()
```

Properties

EndpointsConfiguration

```
public FlowEndpointsConfiguration EndpointsConfiguration { get; }
```

Property Value

FlowEndpointsConfiguration

SecurityConfiguration

```
public FlowSecurityConfiguration SecurityConfiguration { get; }
```

Property Value

FlowSecurityConfiguration

SoftwareVersion

```
public string SoftwareVersion { get; }
```

Property Value

string

UserAgent

```
public string UserAgent { get; }
```

Property Value

string

Methods

CopyValuesMissingOnTarget<T>(T, T)

```
public static void CopyValuesMissingOnTarget<T>(T target, T source)
```

Parameters

`target` T

`source` T

Type Parameters

`T`

MakeSureFileExists(string)

```
public static void MakeSureFileExists(string filename)
```

Parameters

`filename` string

RegisterJwt(string)

```
public void RegisterJwt(string jwt)
```

Parameters

`jwt` string

UpdateCredentialsJson(FlowSecurityConfiguration)

```
public void UpdateCredentialsJson(FlowSecurityConfiguration securityConfiguration)
```

Parameters

`securityConfiguration` FlowSecurityConfiguration

Class FlowCredentialsManagerService

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class FlowCredentialsManagerService : IFlowCredentialsManagerService
```

Inheritance

object ← FlowCredentialsManagerService

Implements

IFlowCredentialsManagerService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowCredentialsManagerService(ICloudCommunicationService, IFlowCredentialsStoreService)

```
public FlowCredentialsManagerService(ICloudCommunicationService
flowCommunicationService, IFlowCredentialsStoreService credentialsStore)
```

Parameters

`flowCommunicationService` ICloudCommunicationService

`credentialsStore` IFlowCredentialsStoreService

Fields

_credentialsStore

```
protected IFlowCredentialsStoreService _credentialsStore
```

Field Value

IFlowCredentialsStoreService

_flowCommunicationService

```
protected ICloudCommunicationService _flowCommunicationService
```

Field Value

ICloudCommunicationService

Methods

GetCredential(Guid)

```
public Task<IResult<Credential>> GetCredential(Guid credentialId)
```

Parameters

`credentialId` Guid

Returns

Task<IResult<Credential>>

Class FlowCredentialsStoreService

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class FlowCredentialsStoreService : IFlowCredentialsStoreService
```

Inheritance

object ← FlowCredentialsStoreService

Implements

IFlowCredentialsStoreService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

FILE_EXTENSION

```
public const string FILE_EXTENSION = "encf"
```

Field Value

string

StorePath

```
protected static readonly string StorePath
```

Field Value

string

Methods

CreateCredential(Credential)

```
public IResult<Credential> CreateCredential(Credential credential)
```

Parameters

`credential` Credential

Returns

IResult<Credential>

CredentialExist(Guid)

```
public bool CredentialExist(Guid credentialId)
```

Parameters

`credentialId` Guid

Returns

bool

GetCredential(Guid)

```
public IResult<Credential> GetCredential(Guid credentialId)
```

Parameters

`credentialId` Guid

Returns

IResult<Credential>

Class FlowPathHelpers

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public static class FlowPathHelpers
```

Inheritance

object ← FlowPathHelpers

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

FLOWS_ASSEMBLIES_STORAGE_PATH

```
public static readonly string FLOWS_ASSEMBLIES_STORAGE_PATH
```

Field Value

string

FLOWS_EXECUTION_PATH

```
public static readonly string FLOWS_EXECUTION_PATH
```

Field Value

string

FLOWS_STORAGE_PATH

```
public static readonly string FLOWS_STORAGE_PATH
```

Field Value

string

REMOTE_SESSION_PATH

```
public static readonly string REMOTE_SESSION_PATH
```

Field Value

string

Properties

HOST_LOGS_PATH

```
public static string HOST_LOGS_PATH { get; }
```

Property Value

string

REMOTE_SESSION_LOGS_PATH

```
public static string REMOTE_SESSION_LOGS_PATH { get; }
```

Property Value

string

RUNTIME_LOGS_PATH

```
public static string RUNTIME_LOGS_PATH { get; }
```

Property Value

string

ResourceStoragePath

```
public static string ResourceStoragePath { get; }
```

Property Value

string

Methods

DeleteAllItemInFolder(string)

```
public static IResult DeleteAllItemInFolder(string path)
```

Parameters

`path` string

Returns

IResult

EnsureDataDirectory()

```
public static void EnsureDataDirectory()
```

FlowAssembliesStoragePath()

```
public static string FlowAssembliesStoragePath()
```

Returns

string

FlowExecutionPath(Guid)

```
public static string FlowExecutionPath(Guid flowId)
```

Parameters

`flowId` Guid

Returns

string

FlowJsonFileExecutionPath(Guid)

```
public static string FlowJsonFileExecutionPath(Guid flowId)
```

Parameters

`flowId` Guid

Returns

string

FlowJsonFileStatePath(Guid)

```
public static string FlowJsonFileStatePath(Guid flowId)
```

Parameters

`flowId` Guid

Returns

string

FlowJsonFileStoragePath(Guid)

```
public static string FlowJsonFileStoragePath(Guid flowId)
```

Parameters

`flowId` Guid

Returns

string

FlowLogsPath(Guid)

```
public static string FlowLogsPath(Guid flowId)
```

Parameters

`flowId` Guid

Returns

string

FlowStoragePath(Guid)

```
public static string FlowStoragePath(Guid flowId)
```

Parameters

`flowId` Guid

Returns

string

FlowZipFilePath(Guid)

```
public static string FlowZipFilePath(Guid flowId)
```

Parameters

`flowId` Guid

Returns

string

GetDirectoryName(string)

```
public static string GetDirectoryName(string path)
```

Parameters

`path` string

Returns

string

RemoteSessionFlowAssembliesStoragePath(Guid)

```
public static string RemoteSessionFlowAssembliesStoragePath(Guid sessionId)
```

Parameters

`sessionId` Guid

Returns

string

RemoteSessionFlowExecutionPath(Guid, Guid)

```
public static string RemoteSessionFlowExecutionPath(Guid sessionId, Guid flowId)
```

Parameters

`sessionId` Guid

`flowId` Guid

Returns

string

RemoteSessionFlowJsonFileExecutionPath(Guid, Guid)

```
public static string RemoteSessionFlowJsonFileExecutionPath(Guid sessionId,
Guid flowId)
```

Parameters

`sessionId` Guid

`flowId` Guid

Returns

string

RemoteSessionFlowJsonFileStoragePath(Guid, Guid)

```
public static string RemoteSessionFlowJsonFileStoragePath(Guid sessionId,
Guid flowId)
```

Parameters

`sessionId` Guid

`flowId` Guid

Returns

string

RemoteSessionFlowStoragePath(Guid, Guid)

```
public static string RemoteSessionFlowStoragePath(Guid sessionId, Guid flowId)
```

Parameters

`sessionId` Guid

`flowId` Guid

Returns

string

RemoteSessionFlowZipFilePath(Guid, Guid)

```
public static string RemoteSessionFlowZipFilePath(Guid sessionId, Guid flowId)
```

Parameters

`sessionId` Guid

`flowId` Guid

Returns

string

RemoteSessionLogsPath(Guid)

```
public static string RemoteSessionLogsPath(Guid sessionId)
```

Parameters

`sessionId` Guid

Returns

string

RemoteSessionStoragePath(Guid)

```
public static string RemoteSessionStoragePath(Guid sessionId)
```

Parameters

`sessionId` Guid

Returns

string

Class FlowResourceManagerService

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class FlowResourceManagerService : IFlowResourceManagerService
```

Inheritance

object ← FlowResourceManagerService

Implements

IFlowResourceManagerService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowResourceManagerService(ICloudCommunicationService, IFlowResourceStoreService)

```
public FlowResourceManagerService(ICloudCommunicationService
flowCommunicationService, IFlowResourceStoreService resourceStoreService)
```

Parameters

`flowCommunicationService` ICloudCommunicationService

`resourceStoreService` IFlowResourceStoreService

Fields

_flowCommunicationService

```
protected ICloudCommunicationService _flowCommunicationService
```

Field Value

ICloudCommunicationService

_resourceStoreService

```
protected IFlowResourceStoreService _resourceStoreService
```

Field Value

IFlowResourceStoreService

Methods

GetResource(Guid, int)

```
public Task<IResult<(string file, ResourceItem resourceItem)>> GetResource(Guid
resourceId, int retry = 3)
```

Parameters

`resourceId` Guid

`retry` int

Returns

Task<IResult<(string file, ResourceItem resourceItem)>>

Class FlowResourceStoreService

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class FlowResourceStoreService : IFlowResourceStoreService
```

Inheritance

object ← FlowResourceStoreService

Implements

IFlowResourceStoreService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Create(ResourceItem, Stream)

```
public bool Create(ResourceItem resourceItem, Stream data)
```

Parameters

`resourceItem` ResourceItem

`data` Stream

Returns

bool

Exists(ResourceItem)

```
public bool Exists(ResourceItem resourceItem)
```

Parameters

`resourceItem` ResourceItem

Returns

bool

GetById(Guid)

```
public IResult<(string file, ResourceItem resourceItem)> GetById(Guid resourceId)
```

Parameters

`resourceId` Guid

Returns

IResult<(string file, ResourceItem resourceItem)>

GetResourcePaths(ResourceItem)

```
public static (string dataFile, string metadataFileById, string metadataFileByName)
GetResourcePaths(ResourceItem resourceItem)
```

Parameters

`resourceItem` ResourceItem

Returns

(string dataFile, string metadataFileById, string metadataFileByName)

Class FlowUrlResolverService

Namespace: Crosser.EdgeNode.Flows.Communication

Assembly: Crosser.EdgeNode.Flows.Communication.dll

```
public class FlowUrlResolverService
```

Inheritance

object ← FlowUrlResolverService

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowUrlResolverService(IFlowConfigurationService)

```
public FlowUrlResolverService(IFlowConfigurationService configurationService)
```

Parameters

`configurationService` IFlowConfigurationService

Properties

LoginUrl

```
public string LoginUrl { get; }
```

Property Value

string

Methods

CollectResourceDataUrl(Guid)

```
public string CollectResourceDataUrl(Guid resourceItemId)
```

Parameters

`resourceItemId` Guid

Returns

string

CollectResourceMetadataUrl(Guid)

```
public string CollectResourceMetadataUrl(Guid resourceId)
```

Parameters

`resourceId` Guid

Returns

string

DownloadCredentialUrl(Guid)

```
public string DownloadCredentialUrl(Guid credentialId)
```

Parameters

`credentialId` Guid

Returns

string

Namespace Crosser.EdgeNode.Flows.Interactivity

Classes

FlowInteractivity

InteractivePublishFrame<T>

InteractiveSessionFrameHelpers

InteractiveSessionRequest
Information about what Uri and Method to call

InteractiveSessionRequest<T>

IspMessage

IspMethods

IspMethods.Flow

IspMethods.Node

IspResponseErrorMessage

IspResponseErrorMessage<T>

IspResponseSuccessMessage

IspResponseSuccessMessage<T>

ModelHelpers

NodeInitializeResult
Response object when the client asks for Server interactivity capabilities

PublishMessage<T>

ResponseError

ResponseError<T>

ResponseSuccess

ResponseSuccess<T>

ServerInformation

Class FlowInteractivity

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record FlowInteractivity : IEquatable<FlowInteractivity>
```

Inheritance

object ← FlowInteractivity

Implements

IEquatable<FlowInteractivity>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Methods

```
public required IReadOnlyCollection<MethodInformation> Methods { get; init; }
```

Property Value

IReadOnlyCollection<MethodInformation>

Modules

```
public required IReadOnlyCollection<ModuleInteractivity> Modules { get; init; }
```

Property Value

IReadOnlyCollection<ModuleInteractivity>

Uri

```
public required string Uri { get; init; }
```

Property Value

string

Class InteractivePublishFrame<T>

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record InteractivePublishFrame<T> : IEquatable<InteractivePublishFrame<T>>
```

Type Parameters

T

Inheritance

object ← InteractivePublishFrame<T>

Implements

IEquatable<InteractivePublishFrame<T>>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

InteractivePublishFrame(string, T)

```
public InteractivePublishFrame(string channel, T data)
```

Parameters

channel string

data T

Properties

channel

```
public string channel { get; set; }
```

Property Value

string

message

```
public PublishMessage<T> message { get; set; }
```

Property Value

PublishMessage<T>

Class InteractiveSessionFrameHelpers

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public static class InteractiveSessionFrameHelpers
```

Inheritance

object ← InteractiveSessionFrameHelpers

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetFlowIdFromInteractivityParameters(JsonElement)

```
public static IResult<Guid> GetFlowIdFromInteractivityParameters(JsonElement
jsonElement)
```

Parameters

`jsonElement` JsonElement

Returns

IResult<Guid>

GetFlowInformationFromFrame(JsonElement)

```
public static IResult<(string flow, Guid flowId)>
GetFlowInformationFromFrame(JsonElement flowFrame)
```

Parameters

`flowFrame` JsonElement

Returns

IResult<(string flow, Guid flowId)>

GetIspWebSocketPublishFrame(ResponseError, Guid, string)

```
public static string GetIspWebSocketPublishFrame(this ResponseError message, Guid
sessionId, string publishChannel)
```

Parameters

`message` ResponseError

`sessionId` Guid

`publishChannel` string

Returns

string

GetIspWebSocketPublishFrame(ResponseSuccess, Guid, string)

```
public static string GetIspWebSocketPublishFrame(this ResponseSuccess message, Guid
sessionId, string publishChannel)
```

Parameters

`message` ResponseSuccess

`sessionId` Guid

`publishChannel` string

Returns

string

GetIspWebSocketPublishFrame<T>(ResponseError<T>, Guid, string)

```
public static string GetIspWebSocketPublishFrame<T>(this ResponseError<T> message,
Guid sessionId, string publishChannel)
```

Parameters

`message` ResponseError<T>

`sessionId` Guid

`publishChannel` string

Returns

string

Type Parameters

`T`

GetIspWebSocketPublishFrame<T>(ResponseSuccess<T>, Guid, string)

```
public static string GetIspWebSocketPublishFrame<T>(this ResponseSuccess<T> message,
Guid sessionId, string publishChannel)
```

Parameters

`message` ResponseSuccess<T>

`sessionId` Guid

`publishChannel` string

Returns

string

Type Parameters

T

Class InteractiveSessionRequest

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

Information about what Uri and Method to call

```
public record InteractiveSessionRequest : IspMessage, IEquatable<IspMessage>,
IEquatable<InteractiveSessionRequest>
```

Inheritance

object ← IspMessage ← InteractiveSessionRequest

Implements

IEquatable<IspMessage>, IEquatable<InteractiveSessionRequest>

Inherited Members

IspMessage.sessionId , IspMessage.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

InteractiveSessionRequest(Guid)

Information about what Uri and Method to call

```
public InteractiveSessionRequest(Guid sessionId)
```

Parameters

`sessionId` Guid

Properties

Id

If not provided this is considered to be a notification instead of a request

```
public Guid? Id { get; init; }
```

Property Value

Guid?

Method

```
public required string Method { get; set; }
```

Property Value

string

Parameters

```
public object? Parameters { get; init; }
```

Property Value

object

Uri

```
public required string Uri { get; set; }
```

Property Value

string

Class InteractiveSessionRequest<T>

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record InteractiveSessionRequest<T> : IspMessage, IEquatable<IspMessage>,
IEquatable<InteractiveSessionRequest<T>>
```

Type Parameters

T

Inheritance

object ← IspMessage ← InteractiveSessionRequest<T>

Implements

IEquatable<IspMessage>, IEquatable<InteractiveSessionRequest<T>>

Inherited Members

IspMessage.sessionId , IspMessage.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

InteractiveSessionRequest(Guid)

```
public InteractiveSessionRequest(Guid sessionId)
```

Parameters

sessionId Guid

Properties

Id

```
public required Guid Id { get; init; }
```

Property Value

Guid

Method

```
public required string Method { get; set; }
```

Property Value

string

Parameters

```
public T? Parameters { get; init; }
```

Property Value

T

Uri

```
public required string Uri { get; set; }
```

Property Value

string

Class IspMessage

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record IspMessage : IEquatable<IspMessage>
```

Inheritance

object ← IspMessage

Implements

IEquatable<IspMessage>

Derived

InteractiveSessionRequest, InteractiveSessionRequest<T>, IspResponseErrorMessage, IspResponseErrorMessage<T>, IspResponseSuccessMessage, IspResponseSuccessMessage<T>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

IspMessage(Guid)

```
public IspMessage(Guid sessionId)
```

Parameters

`sessionId` Guid

Properties

Version

```
public string Version { get; init; }
```

Property Value

string

sessionId

```
public Guid sessionId { get; init; }
```

Property Value

Guid

Class IspMethods

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public static class IspMethods
```

Inheritance

object ← IspMethods

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Class IspMethods.Flow

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public static class IspMethods.Flow
```

Inheritance

object ← IspMethods.Flow

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

CloseInteractive

```
public const string CloseInteractive = "flow/closeInteractive"
```

Field Value

string

InitializeInteractive

```
public const string InitializeInteractive = "flow/initializeInteractive"
```

Field Value

string

Interactivity

```
public const string Interactivity = "flow/interactivity"
```

Field Value

string

Class IspMethods.Node

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public static class IspMethods.Node
```

Inheritance

object ← IspMethods.Node

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Fields

Initialize

```
public const string Initialize = "initialize"
```

Field Value

string

NodeUri

```
public const string NodeUri = "/node"
```

Field Value

string

Class IspResponseErrorMessage

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record IspResponseErrorMessage : IspMessage, IEquatable<IspMessage>,
IEquatable<IspResponseErrorMessage>
```

Inheritance

object ← IspMessage ← IspResponseErrorMessage

Implements

IEquatable<IspMessage>, IEquatable<IspResponseErrorMessage>

Inherited Members

IspMessage.sessionId , IspMessage.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

IspResponseErrorMessage(ResponseError, Guid)

```
public IspResponseErrorMessage(ResponseError Result, Guid sessionId)
```

Parameters

`Result` ResponseError

`sessionId` Guid

Properties

Result

```
public ResponseError Result { get; init; }
```

Property Value

ResponseError

Class IspResponseErrorMessage<T>

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record IspResponseErrorMessage<T> : IspMessage, IEquatable<IspMessage>,
IEquatable<IspResponseErrorMessage<T>>
```

Type Parameters

T

Inheritance

object ← IspMessage ← IspResponseErrorMessage<T>

Implements

IEquatable<IspMessage>, IEquatable<IspResponseErrorMessage<T>>

Inherited Members

IspMessage.sessionId , IspMessage.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

IspResponseErrorMessage(ResponseError<T>, Guid)

```
public IspResponseErrorMessage(ResponseError<T> Result, Guid sessionId)
```

Parameters

Result ResponseError<T>

sessionId Guid

Properties

Result

```
public ResponseError<T> Result { get; init; }
```

Property Value

ResponseError<T>

Class IspResponseSuccessMessage

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record IspResponseSuccessMessage : IspMessage, IEquatable<IspMessage>,
IEquatable<IspResponseSuccessMessage>
```

Inheritance

object ← IspMessage ← IspResponseSuccessMessage

Implements

IEquatable<IspMessage>, IEquatable<IspResponseSuccessMessage>

Inherited Members

IspMessage.sessionId , IspMessage.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

IspResponseSuccessMessage(ResponseSuccess, Guid)

```
public IspResponseSuccessMessage(ResponseSuccess Result, Guid sessionId)
```

Parameters

`Result` ResponseSuccess

`sessionId` Guid

Properties

Result

```
public ResponseSuccess Result { get; init; }
```

Property Value

ResponseSuccess

Class IspResponseSuccessMessage<T>

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record IspResponseSuccessMessage<T> : IspMessage, IEquatable<IspMessage>,
IEquatable<IspResponseSuccessMessage<T>>
```

Type Parameters

`T`

Inheritance

object ← IspMessage ← IspResponseSuccessMessage<T>

Implements

IEquatable<IspMessage>, IEquatable<IspResponseSuccessMessage<T>>

Inherited Members

IspMessage.sessionId , IspMessage.Version , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

IspResponseSuccessMessage(ResponseSuccess<T>, Guid)

```
public IspResponseSuccessMessage(ResponseSuccess<T> Result, Guid sessionId)
```

Parameters

`Result` ResponseSuccess<T>

`sessionId` Guid

Properties

Result

```
public ResponseSuccess<T> Result { get; init; }
```

Property Value

ResponseSuccess<T>

Class ModelHelpers

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public static class ModelHelpers
```

Inheritance

object ← ModelHelpers

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

GetInitializeNodeResponse()

```
public static NodeInitializeResult GetInitializeNodeResponse()
```

Returns

NodeInitializeResult

GetInteractiveSessionCommand(JsonElement)

```
public static InteractiveSessionRequest GetInteractiveSessionCommand(this
JsonElement jsonElement)
```

Parameters

`jsonElement` JsonElement

Returns

InteractiveSessionRequest

Class NodeInitializeResult

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

Response object when the client asks for Server interactivity capabilities

```
public record NodeInitializeResult : IEquatable<NodeInitializeResult>
```

Inheritance

object ← NodeInitializeResult

Implements

IEquatable<NodeInitializeResult>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Methods

```
public required IReadOnlyCollection<MethodInformation> Methods { get; init; }
```

Property Value

IReadOnlyCollection<MethodInformation>

ServerInformation

```
public required ServerInformation ServerInformation { get; init; }
```

Property Value

ServerInformation

Uri

```
public static string Uri { get; }
```

Property Value

string

Class PublishMessage<T>

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record PublishMessage<T> : IEquatable<PublishMessage<T>>
```

Type Parameters

T

Inheritance

object ← PublishMessage<T>

Implements

IEquatable<PublishMessage<T>>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

PublishMessage(T, string)

```
public PublishMessage(T data, string type)
```

Parameters

data T

type string

Properties

data

```
public T data { get; init; }
```

Property Value

T

type

```
public string type { get; init; }
```

Property Value

string

Class ResponseError

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record ResponseError : IEquatable<ResponseError>
```

Inheritance

object ← ResponseError

Implements

IEquatable<ResponseError>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Extension Methods

InteractiveSessionFrameHelpers.GetIspWebSocketPublishFrame(ResponseError, Guid, string)

Constructors

ResponseError(FlowError, Guid)

```
public ResponseError(FlowError error, Guid id)
```

Parameters

`error` FlowError

`id` Guid

Properties

error

```
public FlowError error { get; init; }
```

Property Value

FlowError

id

```
public Guid id { get; init; }
```

Property Value

Guid

Class ResponseError<T>

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record ResponseError<T> : IEquatable<ResponseError<T>>
```

Type Parameters

T

Inheritance

object ← ResponseError<T>

Implements

IEquatable<ResponseError<T>>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Extension Methods

InteractiveSessionFrameHelpers.GetIspWebSocketPublishFrame<T>(ResponseError<T>, Guid, string)

Constructors

ResponseError(FlowError<T>, Guid)

```
public ResponseError(FlowError<T> error, Guid id)
```

Parameters

error FlowError<T>

id Guid

Properties

error

```
public FlowError<T> error { get; init; }
```

Property Value

FlowError<T>

id

```
public Guid id { get; init; }
```

Property Value

Guid

Class ResponseSuccess

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record ResponseSuccess : IEquatable<ResponseSuccess>
```

Inheritance

object ← ResponseSuccess

Implements

IEquatable<ResponseSuccess>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Extension Methods

InteractiveSessionFrameHelpers.GetIspWebSocketPublishFrame(ResponseSuccess, Guid, string)

Constructors

ResponseSuccess(object, Guid)

```
public ResponseSuccess(object value, Guid id)
```

Parameters

`value` object

`id` Guid

Properties

id

```
public Guid id { get; init; }
```

Property Value

Guid

value

```
public object value { get; init; }
```

Property Value

object

Class ResponseSuccess<T>

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record ResponseSuccess<T> : IEquatable<ResponseSuccess<T>>
```

Type Parameters

`T`

Inheritance

object ← ResponseSuccess<T>

Implements

IEquatable<ResponseSuccess<T>>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Extension Methods

InteractiveSessionFrameHelpers.GetIspWebSocketPublishFrame<T>(ResponseSuccess<T>, Guid, string)

Constructors

ResponseSuccess(T, Guid)

```
public ResponseSuccess(T value, Guid id)
```

Parameters

`value` T

`id` Guid

Properties

id

```
public Guid id { get; init; }
```

Property Value

Guid

value

```
public T value { get; init; }
```

Property Value

T

Class ServerInformation

Namespace: Crosser.EdgeNode.Flows.Interactivity

Assembly: Crosser.EdgeNode.Flows.Interactivity.dll

```
public record ServerInformation : IEquatable<ServerInformation>
```

Inheritance

object ← ServerInformation

Implements

IEquatable<ServerInformation>

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Name

```
public required string Name { get; set; }
```

Property Value

string

Version

```
public required string Version { get; set; }
```

Property Value

string

Namespace Crosser.EdgeNode.Flows.Metrics

Classes

CrosserModuleEventSource

DynamicEventIncrementPollingCounterMapper

DynamicEventPollingCounterMapper

Class CrosserModuleEventSource

Namespace: Crosser.EdgeNode.Flows.Metrics

Assembly: Crosser.EdgeNode.Flows.dll

```
[EventSource(Name = "Crosser.EdgeNode.Module")]
public sealed class CrosserModuleEventSource : EventSource, IDisposable
```

Inheritance

object ← EventSource ← CrosserModuleEventSource

Implements

IDisposable

Inherited Members

EventSource.Dispose() , EventSource.GenerateManifest(Type, string) , EventSource.GenerateManifest(Type, string, EventManifestOptions) , EventSource.GetGuid(Type) , EventSource.GetName(Type) , EventSource.GetSources() , EventSource.GetTrait(string) , EventSource.IsEnabled() , EventSource.IsEnabled(EventLevel, EventKeywords) , EventSource.IsEnabled(EventLevel, EventKeywords, EventChannel) , EventSource.SendCommand(EventSource, EventCommand, IDictionary<string, string>) , EventSource.SetCurrentThreadActivityId(Guid) , EventSource.SetCurrentThreadActivityId(Guid, out Guid) , EventSource.ToString() , EventSource.Write(string) , EventSource.Write(string, EventSourceOptions) , EventSource.Write<T>(string, EventSourceOptions, T) , EventSource.Write<T>(string, ref EventSourceOptions, ref Guid, ref Guid, ref T) , EventSource.Write<T>(string, ref EventSourceOptions, ref T) , EventSource.Write<T>(string, T) , EventSource.ConstructionException , EventSource.CurrentThreadActivityId , EventSource.Guid , EventSource.Name , EventSource.Settings , EventSource.EventCommandExecuted , object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.ReferenceEquals(object, object)

Fields

Log

```
public static readonly CrosserModuleEventSource Log
```

Field Value

CrosserModuleEventSource

Methods

DecrementModuleQueue(Guid)

```
[NonEvent]
public void DecrementModuleQueue(Guid id)
```

Parameters

`id` Guid

IncrementModuleDroppedMessages(Guid)

```
[NonEvent]
public void IncrementModuleDroppedMessages(Guid id)
```

Parameters

`id` Guid

IncrementModuleQueue(Guid)

```
[NonEvent]
public void IncrementModuleQueue(Guid id)
```

Parameters

`id` Guid

ModuleError(Guid)

```
[NonEvent]
public void ModuleError(Guid id)
```

Parameters

`id` Guid

ModuleWarning(Guid)

```
[NonEvent]
public void ModuleWarning(Guid id)
```

Parameters

`id` Guid

RegisterCounter(IFlowModule)

```
[NonEvent]
public void RegisterCounter(IFlowModule module)
```

Parameters

`module` IFlowModule

Class DynamicEventIncrementPollingCounterMapper

Namespace: Crosser.EdgeNode.Flows.Metrics

Assembly: Crosser.EdgeNode.Flows.dll

```
public class DynamicEventIncrementPollingCounterMapper : IDisposable
```

Inheritance

object ← DynamicEventIncrementPollingCounterMapper

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

DynamicEventIncrementPollingCounterMapper(EventSource, string, IFlowModule)

```
public DynamicEventIncrementPollingCounterMapper(EventSource eventSource, string
prefix, IFlowModule module)
```

Parameters

`eventSource` EventSource

`prefix` string

`module` IFlowModule

Methods

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Dispose(bool)

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

Increment()

```
public void Increment()
```

Class DynamicEventPollingCounterMapper

Namespace: Crosser.EdgeNode.Flows.Metrics

Assembly: Crosser.EdgeNode.Flows.dll

```
public class DynamicEventPollingCounterMapper : IDisposable
```

Inheritance

object ← DynamicEventPollingCounterMapper

Implements

IDisposable

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

DynamicEventPollingCounterMapper(EventSource, string, IFlowModule)

```
public DynamicEventPollingCounterMapper(EventSource eventSource, string prefix,
IFlowModule module)
```

Parameters

`eventSource` EventSource

`prefix` string

`module` IFlowModule

Methods

Decrement()

```
public void Decrement()
```

Dispose()

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

```
public void Dispose()
```

Dispose(bool)

```
protected virtual void Dispose(bool disposing)
```

Parameters

`disposing` bool

Increment()

```
public void Increment()
```

Namespace Crosser.EdgeNode.Flows.Models.Abstractions.Models

Classes

ChannelFullModeJsonConverter

DynamicObjectJsonConverter

FlowMessage

This is a customized version of the .NET System.Dynamic.DynamicObject. The object is mainly used to pass data between modules in a flow with the possibility to change/remove/add properties at runtime

FlowMessageArrayJsonConverter

FlowMessageExtensions

Helpers for FlowMessage

FlowMessageJsonConverter

FlowMessageJsonSerializer

The default JSON serializer

IFlowMessageJsonConverter

JsonSerializer

Wrapper for easier serialization

Class ChannelFullModeJsonConverter

Namespace: Crosser.EdgeNode.Flows.Models.Abstractions.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class ChannelFullModeJsonConverter : JsonConverter<BoundedChannelFullMode>
```

Inheritance

object ← JsonConverter ← JsonConverter<BoundedChannelFullMode> ← ChannelFullModeJsonConverter

Inherited Members

JsonConverter<BoundedChannelFullMode>.CanConvert(Type) ,
JsonConverter<BoundedChannelFullMode>.ReadAsPropertyName(ref Utf8JsonReader, Type, JsonSerializerOptions) ,
JsonConverter<BoundedChannelFullMode>.WriteAsPropertyName(Utf8JsonWriter, BoundedChannelFullMode, JsonSerializerOptions) ,
JsonConverter<BoundedChannelFullMode>.HandleNull ,
JsonConverter<BoundedChannelFullMode>.Type , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Read(ref Utf8JsonReader, Type, JsonSerializerOptions)

Reads and converts the JSON to type BoundedChannelFullMode.

```
public override BoundedChannelFullMode Read(ref Utf8JsonReader reader, Type type,
JsonSerializerOptions options)
```

Parameters

`reader` Utf8JsonReader

The reader.

`type` Type

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Returns

BoundedChannelFullMode

The converted value.

Write(Utf8JsonWriter, BoundedChannelFullMode, JsonSerializerOptions)

Writes a specified value as JSON.

```
public override void Write(Utf8JsonWriter writer, BoundedChannelFullMode value,
JsonSerializerOptions options)
```

Parameters

`writer` Utf8JsonWriter

The writer to write to.

`value` BoundedChannelFullMode

The value to convert to JSON.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Class DynamicObjectJsonConverter

Namespace: Crosser.EdgeNode.Flows.Models.Abstractions.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class DynamicObjectJsonConverter : JsonConverter<dynamic>
```

Inheritance

object ← JsonConverter ← JsonConverter<dynamic> ← DynamicObjectJsonConverter

Inherited Members

JsonConverter<dynamic>.CanConvert(Type) ,
JsonConverter<dynamic>.ReadAsPropertyName(ref Utf8JsonReader, Type, JsonSerializerOptions) ,
JsonConverter<dynamic>.WriteAsPropertyName(Utf8JsonWriter, dynamic, JsonSerializerOptions) ,
JsonConverter<dynamic>.HandleNull , JsonConverter<dynamic>.Type , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Read(ref Utf8JsonReader, Type, JsonSerializerOptions)

Reads and converts the JSON to type `T`.

```
public override dynamic? Read(ref Utf8JsonReader reader, Type typeToConvert,
JsonSerializerOptions options)
```

Parameters

`reader` Utf8JsonReader

The reader.

`typeToConvert` Type

The type to convert.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Returns

dynamic

The converted value.

Write(Utf8JsonWriter, dynamic, JsonSerializerOptions)

Writes a specified value as JSON.

```
public override void Write(Utf8JsonWriter writer, dynamic value,
JsonSerializerOptions options)
```

Parameters

`writer` Utf8JsonWriter

The writer to write to.

`value` dynamic

The value to convert to JSON.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Class FlowMessage

Namespace: Crosser.EdgeNode.Flows.Models.Abstractions.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

This is a customized version of the .NET System.Dynamic.DynamicObject. The object is mainly used to pass data between modules in a flow with the possibility to change/remove/add properties at runtime

```
[JsonConverter(typeof(FlowMessageJsonConverter))]
public class FlowMessage : DynamicObject, IFlowMessage, IDynamicMetaObjectProvider
```

Inheritance

object ← DynamicObject ← FlowMessage

Implements

IFlowMessage, IDynamicMetaObjectProvider

Inherited Members

DynamicObject.GetMetaObject(Expression) ,
DynamicObject.TryBinaryOperation(BinaryOperationBinder, object, out object) ,
DynamicObject.TryConvert(ConvertBinder, out object) ,
DynamicObject.TryCreateInstance(CreateInstanceBinder, object[], out object) ,
DynamicObject.TryDeleteIndex(DeleteIndexBinder, object[]) ,
DynamicObject.TryDeleteMember(DeleteMemberBinder) ,
DynamicObject.TryGetIndex(GetIndexBinder, object[], out object) ,
DynamicObject.TryInvoke(InvokeBinder, object[], out object) ,
DynamicObject.TryInvokeMember(InvokeMemberBinder, object[], out object) ,
DynamicObject.TrySetIndex(SetIndexBinder, object[], object) ,
DynamicObject.TryUnaryOperation(UnaryOperationBinder, out object) , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object)

Extension Methods

MessageFilterExtensions.FilterApplyAll(IFlowMessage, List<MessageFilter>) ,
MessageFilterExtensions.FilterApplyAny(IFlowMessage, List<MessageFilter>) ,
FlowMessageExtensions.DeepCopy(IFlowMessage) ,
FlowMessageExtensions.GetPropertyLevel(IFlowMessage, string, string) ,
FlowMessageExtensions.GetPropertyPath(IFlowMessage, string, string) ,
FlowMessageExtensions.RemoveChangedProperties(IFlowMessage, IFlowMessage) ,
FlowMessageExtensions.Serialize(IFlowMessage) , FlowMessageExtensions.Serialize(FlowMessage)

Examples

You can build any object by creating a new FlowMessage as a dynamic and just set properties

```
dynamic d = new FlowMessage();
d.name = "Harry The Hippo";
d.age = 35;
```

You can also build object with dot-notation.

```
dynamic o = new FlowMessage();
p.person.name = "Harry The Hippo";
d.person.age = 35;
```

Constructors

FlowMessage()

This constructor just works off the internal dictionary and any public properties of this object.

Note you can subclass FlowMessage.

```
public FlowMessage()
```

Fields

ALLOWED_CHARS

```
protected static readonly string ALLOWED_CHARS
```

Field Value

string

MESSAGE_PROPERTY

```
public const string MESSAGE_PROPERTY = "crosser.message"
```

Field Value

string

REGEX_ENDS_WITH_INDEX

```
protected const string REGEX_ENDS_WITH_INDEX = "\\[\\d+\\]$"
```

Field Value

string

REGEX_FIND_TEMPLATES

```
protected static readonly string REGEX_FIND_TEMPLATES
```

Field Value

string

REGEX_FIND_TEMPLATE_CONTENT

```
protected static readonly string REGEX_FIND_TEMPLATE_CONTENT
```

Field Value

string

REGEX_GET_ALL_INDEX

```
protected static readonly string REGEX_GET_ALL_INDEX
```

Field Value

string

REGEX_HAS_INDEX

```
protected const string REGEX_HAS_INDEX = "(\\[\\d+\\]*)+$"
```

Field Value

string

REGEX_VALID_PROPERTY

```
protected static readonly string REGEX_VALID_PROPERTY
```

Field Value

string

SUCCESS_PROPERTY

```
public const string SUCCESS_PROPERTY = "crosser.success"
```

Field Value

string

Properties

ENVIRONMENT_NAMES_SUPPORTED

A read-only dictionary that exposes the ENV_VARs available to use with template syntax.

```
public static IReadOnlyDictionary<string, string> ENVIRONMENT_NAMES_SUPPORTED {
get; }
```

Property Value

IReadOnlyDictionary<string, string>

ErrorMessage

```
public string ErrorMessage { get; }
```

Property Value

string

IsError

```
public bool IsError { get; }
```

Property Value

bool

IsSuccess

```
public bool IsSuccess { get; }
```

Property Value

bool

this[int]

```
public object? this[int index] { get; }
```

Parameters

`index` int

Property Value

object

this[string]

```
public object? this[string key] { get; set; }
```

Parameters

`key` string

Property Value

object

Properties

```
public Dictionary<string, object?> Properties { get; set; }
```

Property Value

Dictionary<string, object>

_properties

```
public ImmutableSortedDictionary<string, object?> _properties { get; set; }
```

Property Value

ImmutableSortedDictionary<string, object>

Methods

AddAllProperties<T>(IFlowMessage?, string, T?, bool, bool)

```
[Obsolete("Use Set<T> instead")]
public void AddAllProperties<T>(IFlowMessage? message, string path, T? value, bool
allowConvertToFlowMessage = true, bool autoCreate = true)
```

Parameters

`message` IFlowMessage

`path` string

`value` T

`allowConvertToFlowMessage` bool

`autoCreate` bool

Type Parameters

`T`

CanBeConvertedToFlowMessage(object?)

```
public static bool CanBeConvertedToFlowMessage(object? o)
```

Parameters

`o` object

Returns

bool

Clear()

```
public void Clear()
```

Clone()

Will return a new FlowMessage based on the properties dynamically added to the current object

```
public IFlowMessage Clone()
```

Returns

IFlowMessage

ConvertTo(object?, Type)

```
public static dynamic? ConvertTo(object? obj, Type type)
```

Parameters

`obj` object

`type` Type

Returns

dynamic

ConvertTo<T>(object?)

```
public static T? ConvertTo<T>(object? obj)
```

Parameters

`obj` object

Returns

T

Type Parameters

`T`

Deserialize(string)

```
[Obsolete("USe Parse or TryParse instead")]
public static IFlowMessage Deserialize(string json)
```

Parameters

`json` string

Returns

IFlowMessage

GetByTemplate(string)

```
public IResult<string> GetByTemplate(string template)
```

Parameters

`template` string

Returns

IResult<string>

GetByTemplate(string, Type)

```
public IResult<dynamic?> GetByTemplate(string template, Type type)
```

Parameters

`template` string

`type` Type

Returns

IResult<dynamic>

GetByTemplate<T>(string)

```
public IResult<T?> GetByTemplate<T>(string template)
```

Parameters

`template` string

Returns

IResult<T>

Type Parameters

`T`

GetDynamicMemberNames()

Returns the enumeration of all dynamic member names.

```
public override IEnumerable<string> GetDynamicMemberNames()
```

Returns

IEnumerable<string>

A sequence that contains dynamic member names.

GetProperties()

```
public IEnumerable<KeyValuePair<string, object?>> GetProperties()
```

Returns

IEnumerable<KeyValuePair<string, object>>

GetProperty<T>(IFlowMessage, string)

```
[Obsolete("Use Get<T>")]
public T? GetProperty<T>(IFlowMessage message, string path)
```

Parameters

`message` IFlowMessage

`path` string

Returns

T

Type Parameters

`T`

GetValue<T>(string)

```
public T? GetValue<T>(string path)
```

Parameters

`path` string

Returns

T

Type Parameters

T

Get<T>(IFlowMessage, string)

```
public T? Get<T>(IFlowMessage message, string path)
```

Parameters

message IFlowMessage

path string

Returns

T

Type Parameters

T

Get<T>(string)

```
public T? Get<T>(string path)
```

Parameters

path string

Returns

T

Type Parameters

`T`

Has(IFlowMessage, string)

```
public bool Has(IFlowMessage message, string path)
```

Parameters

`message` IFlowMessage

`path` string

Returns

bool

Has(string)

```
public bool Has(string path)
```

Parameters

`path` string

Returns

bool

HasOwnProperty(IFlowMessage, string)

```
[Obsolete("Use Has instead")]
public bool HasOwnProperty(IFlowMessage message, string path)
```

Parameters

`message` IFlowMessage

`path` string

Returns

bool

HasOwnProperty(string)

Check if the object has a property with a specific path defined

```
[Obsolete("Use Has instead")]
public bool HasOwnProperty(string path)
```

Parameters

`path` string

The property-name to look for

Returns

bool

HasOwnProperty<T>(IFlowMessage, string)

```
[Obsolete("Use Has<T> instead")]
public bool HasOwnProperty<T>(IFlowMessage message, string path)
```

Parameters

`message` IFlowMessage

`path` string

Returns

bool

Type Parameters

T

HasOwnProperty<T>(string)

Check if the object has a property with a specific path and type defined

```
[Obsolete("Use Has<T> instead")]
public bool HasOwnProperty<T>(string path)
```

Parameters

path string

Returns

bool

Type Parameters

T

The expected type of the property

Has<T>(IFlowMessage, string)

```
public bool Has<T>(IFlowMessage message, string path)
```

Parameters

message IFlowMessage

path string

Returns

bool

Type Parameters

T

Has<T>(string)

```
public bool Has<T>(string path)
```

Parameters

path string

Returns

bool

Type Parameters

T

IsComplex(object)

```
[Obsolete("Use CanBeConvertedToFlowMessage(object o) instead")]
public static bool IsComplex(object o)
```

Parameters

o object

Returns

bool

Parse(string)

```
public static IFlowMessage Parse(string json)
```

Parameters

`json` string

Returns

IFlowMessage

Parse(string, string)

Use this if you are not sure what kind of JSON you have. If JSOn is an array for example this is needed instead of the basic FlowMessage.Parse

```
public static IFlowMessage Parse(string path, string json)
```

Parameters

`path` string

`json` string

Returns

IFlowMessage

Exceptions

NullReferenceException

Remove(string)

```
public IFlowMessage Remove(string path)
```

Parameters

`path` string

Returns

IFlowMessage

RemoveProperty(string)

```
[Obsolete("Use Remove instead")]
public void RemoveProperty(string path)
```

Parameters

`path` string

Set(string, string, bool, bool)

```
public IFlowMessage Set(string path, string value, bool allowConvertToFlowMessage =
true, bool autoCreate = true)
```

Parameters

`path` string

`value` string

`allowConvertToFlowMessage` bool

`autoCreate` bool

Returns

IFlowMessage

SetError(string)

```
public void SetError(string message = "")
```

Parameters

`message` string

SetSuccess()

```
public void SetSuccess()
```

SetValue<T>(string, T?, bool, bool)

```
public void SetValue<T>(string path, T? value, bool allowConvertToFlowMessage =
true, bool autoCreate = true)
```

Parameters

`path` string

`value` T

`allowConvertToFlowMessage` bool

`autoCreate` bool

Type Parameters

`T`

Set<T>(string, T?, bool, bool)

```
public IFlowMessage Set<T>(string path, T? value, bool allowConvertToFlowMessage =
true, bool autoCreate = true)
```

Parameters

`path` string

`value` T

`allowConvertToFlowMessage` bool

`autoCreate` bool

Returns

IFlowMessage

Type Parameters

`T`

ToJSON()

```
public string ToJSON()
```

Returns

string

ToString()

Returns a string that represents the current object.

```
public override string ToString()
```

Returns

string

A string that represents the current object.

TryGetMember(GetMemberBinder, out object?)

Provides the implementation for operations that get member values. Classes derived from the Dynamic Object class can override this method to specify dynamic behavior for operations such as getting a value for a property.

```
public override bool TryGetMember(GetMemberBinder binder, out object? result)
```

Parameters

`binder` GetMemberBinder

Provides information about the object that called the dynamic operation. The `binder.Name` property provides the name of the member on which the dynamic operation is performed. For example, for the `Console.WriteLine(sampleObject.SampleProperty)` statement, where `sampleObject` is an instance of the class derived from the DynamicObject class, `binder.Name` returns "SampleProperty". The `binder.IgnoreCase` property specifies whether the member name is case-sensitive.

`result` object

The result of the get operation. For example, if the method is called for a property, you can assign the property value to `result`.

Returns

bool

true if the operation is successful; otherwise, false. If this method returns false, the run-time binder of the language determines the behavior. (In most cases, a run-time exception is thrown.)

TryParse(string, out IFlowMessage)

```
public static bool TryParse(string json, out IFlowMessage flowMessage)
```

Parameters

`json` string

`flowMessage` IFlowMessage

Returns

bool

TryParse(string, string, out IFlowMessage)

Use this if you are not sure what kind of JSON you have. If JSOn is an array for example this is needed instead of the basic FlowMessage.Parse

```
public static bool TryParse(string path, string json, out IFlowMessage flowMessage)
```

Parameters

`path` string

`json` string

`flowMessage` IFlowMessage

Returns

bool

TrySetMember(SetMemberBinder, object?)

Will set Crosser.EdgeNode.Common.Utilities.Json.CrosserDynamicObject.Properties binder.Name to the value passed in

```
public override bool TrySetMember(SetMemberBinder binder, object? value)
```

Parameters

`binder` SetMemberBinder

`value` object

Returns

bool

Class FlowMessageArrayJsonConverter

Namespace: Crosser.EdgeNode.Flows.Models.Abstractions.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowMessageArrayJsonConverter : JsonConverter<FlowMessage[]>
```

Inheritance

object ← JsonConverter ← JsonConverter<FlowMessage[]> ← FlowMessageArrayJsonConverter

Inherited Members

JsonConverter<FlowMessage[]>.CanConvert(Type) ,
JsonConverter<FlowMessage[]>.Read(ref Utf8JsonReader, Type, JsonSerializerOptions) ,
JsonConverter<FlowMessage[]>.ReadAsPropertyName(ref Utf8JsonReader, Type, JsonSerializerOptions)
,
JsonConverter<FlowMessage[]>.Write(Utf8JsonWriter, FlowMessage[], JsonSerializerOptions) ,
JsonConverter<FlowMessage[]>.WriteAsPropertyName(Utf8JsonWriter, FlowMessage[],
JsonSerializerOptions) ,
JsonConverter<FlowMessage[]>.HandleNull , JsonConverter<FlowMessage[]>.Type ,
JsonConverter.CanConvert(Type) , JsonConverter.Type , object.Equals(object) ,
object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Read(ref Utf8JsonReader, Type, JsonSerializerOptions)

Reads and converts the JSON to type `T`.

```
public override FlowMessage[] Read(ref Utf8JsonReader reader, Type typeToConvert,
JsonSerializerOptions options)
```

Parameters

`reader` Utf8JsonReader

The reader.

`typeToConvert` Type

The type to convert.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Returns

FlowMessage[]

The converted value.

Write(Utf8JsonWriter, FlowMessage[], JsonSerializerOptions)

Writes a specified value as JSON.

```
public override void Write(Utf8JsonWriter writer, FlowMessage[] values,
JsonSerializerOptions options)
```

Parameters

`writer` Utf8JsonWriter

The writer to write to.

`values` FlowMessage[]

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Class FlowMessageExtensions

Namespace: Crosser.EdgeNode.Flows.Models.Abstractions.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

Helpers for FlowMessage

```
public static class FlowMessageExtensions
```

Inheritance

object ← FlowMessageExtensions

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

DeepCopy(IFlowMessage)

```
[Obsolete("Use Clone instead")]
public static IFlowMessage DeepCopy(this IFlowMessage original)
```

Parameters

`original` IFlowMessage

Returns

IFlowMessage

Deserialize(string)

```
[Obsolete("Use FlowMessage.Deserialize(string json) instead")]
public static object? Deserialize(this string json)
```

Parameters

`json` string

Returns

object

GetPropertyLevel(IFlowMessage, string, string)

```
[Obsolete("Renamed to GetPropertyPath")]
public static string GetPropertyLevel(this IFlowMessage message, string property,
string newProperty)
```

Parameters

`message` IFlowMessage

`property` string

`newProperty` string

Returns

string

GetPropertyPath(IFlowMessage, string, string)

```
public static string GetPropertyPath(this IFlowMessage message, string property,
string newProperty)
```

Parameters

`message` IFlowMessage

`property` string

`newProperty` string

Returns

string

RemoveChangedProperties(IFlowMessage, IFlowMessage)

```
[Obsolete("Will be removed in the next major version")]
public static void RemoveChangedProperties(this IFlowMessage originalObject,
IFlowMessage changedObject)
```

Parameters

`originalObject` IFlowMessage

`changedObject` IFlowMessage

Serialize(IFlowMessage)

```
[Obsolete("Use FlowMessage.ToString() instead")]
public static string Serialize(this IFlowMessage message)
```

Parameters

`message` IFlowMessage

Returns

string

Serialize(FlowMessage)

```
[Obsolete("Use FlowMessage.ToString() instead")]
public static string Serialize(this FlowMessage message)
```

Parameters

`message` FlowMessage

Returns

string

Serialize(object)

```
[Obsolete("Use Crosser.EdgeNode.Common.Utilities.Json.JsonSerializer instead")]
public static string Serialize(object message)
```

Parameters

`message` object

Returns

string

ToFlowMessage(object?)

Converts an object into a FlowMessage

```
public static FlowMessage ToFlowMessage(object? obj)
```

Parameters

`obj` object

the object to convert to a FlowMessage

Returns

FlowMessage

A new FlowMessage

Class FlowMessageJsonConverter

Namespace: Crosser.EdgeNode.Flows.Models.Abstractions.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowMessageJsonConverter : JsonConverter<FlowMessage>
```

Inheritance

object ← JsonConverter ← JsonConverter<FlowMessage> ← FlowMessageJsonConverter

Inherited Members

JsonConverter<FlowMessage>.CanConvert(Type) ,
JsonConverter<FlowMessage>.ReadAsPropertyName(ref Utf8JsonReader, Type, JsonSerializerOptions) ,
JsonConverter<FlowMessage>.WriteAsPropertyName(Utf8JsonWriter, FlowMessage, JsonSerializerOptions) ,
JsonConverter<FlowMessage>.HandleNull , JsonConverter<FlowMessage>.Type ,
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Read(ref Utf8JsonReader, Type, JsonSerializerOptions)

Reads and converts the JSON to type FlowMessage.

```
public override FlowMessage Read(ref Utf8JsonReader reader, Type typeToConvert,
JsonSerializerOptions options)
```

Parameters

`reader` Utf8JsonReader

The reader.

`typeToConvert` Type

The type to convert.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Returns

FlowMessage

The converted value.

Write(Utf8JsonWriter, FlowMessage, JsonSerializerOptions)

Writes a specified value as JSON.

```
public override void Write(Utf8JsonWriter writer, FlowMessage value,
JsonSerializerOptions options)
```

Parameters

`writer` Utf8JsonWriter

The writer to write to.

`value` FlowMessage

The value to convert to JSON.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Class FlowMessageJsonSerializer

Namespace: Crosser.EdgeNode.Flows.Models.Abstractions.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

The default JSON serializer

```
public class FlowMessageJsonSerializer
```

Inheritance

object ← FlowMessageJsonSerializer

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

SerializerOptions

```
public static JsonSerializerOptions SerializerOptions { get; set; }
```

Property Value

JsonSerializerOptions

Methods

Deserialize(byte[], Type)

```
public dynamic? Deserialize(byte[] o, Type t)
```

Parameters

o byte[]

`t` Type

Returns

dynamic

Deserialize(byte[], Type, JsonSerializerOptions?)

Deserialize from byte[] to Type

```
public dynamic? Deserialize(byte[] o, Type t, JsonSerializerOptions? options)
```

Parameters

`o` byte[]

byte array

`t` Type

Target type

`options` JsonSerializerOptions

Returns

dynamic

Deserialize(Span<byte>, Type)

```
public dynamic? Deserialize(Span<byte> json, Type t)
```

Parameters

`json` Span<byte>

`t` Type

Returns

dynamic

Deserialize(Span<byte>, Type, JsonSerializerOptions?)

```
public dynamic? Deserialize(Span<byte> json, Type t, JsonSerializerOptions? options)
```

Parameters

`json` Span<byte>

`t` Type

`options` JsonSerializerOptions

Returns

dynamic

Deserialize(string)

```
public dynamic? Deserialize(string json)
```

Parameters

`json` string

Returns

dynamic

Deserialize(string, JsonSerializerOptions?)

```
public dynamic? Deserialize(string json, JsonSerializerOptions? options)
```

Parameters

`json` string

`options` JsonSerializerOptions

Returns

dynamic

Deserialize(string, Type)

```
public dynamic? Deserialize(string json, Type t)
```

Parameters

`json` string

`t` Type

Returns

dynamic

Deserialize(string, Type, JsonSerializerOptions?)

Deserialize from string to Type

```
public dynamic? Deserialize(string json, Type t, JsonSerializerOptions? options)
```

Parameters

`json` string

JSON string

`t` Type

Target type

`options` JsonSerializerOptions

Returns

dynamic

Deserialize<TTarget>(byte[])

```
public TTarget? Deserialize<TTarget>(byte[] o)
```

Parameters

`o` byte[]

Returns

TTarget

Type Parameters

`TTarget`

Deserialize<TTarget>(byte[], JsonSerializerOptions?)

Deserialize from byte[] to TTarget

```
public TTarget? Deserialize<TTarget>(byte[] o, JsonSerializerOptions? options)
```

Parameters

`o` byte[]

byte array

`options` JsonSerializerOptions

Returns

TTarget

Type Parameters

`TTarget`

Target type

Deserialize<TTarget>(Span<byte>)

```
public TTarget? Deserialize<TTarget>(Span<byte> json)
```

Parameters

`json` Span<byte>

Returns

TTarget

Type Parameters

`TTarget`

Deserialize<TTarget>(Span<byte>, JsonSerializerOptions?)

```
public TTarget? Deserialize<TTarget>(Span<byte> json, JsonSerializerOptions?
options)
```

Parameters

`json` Span<byte>

`options` JsonSerializerOptions

Returns

TTarget

Type Parameters

`TTarget`

Deserialize<TTarget>(string)

```
public TTarget? Deserialize<TTarget>(string json)
```

Parameters

`json` string

Returns

TTarget

Type Parameters

`TTarget`

Deserialize<TTarget>(string, JsonSerializerOptions?)

Deserialize from string to TTarget

```
public TTarget? Deserialize<TTarget>(string json, JsonSerializerOptions? options)
```

Parameters

`json` string

JSON string

`options` JsonSerializerOptions

Returns

TTarget

Type Parameters

`TTarget`

Target type

Serialize(object)

```
public string Serialize(object o)
```

Parameters

`o` object

Returns

string

Serialize(object, JsonSerializerOptions?)

Serialize object to JSON string

```
public string Serialize(object o, JsonSerializerOptions? options)
```

Parameters

`o` object

object

`options` JsonSerializerOptions

Returns

string

JSON string

Serialize(object, Type)

```
public string Serialize(object o, Type t)
```

Parameters

`o` object

`t` Type

Returns

string

Serialize(object, Type, JsonSerializerOptions?)

Serialize object of a specific Type to a JSON string

```
public string Serialize(object o, Type t, JsonSerializerOptions? options)
```

Parameters

`o` object

`t` Type

`options` JsonSerializerOptions

Returns

string

JSON string

Serialize<TSource>(TSource)

```
public string Serialize<TSource>(TSource o)
```

Parameters

`o` TSource

Returns

string

Type Parameters

`TSource`

Serialize<TSource>(TSource, JsonSerializerOptions?)

Serialize object TSource JSON string

```
public string Serialize<TSource>(TSource o, JsonSerializerOptions? options)
```

Parameters

`o` TSource

`options` JsonSerializerOptions

Returns

string

JSON string

Type Parameters

`TSource`

Class IFlowMessageJsonConverter

Namespace: Crosser.EdgeNode.Flows.Models.Abstractions.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class IFlowMessageJsonConverter : JsonConverter<IFlowMessage>
```

Inheritance

object ← JsonConverter ← JsonConverter<IFlowMessage> ← IFlowMessageJsonConverter

Inherited Members

JsonConverter<IFlowMessage>.CanConvert(Type) ,
JsonConverter<IFlowMessage>.ReadAsPropertyName(ref Utf8JsonReader, Type, JsonSerializerOptions)
,
JsonConverter<IFlowMessage>.WriteAsPropertyName(Utf8JsonWriter, IFlowMessage,
JsonSerializerOptions) ,
JsonConverter<IFlowMessage>.HandleNull , JsonConverter<IFlowMessage>.Type ,
object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() ,
object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Read(ref Utf8JsonReader, Type, JsonSerializerOptions)

Reads and converts the JSON to type Crosser.EdgeNode.Common.Abstractions.Models.IFlowMessage.

```
public override IFlowMessage Read(ref Utf8JsonReader reader, Type typeToConvert,
JsonSerializerOptions options)
```

Parameters

`reader` Utf8JsonReader

The reader.

`typeToConvert` Type

The type to convert.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Returns

IFlowMessage

The converted value.

Write(Utf8JsonWriter, IFlowMessage, JsonSerializerOptions)

Writes a specified value as JSON.

```
public override void Write(Utf8JsonWriter writer, IFlowMessage value,
JsonSerializerOptions options)
```

Parameters

`writer` Utf8JsonWriter

The writer to write to.

`value` IFlowMessage

The value to convert to JSON.

`options` JsonSerializerOptions

An object that specifies serialization options to use.

Class JsonSerializer

Namespace: Crosser.EdgeNode.Flows.Models.Abstractions.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

Wrapper for easier serialization

```
public static class JsonSerializer
```

Inheritance

object ← JsonSerializer

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Methods

Deserialize(byte[], Type)

Deserialize from byte[] to Type

```
public static dynamic? Deserialize(byte[] o, Type t)
```

Parameters

o byte[]

byte array

t Type

Target type

Returns

dynamic

Deserialize(string)

```
public static dynamic? Deserialize(string json)
```

Parameters

`json` string

Returns

dynamic

Deserialize(string, Type)

Deserialize from string to Type

```
public static dynamic? Deserialize(string json, Type type)
```

Parameters

`json` string

JSON string

`type` Type

Target type

Returns

dynamic

Deserialize<TTarget>(byte[])

Deserialize from byte[] to TTarget

```
public static TTarget? Deserialize<TTarget>(byte[] o)
```

Parameters

`o` byte[]

byte array

Returns

TTarget

Type Parameters

`TTarget`

Target type

Deserialize<TTarget>(string)

Deserialize from string to TTarget

```
public static TTarget? Deserialize<TTarget>(string json)
```

Parameters

`json` string

JSON string

Returns

TTarget

Type Parameters

`TTarget`

Target type

Serialize(object)

Serialize object to JSON string

```
public static string Serialize(object o)
```

Parameters

o object

object

Returns

string

JSON string

Serialize(object, Type)

Serialize object of a specific Type to a JSON string

```
public static string Serialize(object o, Type t)
```

Parameters

o object

t Type

Returns

string

JSON string

Serialize<TSource>(TSource)

Serialize object TSource JSON string

```
public static string Serialize<TSource>(TSource o)
```

Parameters

o TSource

Returns

string

JSON string

Type Parameters

TSource

Namespace Crosser.EdgeNode.Shared.Models

Classes

FlowBase

FlowInformation

FlowModule

FlowModuleCommonSettings

FlowModuleSettings

Metadata

Enums

RequestOrigin

Class FlowBase

Namespace: Crosser.EdgeNode.Shared.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public abstract class FlowBase
```

Inheritance

object ← FlowBase

Derived

Flow, FlowInformation

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Description

```
public string Description { get; set; }
```

Property Value

string

FlowDefinitionId

```
public Guid FlowDefinitionId { get; set; }
```

Property Value

Guid

FlowDeploymentId

```
public Guid? FlowDeploymentId { get; set; }
```

Property Value

Guid?

FlowVersion

```
public int FlowVersion { get; set; }
```

Property Value

int

Id

```
public Guid Id { get; set; }
```

Property Value

Guid

Name

```
public string Name { get; set; }
```

Property Value

string

ResolveFlowDeploymentId

```
[JsonIgnore]
public Guid ResolveFlowDeploymentId { get; }
```

Property Value

Guid

Class FlowInformation

Namespace: Crosser.EdgeNode.Shared.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowInformation : FlowBase
```

Inheritance

object ← FlowBase ← FlowInformation

Inherited Members

FlowBase.Name , FlowBase.FlowVersion , FlowBase.Description , FlowBase.Id , FlowBase.FlowDefinitionId , FlowBase.FlowDeploymentId , FlowBase.ResolveFlowDeploymentId , object.Equals(object) , object.Equals(object, object) , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowInformation()

```
public FlowInformation()
```

Properties

Credentials

```
public List<CredentialBase> Credentials { get; set; }
```

Property Value

List<CredentialBase>

FlowStatus

```
public Status FlowStatus { get; set; }
```

Property Value

Status

HaltOnError

```
public bool HaltOnError { get; set; }
```

Property Value

bool

IsInteractive

```
public bool IsInteractive { get; set; }
```

Property Value

bool

Modules

```
public List<FlowModule> Modules { get; set; }
```

Property Value

List<FlowModule>

OnFlowStatusChanged

```
[JsonIgnore]
public EventHandler<Status>? OnFlowStatusChanged { get; set; }
```

Property Value

EventHandler<Status>

Resources

```
public List<ResourceBase> Resources { get; set; }
```

Property Value

List<ResourceBase>

ShouldBeStarted

```
[JsonIgnore]
public bool ShouldBeStarted { get; set; }
```

Property Value

bool

Methods

GetHashCode()

Serves as the default hash function.

```
public override int GetHashCode()
```

Returns

int

A hash code for the current object.

SetModuleStatus(Guid, Status, string)

```
public Status SetModuleStatus(Guid moduleId, Status status, string message)
```

Parameters

`moduleId` Guid

`status` Status

`message` string

Returns

Status

Class FlowModule

Namespace: Crosser.EdgeNode.Shared.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowModule
```

Inheritance

object ← FlowModule

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Constructors

FlowModule()

```
public FlowModule()
```

Properties

FlowId

```
public Guid FlowId { get; set; }
```

Property Value

Guid

HaltOnError

```
public bool HaltOnError { get; set; }
```

Property Value

bool

Id

```
public Guid Id { get; set; }
```

Property Value

Guid

LastReportedStatus

```
public string LastReportedStatus { get; set; }
```

Property Value

string

LastReportedTime

```
public DateTime? LastReportedTime { get; set; }
```

Property Value

DateTime?

Metadata

```
public Metadata Metadata { get; set; }
```

Property Value

Metadata

ModuleVersion

```
public string ModuleVersion { get; set; }
```

Property Value

string

Name

```
public string Name { get; set; }
```

Property Value

string

Resources

```
public IList<ResourceBase> Resources { get; set; }
```

Property Value

IList<ResourceBase>

Settings

```
public FlowModuleSettings Settings { get; set; }
```

Property Value

FlowModuleSettings

Status

```
public Status Status { get; set; }
```

Property Value

Status

Type

```
public string Type { get; set; }
```

Property Value

string

Class FlowModuleCommonSettings

Namespace: Crosser.EdgeNode.Shared.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowModuleCommonSettings
```

Inheritance

object ← FlowModuleCommonSettings

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

ChannelFullMode

```
public BoundedChannelFullMode ChannelFullMode { get; set; }
```

Property Value

BoundedChannelFullMode

MaxItemsInQueue

```
public int MaxItemsInQueue { get; set; }
```

Property Value

int

PersistentMessages

```
public bool PersistentMessages { get; set; }
```

Property Value

bool

PersistentMessagesRetries

```
public int PersistentMessagesRetries { get; set; }
```

Property Value

int

PersistentMessagesRetryDelay

```
public int PersistentMessagesRetryDelay { get; set; }
```

Property Value

int

Class FlowModuleSettings

Namespace: Crosser.EdgeNode.Shared.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class FlowModuleSettings
```

Inheritance

object ← FlowModuleSettings

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

CommonSettings

```
public FlowModuleCommonSettings CommonSettings { get; set; }
```

Property Value

FlowModuleCommonSettings

Class Metadata

Namespace: Crosser.EdgeNode.Shared.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public class Metadata
```

Inheritance

object ← Metadata

Inherited Members

object.Equals(object) , object.Equals(object, object) , object.GetHashCode() , object.GetType() , object.MemberwiseClone() , object.ReferenceEquals(object, object) , object.ToString()

Properties

Name

```
public string? Name { get; set; }
```

Property Value

string

Enum RequestOrigin

Namespace: Crosser.EdgeNode.Shared.Models

Assembly: Crosser.EdgeNode.Flows.Abstractions.dll

```
public enum RequestOrigin
```

Fields

`Cloud = 1`

`Local = 2`

`Unknown = 0`