EPIPHRON SECURITY ID VALIDATOR DEVELOPMENT

Introduction

This document describes the steps required to implement an Epiphron Security Identifier Validator and make it available to the Epiphron DRTx® platform.

A security identifier validator is a software component (DLL) that enhances the Epiphron platform's ability to recognize different types of security identifiers (SEDOL, CUSIP, etc.). While some validators are already present in Epiphron DRTx, new or customized validators can be developed and integrated into Epiphron DRTx so that security identifiers can be validated to ensure they are correctly entered by users.

Implementation of a Security ID Validator

At its core, an Epiphron Security Identifier Validator is a class that implements an interface recognized by Epiphron. This interface defines the following properties and methods:

```
public interface IDRTxSecIdValidator
    {
        string UniqueId { get; }
        string Name { get; }
        bool IsImplemented { get; }
        string Author { get; }
        string Version { get; }

        bool AllowedMixedCase { get; }
        int MinLength { get; }
        int MaxLength { get; }
        string AllowedCharacters { get; }

        int DefaultSmartSearchEvaluationOrder { get; }

        bool IsValidIdentifier(string identifier, bool checkDigitRequired, out string
identifierWithCRC, out string nonValidReason);
    }
```

Additional details can be found in the "Epiphron Security Identifier Validator Interfaces" section of this document.

We will use Visual Studio 2022 and C# to create and implement a custom security identifier validator within Epiphron. A basic general knowledge of .NET and C# is required to follow this guide; however, other .NET languages can be used instead of C#. The only requirement

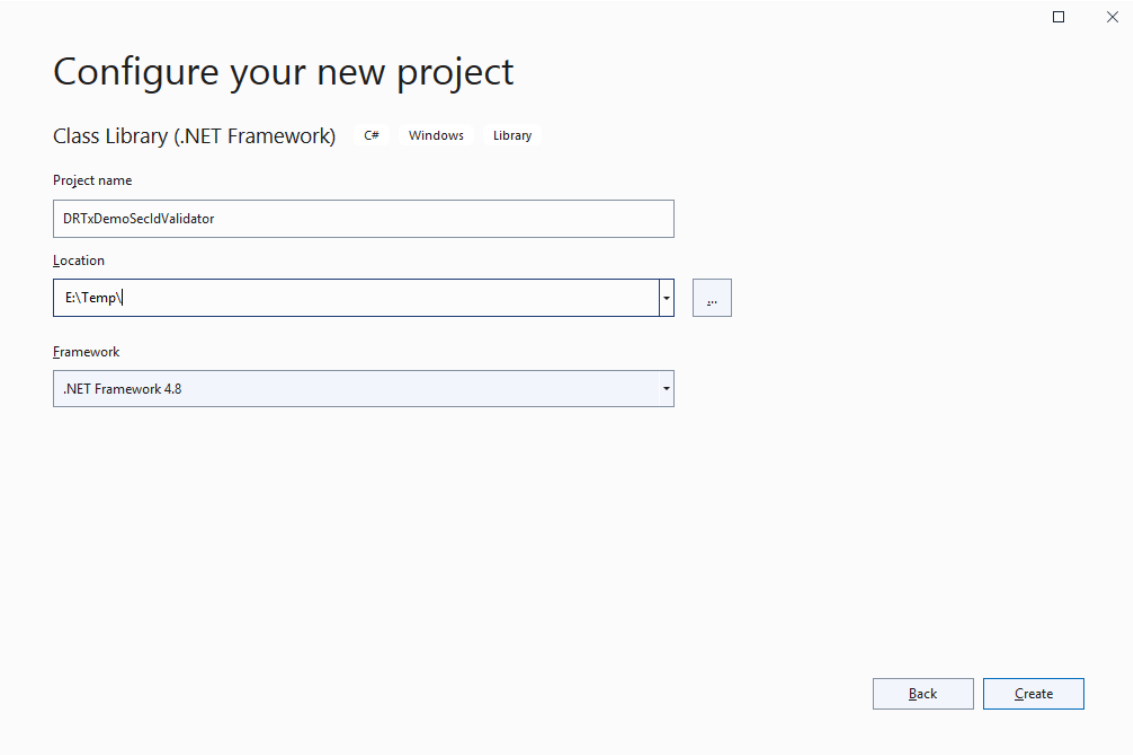
related to the .NET Framework version is that the Security ID validator must be coded in .NET 4.8 or older, as the Epiphron service can currently only consume validator DLLs written in these versions.

Step-by-Step Implementation Process

Step 1: Create the Project

Create a DLL library project to host the class that will implement the validator:

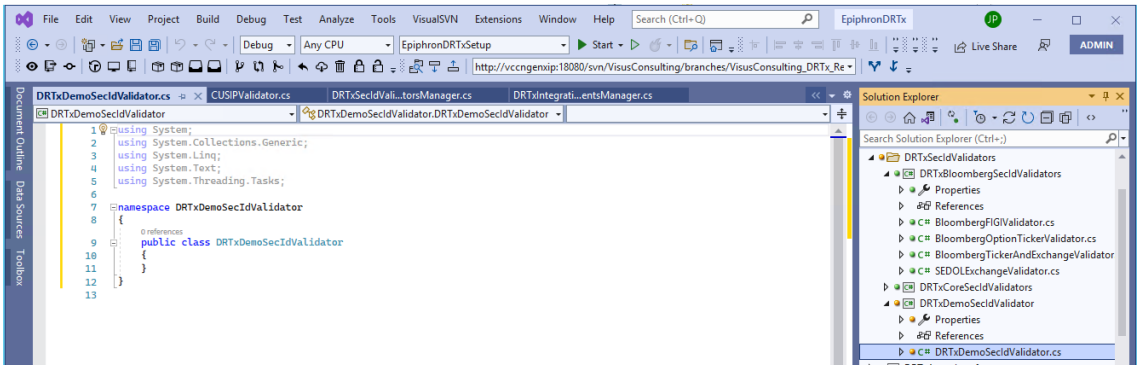
1. Open Visual Studio 2022
2. Select "Create a new project"
3. Choose "Class Library (.NET Framework)"
4. Name the project **DRTxDemoSecIdValidator**
5. Ensure the target framework is .NET Framework 4.8 or earlier
6. Click "Create"



Step 2: Add the Public Validator Class

Once the project is created, add a public class that will implement the validator:

1. Right-click on the project in Solution Explorer
2. Select "Add" → "Class..."
3. Name the class file "*DRTxDemoSecIdValidator.cs*"
4. Ensure the class is marked as public

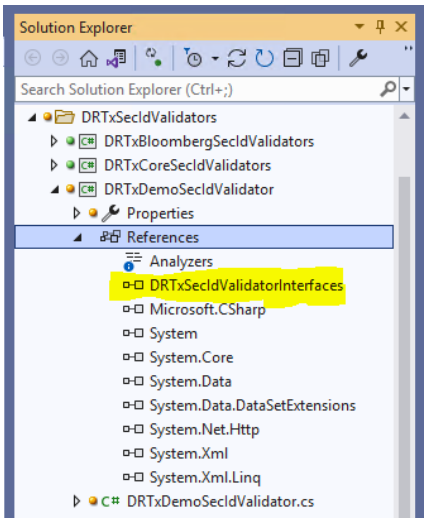


Critical: The class must be marked as 'public'; otherwise, the Epiphron service will not be able to load it.

Step 3: Add Required References

Reference the required Epiphron interface library:

1. Right-click on "References" in Solution Explorer
2. Select "Add Reference..."
3. Click "Browse" and navigate to the Epiphron service binaries folder
4. Select *DRTxSecIdValidatorInterfaces.dll*
5. Click "Add" and then "OK"



Step 4: Implement the Interface

Implement the IDRTxSecIdValidator interface in your class:

1. Add the using statement: using *VCC.DRTx.SecIdValidatorInterfaces*;
2. Make your class inherit from *IDRTxSecIdValidator*
3. Use Visual Studio's Quick Actions (Ctrl+.) to implement the interface
4. It is recommended to Implement interface explicitly for better encapsulation

```
    public class DRTxDemoSecIdValidator : IDRTxSecIdValidator
    {
        string IDRTxSecIdValidator.UniqueId => throw new NotImplementedException();
        string IDRTxSecIdValidator.Name => throw new NotImplementedException();
        bool IDRTxSecIdValidator.IsImplemented => throw new NotImplementedException();
        string IDRTxSecIdValidator.Author => throw new NotImplementedException();
        string IDRTxSecIdValidator.Version => throw new NotImplementedException();

        bool IDRTxSecIdValidator.AllowedMixedCase => throw new NotImplementedException();
        int IDRTxSecIdValidator.MinLength => throw new NotImplementedException();
        int IDRTxSecIdValidator.MaxLength => throw new NotImplementedException();
        string IDRTxSecIdValidator.AllowedCharacters => throw new NotImplementedException();

        int IDRTxSecIdValidator.DefaultSmartSearchEvaluationOrder => throw new
NotImplementedException();

        bool IDRTxSecIdValidator.IsValidIdentifier(string identifier, bool checkDigitRequired,
out string identifierWithCRC, out string nonValidReason)
        {
            throw new NotImplementedException();
        }
    }
```

Step 5: Implement the Interface Methods

Replace the *NotImplementedException* placeholders with actual implementation:

1. Implement each property to return appropriate values for your validator
2. Implement the *IsValidIdentifier* method with your validation logic
3. Test your implementation thoroughly
4. Build the project to create the DLL

A detailed description of the purpose of each method is provided in the Interfaces section at the bottom of this document. For convenience, a working code sample for Security ID Validator creation can be found in the Epiphron DRTx™ code sample directory that is created upon installation of Epiphron DRTx™.

Deployment of a Security ID Validator

The Epiphron service dynamically loads all available validators at service startup. Follow these steps to deploy your validator:

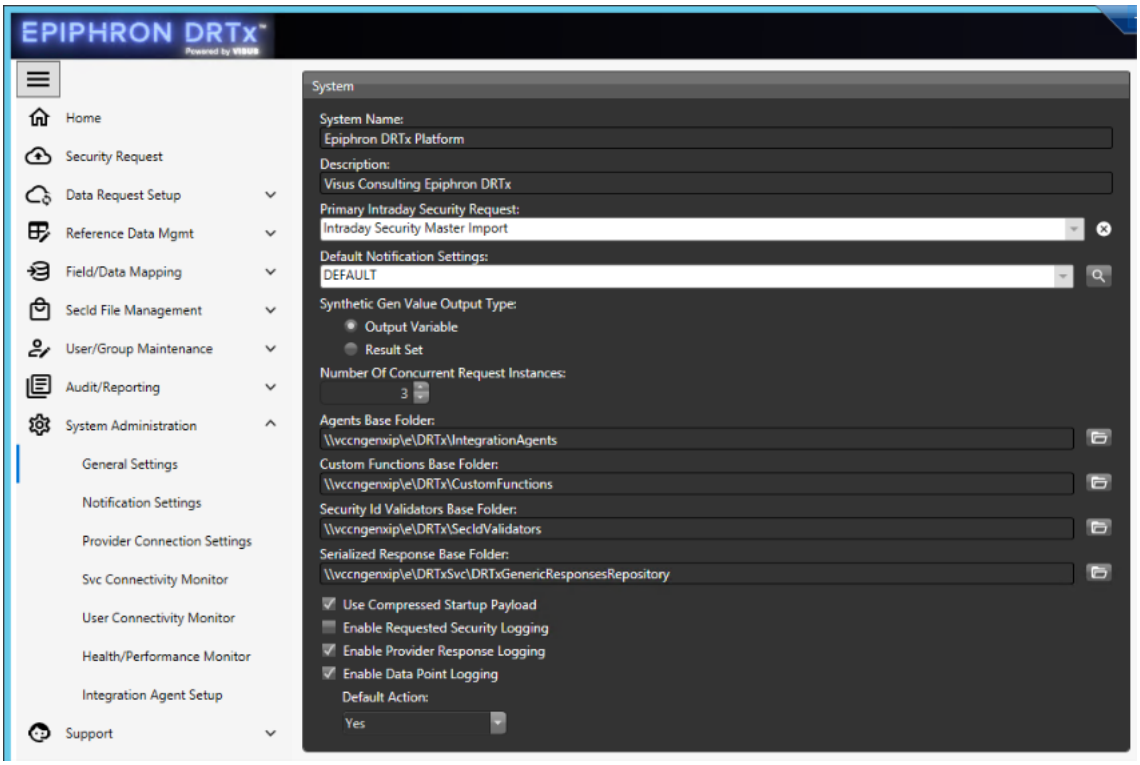
Step-by-Step Deployment Process

Step 1: Build the Validator DLL

1. Ensure your validator implementation is complete and tested
2. Build the project in Visual Studio (Build → Build Solution)
3. Verify that the DLL file is created in the output directory (typically bin/Debug or bin/Release)

Step 2: Locate the Validators Folder

1. Open the Epiphron system
2. Navigate to *System Administration → General Settings → System section*
3. Find the *"Security Id Validators Base Folder"* setting
4. Note the folder path specified in this setting



Step 3: Deploy the Validator Files

1. Copy your compiled validator DLL to the validators folder identified in Step 2
2. Copy any required dependency DLLs to the same folder
3. Ensure all files have appropriate read permissions for the Epiphron service account

Step 4: Restart the Epiphron Service

1. Stop the Epiphron service
2. Start the Epiphron service
3. Verify in the service logs that your validator was loaded successfully

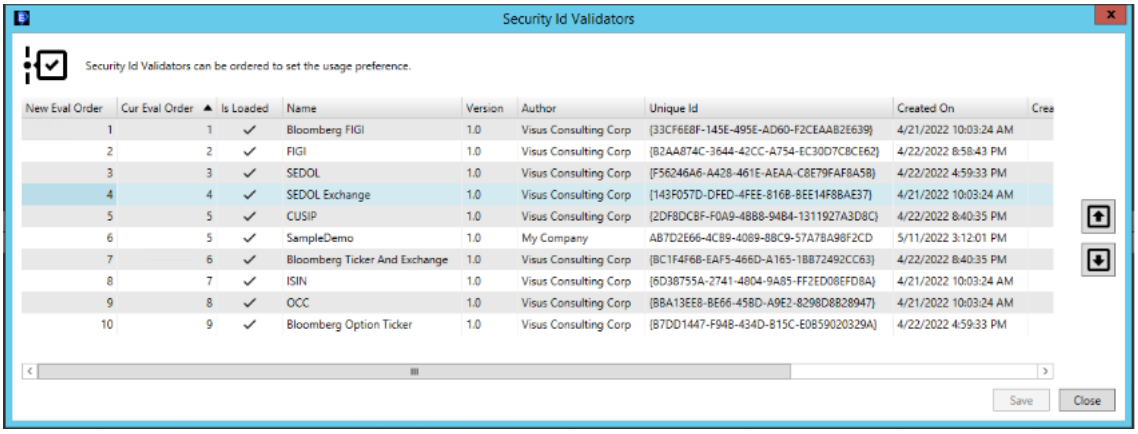
Configuration of an Epiphron Security ID Validator

After deployment, you can configure your validator's behavior within the Epiphron system. Currently, only the "Smart Search Order" can be user-defined for each validator.

Step-by-Step Configuration Process

Step 1: Access Validator Management

1. Open the Epiphron system
2. Navigate to *Reference Data Management → Security Id Types*
3. Right-click to access the context menu
4. Select *"Security Id Validator Mgmt"*



Step 2: Configure Smart Search Order

1. In the Security Id Validator Management interface, locate your validator
2. Adjust the evaluation order number as needed
3. Consider the sequence in which validators should be evaluated
4. Save the configuration changes

Note: Lower numbers are evaluated first. The order is important because once a validator successfully identifies and validates an identifier, the evaluation stops.

Usage of Epiphron Validators

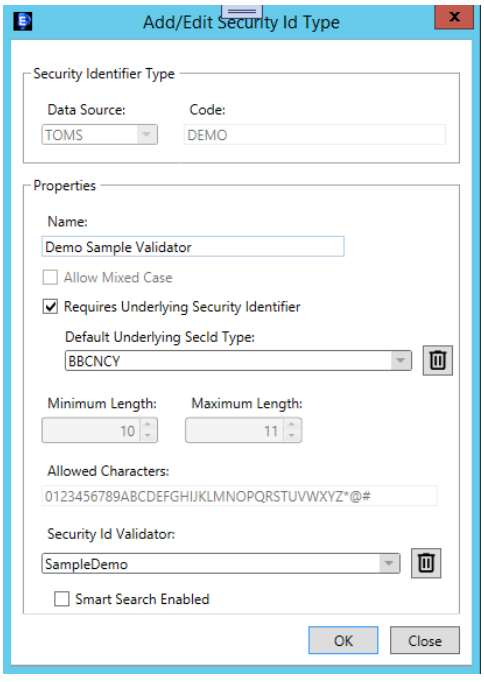
To successfully use a validator in Epiphron, it must be assigned to one or more Security ID Types. The following steps will guide you through this process.

Step-by-Step Usage Process

Step 1: Create a New Security ID Type

For this example, we will create a new Security ID Type called **Demo**:

1. Navigate to *Reference Data → Security Id Type*
2. Select *Create...*
3. Enter "Demo" as the Security ID Type name
4. Select your custom validator from the available validators list
5. Configure other required properties as needed



Step 2: Configure Smart Search (Optional)

If you want this Security ID Type to be used with the "Smart Search" feature:

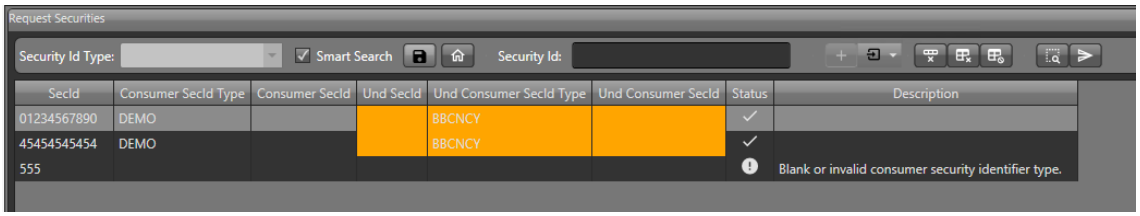
1. In the Security ID Type creation dialog
2. Locate the *"Smart Search Enabled"* checkbox at the bottom
3. Check the box to enable Smart Search for this type
4. Save the Security ID Type configuration

Step 3: Test the Validator

Once the validator is configured and in use:

1. **Explicit Testing:** Request a security by explicitly selecting the Security ID Type that uses your new validator
2. **Smart Search Testing:** Use the Smart Search feature to let Epiphron automatically identify the Security ID Type
3. Verify that validation works correctly with valid identifiers
4. Test with invalid identifiers to ensure proper error messages are displayed

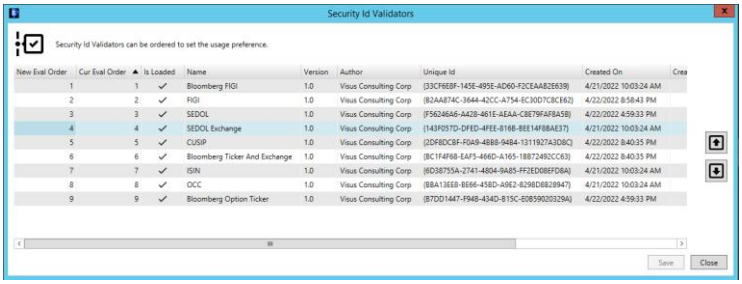
Important: Execution order of validators is critical. Once a validator finds and validates an identifier, it executes and does not move on to additional validators. Ensure your validator's evaluation order is set appropriately.



Epiphron Security ID Validator Interfaces

The tables below describe the methods and properties of the interfaces that must be implemented to successfully create a custom Security ID validator.

Properties

Type	Name	Description
string	UniqueId	A unique and inmutable value that will be used within the Epiphron system to identify the validator.
string	Name	A human-readable name for the agent. It will be used to display information about the Validator.
bool	IsImplemented	A Boolean flag that will be used from the Epiphron system to skip a validator even though it can be loaded. If true is returned, then it will be used. Otherwise, Epiphron will consider the validator not fully implemented and will skip it.
string	Author	The company name that developed the validator.
string	Version	The version of the validator. It is used to display information about the validator.
bool	AllowedMixedCase	Security Identifiers are expected to be uppercase. If this flag is set, it will allow for this particular type of Security Identifiers to be lowercase or mixed upper/lower case.
int	MinLength	Minimum length expected for the identifier.
int	MaxLength	Maximum length of the identifier.
string	AllowedCharacters	A string that contains the characters that might appear in the Security Identifier (i.e.: “0123456789ABCDEF” so “12AB” is allowed but “ZZ89” is not).
int	DefaultSmartSearchEvaluationOrder	A number to define the order sequence to evaluate validators when using “Smart Security Search” feature. This order can be rearranged inside Epiphron, so this property will set the initial value used (on “Reference Data Management → Security Id Types → Security Id Validator Mgmt context menu): 

Methods

Name	Argument	Description
IsValidIdentifier	Function that evaluates a Security Identifier and indicates if it is valid or not.	
	string identifier	The identifier text.
	bool checkDigitRequired	If set, the identifier must include a check character (CRC) and it should be valid. If not set, CRC is not expected to be in the identifier.
	out string identifierWithCRC	Output parameter that contains the identifier with the CRC character.
	out string nonValidReason	In case the identifier is not valid, this output parameter should contain a description of the reason for it to be rejected. This message will be shown to the user.