EPIPHRON EXPRESSIONS DEVELOPMENT GUIDE

Introduction

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

An Expression library is software (DLL) that provides a method to perform custom calculations (numeric, text, date, etc.) within the Epiphron platform. Expressions can be used to display custom columns in data grids.

Implementation of an Epiphron Expression

At its core, an Epiphron Expression is simply a class that implements an interface (IDRTxCustomFunction) recognized by Epiphron. This interface defines the following properties and methods:

```
public interface IDRTxCustomFunction
    {
        string Name { get; }
        string Syntax { get; }
        string Description { get; }
        DRTxCustomFunctionTypeEnum FunctionType { get; }
        int MinArgsCount { get; }
        int MaxArgsCount { get; }
        bool IsMultiArguments { get; }
        IEnumerable<IDRTxCustomFunctionArgument> Arguments { get; }

        object Expression(List<object> values);
    }
```

Details about these properties and methods can be found in the "Epiphron Expression Interfaces" section of this document.

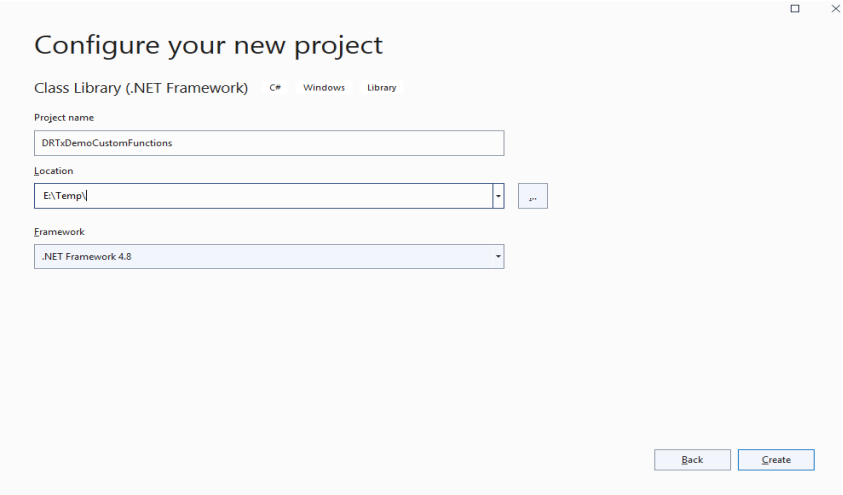
Development Example

For demonstration purposes, let's create a custom expression. This guide uses Visual Studio 2022 and C# for implementation. Basic knowledge of .NET and C# is required to follow this guide. Other .NET languages can be used instead of C#. The only

requirement is related to the .NET Framework version—currently, the Epiphron service can only consume DLLs written in .NET 4.8 or earlier.

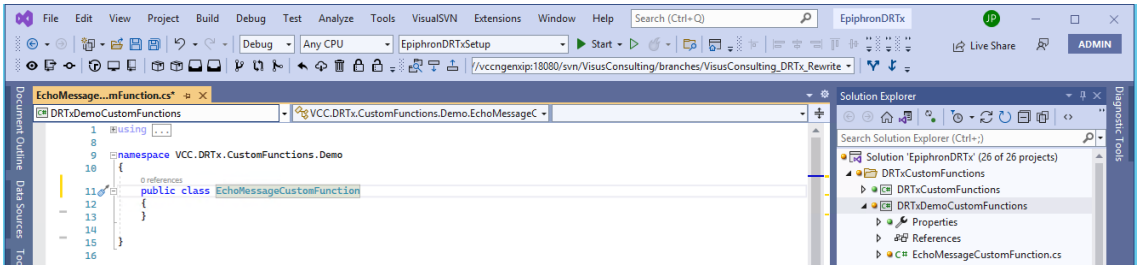
Step 1: Create the Project

First, create a DLL library project to host the class that will implement the expression. For this example, we'll call the project **DRTxDemoCustomFunctions**.



Step 2: Create the Expression Class

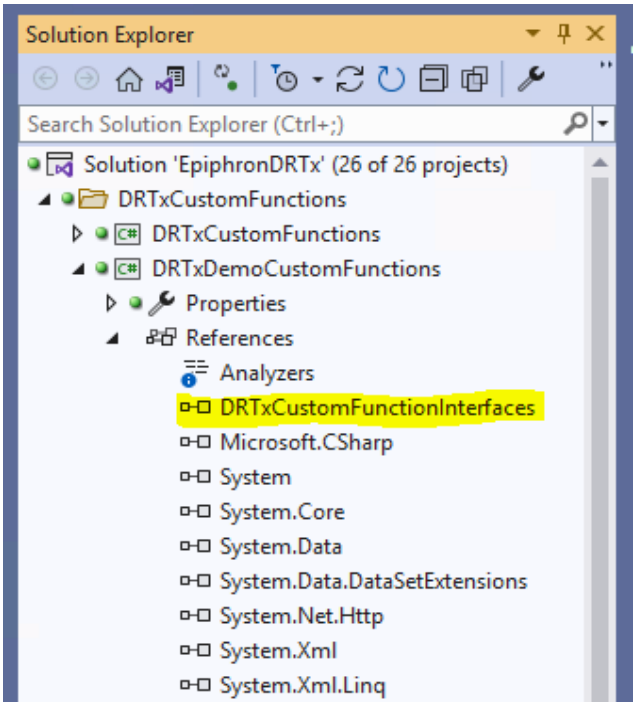
Once the project is created, add a public class *EchoMessageCustomFunction* that will implement the expression.



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

Step 3: Add Reference

Add a reference to the library *DRTxCustomFunctionInterfaces.dll*, which can be found among the service binaries.



Step 4: Implement the Interface

Implement the interface IDRTxCustomFunction declared in the namespace *VCC.DRTx.CustomFunctionInterfaces* located in the newly referenced library. It is recommended to implement the interface explicitly.

Step 5: Implement Expression Logic

Our *EchoMessageCustomFunction* expression will accept one text parameter and return the text "You just typed: " followed by the text parameter. The implementation code is as follows:

```
        string IDRTxCustomFunction.Name => "EchoMessage";
        string IDRTxCustomFunction.Syntax => "EchoMessage()";
        string IDRTxCustomFunction.Description => "Returns the text passed as parameter.";
        DRTxCustomFunctionTypeEnum IDRTxCustomFunction.FunctionType =>
DRTxCustomFunctionTypeEnum.TextFuncs;
        int IDRTxCustomFunction.MinArgsCount => 1;
        int IDRTxCustomFunction.MaxArgsCount => 1;

        bool IDRTxCustomFunction.IsMultiArguments => false;
        IEnumerable<IDRTxCustomFunctionArgument> IDRTxCustomFunction.Arguments => new
IDRTxCustomFunctionArgument[] { new MessageFunctionArgument() };

        object IDRTxCustomFunction.Expression(List<object> values)
        {
            string result = string.Empty;

            bool ok = true;
            string error = string.Empty;

            // 1st step: Verify the validity of the provided arguments (number and types).
            if (ok)
            {
                ok = verify_arguments(values, out error);
            }

            // 2nd step: If the parameters are valid, extract them from 'values', convert them
to the appropriate types and perform the custom function.
            // Catch a potential exception in the custom function and write down the erorr.
            if (ok)
            {
                try
                {
                    // Extract and convert arguments.
                    string message = values[0].ToString();

                    // Perform the custom function.
                    result = string.Format("You just typed: '{0}'", message);
                }
                catch (Exception ex)
                {
                    ok = false;
                    error = ex.Message;
                }
            }

            // 3rd step: If there has been an error while evaluating the input arguments or
when executing the custom function, then throw an exception. Otherwise,
            // return the result of the custom function.
            if (!ok)
            {
                throw new Exception(string.Format("{0}: {1}", Name, error));
            }
```

```
            return result;
        }
```

Step 6: Implement Parameter Class

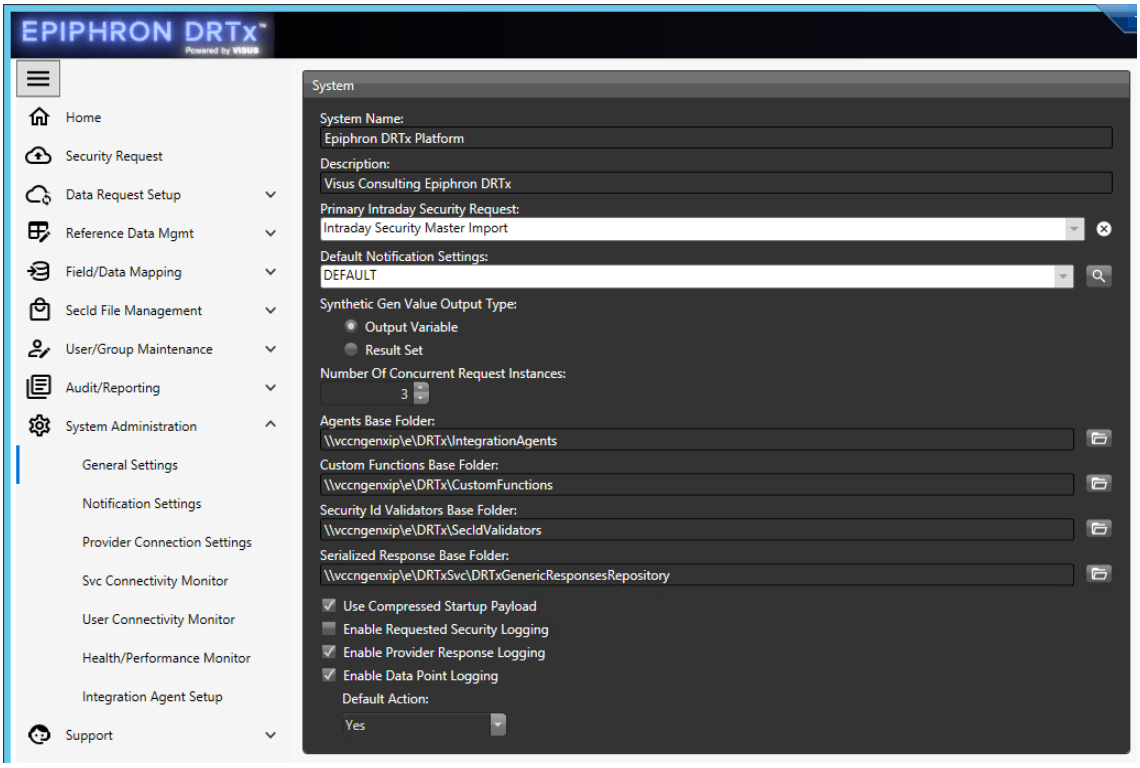
Expressions may have input parameters. To handle them, each parameter should be declared in a public class that implements the interface *IDRTxCustomFunctionArgument*.

```
    public class MessageFunctionArgument : IDRTxCustomFunctionArgument
    {
        public string Name => "Message";

        public string Description => "Message to be returned.";

        public bool IsShowDescription => true;

        public Type[] Types => new Type[] { typeof(string) };
    }
```

Deployment of an Epiphron Expression

The Epiphron service dynamically loads all available expressions at service startup. To accomplish this, binaries must be copied to the folder configured in Epiphron for expressions.

This folder is configured in: *System Administration → General Settings → System section → "Custom Functions Base Folder" setting*.



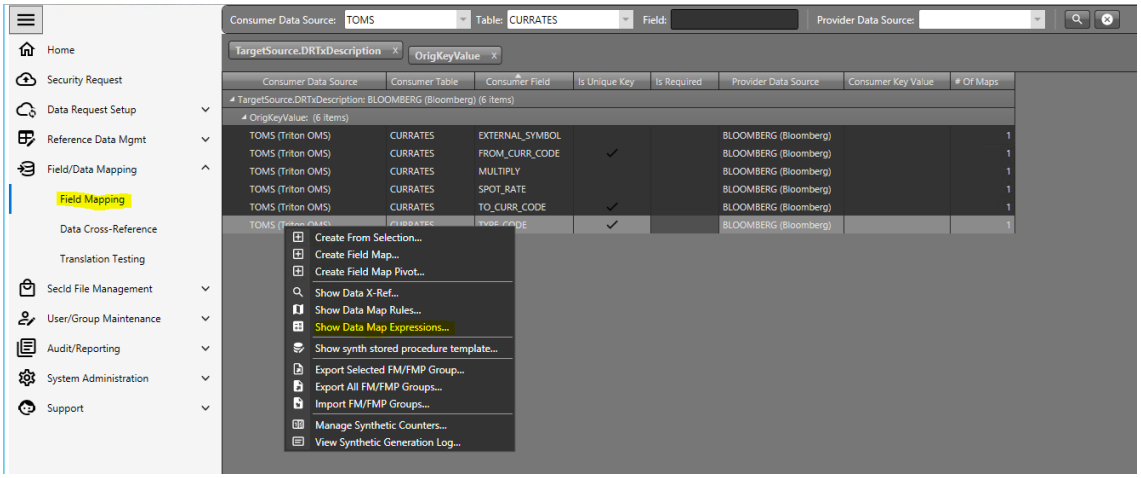
Configuration of an Epiphron Expression

There are no configurable settings for expressions—they are simply used within Epiphron.

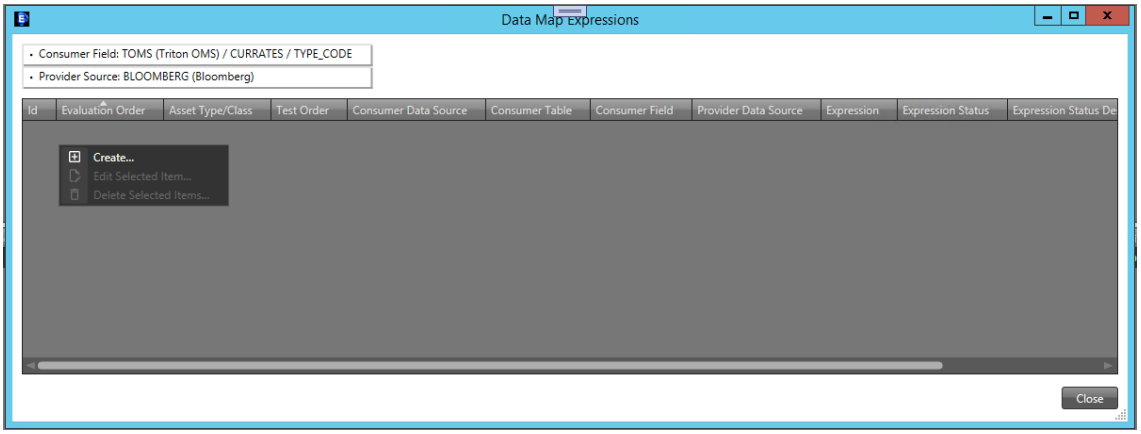
Usage of Epiphron Expressions

Expressions can be used in Data Mappings. To access this functionality:

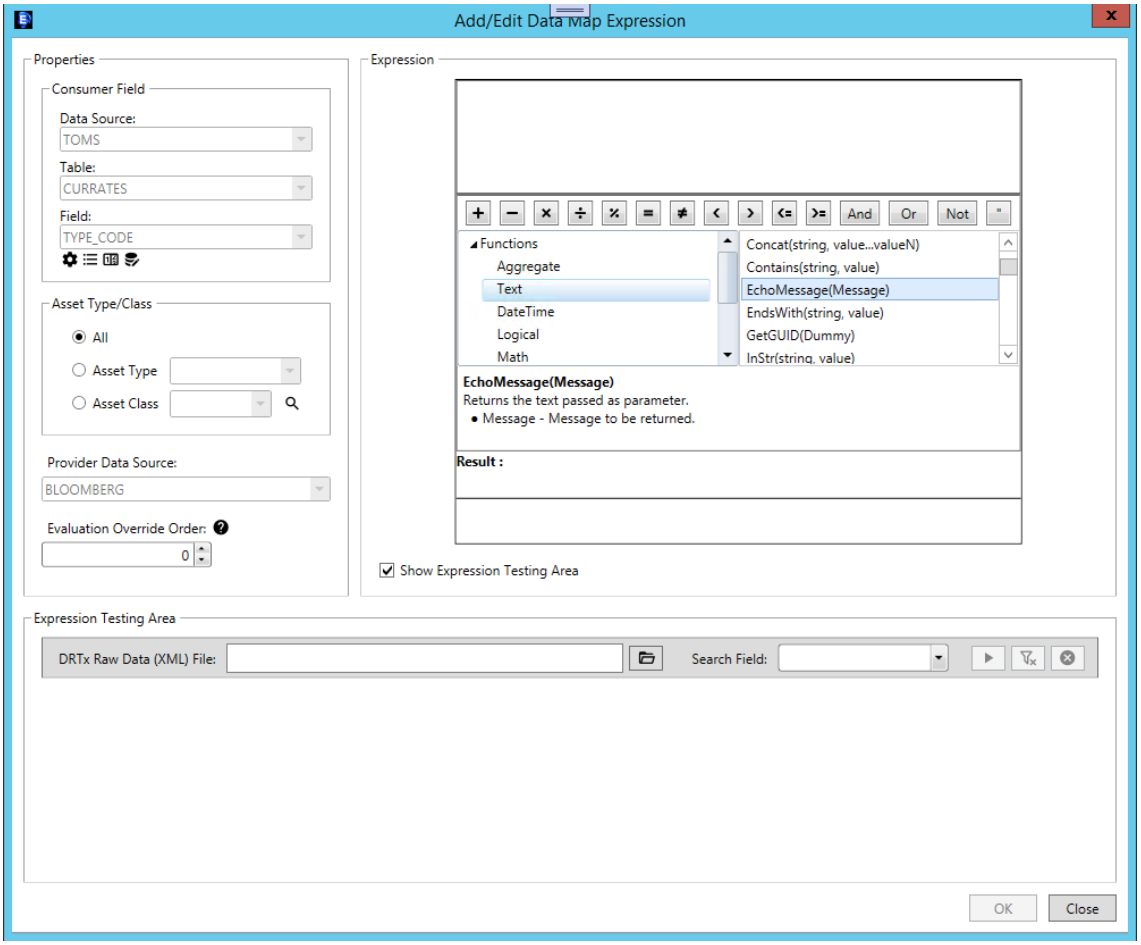
1. Navigate to *Field Data Mapping → Field Mapping*
2. Select a consumer data source and table
3. Click the search button (magnifying glass icon)



Once the Expressions dialog is displayed, users can edit existing expressions or create new ones.



In our example, we declared the function to be of type *DRTxCustomFunctionTypeEnum.TextFuncs*, so it appears in the Text function group. The description of both the expression and parameters are those included in the code.

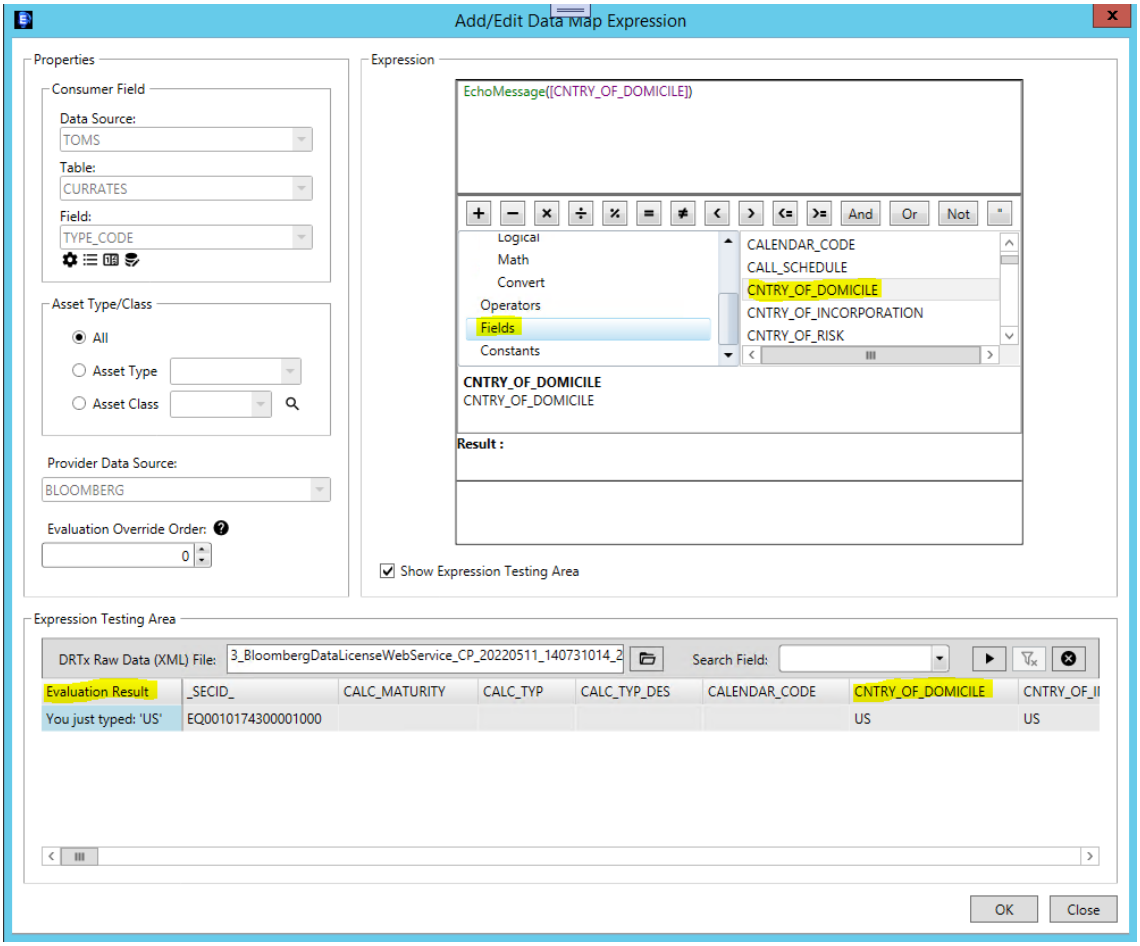


Testing the Expression

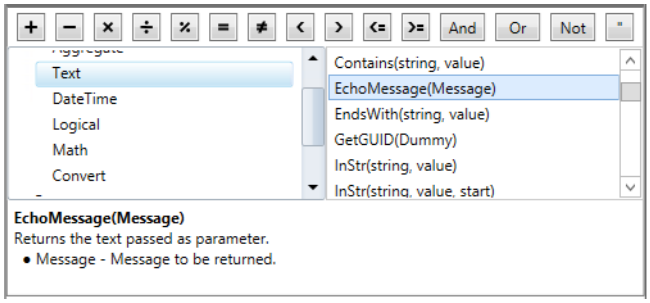
To test our expression:

1. Load a Raw Data (XML) file
2. In the "Expression" section, type the function name (or double-click on it) and include the parameter
3. To use a field from the Raw Data file, set the column name as the parameter
4. Fields are also available as an Expression Group—clicking on it will show all loaded fields
5. Once the expression is complete, evaluate it by clicking the "Evaluate" button

The result of our evaluation shows our implemented fixed text ("You just typed:") followed by the value of the selected field for a given row.



Epiphron Expression Interfaces

Interface IDRTxCustomFunction		
Properties		
Type	**Name**	**Description**
string	Name	Name of the expression. It will be used to display information to the user.
string	Syntax	Syntax of our custom function.
string	Description	Description that will be shown on the Expression editor for this function. 

DRTxCustomFunctionTypeEnum	FunctionType	The type of the function. This will determine the group where the function will be shown on the Expression editor. Possible values are: • Field • Constant: constant functions • Operator: field operators functions • AggregateFuncs: aggregation functions • TextFuncs: text functions • DateTimeFuncs: date/time functions • LogicalFuncs: logical functions • MathFuncs: mathematical functions • ConvertFuncs: conversión functions
int	MinArgsCount	Minimum number of arguments accepted by the function.
int	MaxArgsCount	Maximum number of arguments accepted by the function.
bool	IsMultiArguments	Flag to indicate that the function has an argument range.
IEnumerable<IDRTxCustomFunctionArgument>	Arguments	Enumeration of arguments accepted by the function. Those arguments must implement *IDRTxCustomFunctionArgument*.

Methods

Name	Name	Description
Expression	Performs a calculation based on the input parameters.	
	Object	Result of the calculation.
	List<object> values	Parameter values. The method should handle those values to ensure they do have the required arguments for the custom function (validate its length, type ...).
	out string error	In case tany problem, this output parameter should contain the problem description. This message will be shown to the user.

Interface IDRTxCustomFunctionArgument

Properties

Type	Name	Description
string	Name	Name of the parameter. It will be used to display information to the user.
string	Description	Argument description. It will be used to display information to the user.
bool	IsShowDescription	Flag to indicate if the description should be displayed or hidden.
Type[]	Types	The type of the parameter. It will usually be a single type per parameter but might indicate more for complex parameters.