

Accessing .Net Assemblies in Vijeo Citect 2015

11th March 2015

Graeme Davey
Program Manager

Table of contents

1. Introduction	4
1.1. Purpose	4
1.2. Terminology	4
1.3. Cicode functions used	5
2. Hello World (C#)	6
2.1. Creating a new class library	6
2.2. Constructor	7
2.3. Properties	8
2.4. Methods	8
3. Hello World (Cicode)	10
3.1. Creating an instance of the class	10
3.2. Getting a property	10
3.3. Setting a property	11
3.4. Calling a method	11
3.5. Limitations	11
4. Calling predefined functions in windows	12
4.1. Creating an object	13
4.2. Supported Data Types	15
5. Example: Creating an alarm paging system	16
5.1. Configuring the Paging and Paging Group fields	16
5.2. Alarm Event Queue	17
5.3. Sending an alarm notification via email	18
6. Logging	21
6.1. Unhandled Exceptions	21
6.2. Unsupported data types	22

7. Conclusion	23
8. Appendix A: “Hello World!” C# code	24
9. Appendix B: “Hello World!” cicode	25
10. Appendix C: ProcessEventQue()	27
11. References	28

1. Introduction

1.1. Purpose

The aim of this whitepaper is to give users an overview of how users can access dot net assemblies within Vijeo Citect 2015. This feature allows users to either access Microsoft's .Net™ framework directly, or to write a .Net assembly and access that via a series of cicode functions. This gives users a lot flexibility when engineering their system. Whether there is need to access data from a web service, or push data to or from a 3rd party system, there are numerous examples on how that can be done via the dot net framework on the internet.

For users that have no experience writing Dot Net code, this document will lead the user through the process of creating a simple dot net assembly and calling that code from cicode. For more advanced users, examples will be shown demonstrating how functions from the dot net framework can be called directly, and how the dot net framework can be used to build more complex solutions.

This document is designed to complement the contents of the Vijeo Citect help. For detailed information regarding the cicode functions regarding the functions that are going to be discussed in this document, please refer to the Vijeo Citect help.

1.2. Terminology

Term	Meaning
Assembly	An assembly is a file that is automatically created when a C# project is successfully compiled. This can either be an executable or a dynamic linked library (dll)
Class	A data type that describes an object. Classes contain both data, and the methods for acting on the data
Method	A named code block that provides behavior for a class or struct .
Property	Properties are members that provide a flexible mechanism to read, write, or compute the values of private fields.
Member	A field, property, method, or event declared on a class or struct .
Struct	A compound data type that is typically used to contain a few variables that have some logical relationship. Structs can also contain methods and events. Structs do not support inheritance but they do support interfaces

Static	A class or method declared as static exists without first being instantiated using the keyword new. The .Net Assembly functions do not support static classes or methods.
Object	An instance of a class . An object exists in memory, and has data and methods that act on the data.
MSDN	Microsoft developer network. This site contains the documentation and examples for the .Net framework. This can be accessed at http://msdn.microsoft.com
Namespace	The namespace keyword is used to declare a scope that contains a set of related objects. You can use a namespace to organize code elements and to create globally unique types.
Constructor	A special method on a class or struct that initializes the objects of that type.

1.3. Cicode functions used

This whitepaper will details the user of a series of cicode functions that will allow you to access Dot Net Assemblies. Before reading this whitepaper, please refer to the help topics for these functions.

- DIIClassCreate
- DIIClassCallMethod
- DIIClassGetProperty
- DIIClassDispose

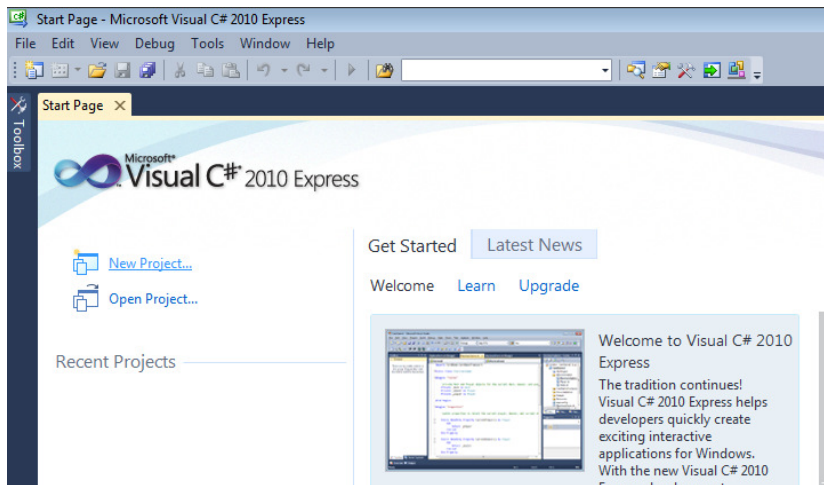
2. Hello World (C#)

This example will lead you through creating a basic c# assembly that will be called from cicode. In this example, I will be using visual studio 2010 express. Microsoft offers a free 90 day trail version is this product. This can be downloaded from http://www.visualstudio.com/downloads/download-visual-studio-vs#DownloadFamilies_4

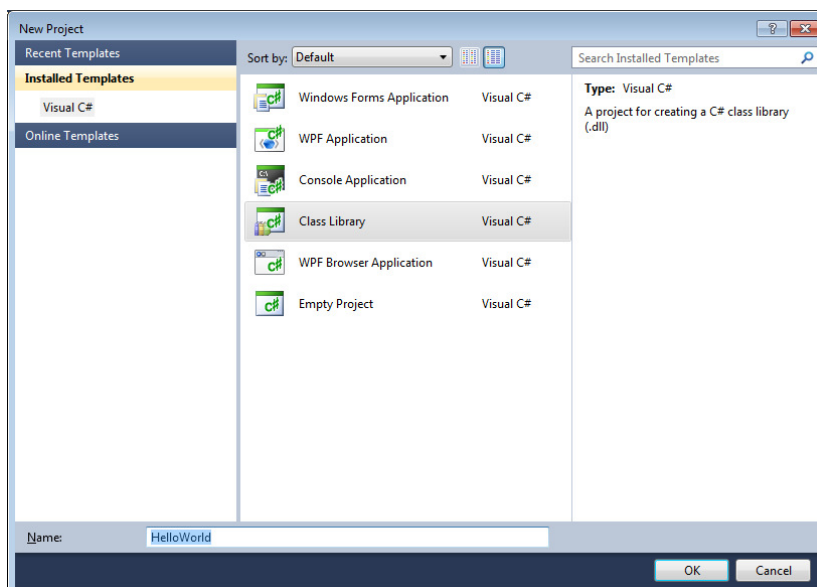
The C# code for this example can be found in Appendix A: “Hello World!” C# code.

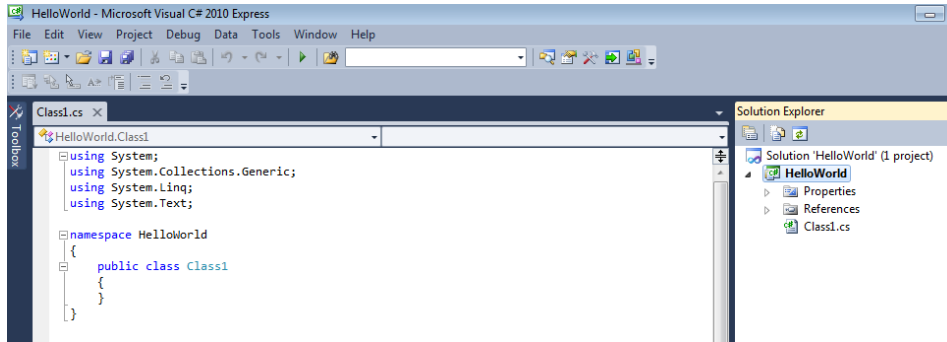
2.1. Creating a new class library

The first thing that we need to is to start visual studio and create a new project.



In the new project dialog, select “Class Library” and enter the name “HelloWorld”, and then press ‘OK’.



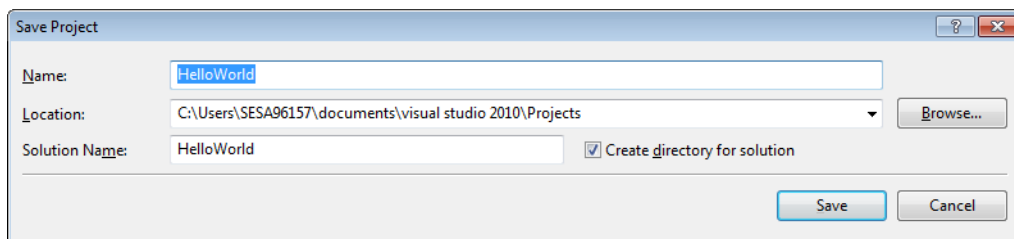


Rename the class “Class1” to “HelloWorld”. This will leave you with the following code.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorld
{
    public class HelloWorld
    {
    }
}
```

Save your code by clicking on the “Save all” button.



2.2. Constructor

To be able to create an instance of our class, our class must contain a constructor. Each non-static class can contain one or more constructors. As described in section 2.1, each constructor must contain a different set of parameters. In this case, the constructor for our class will contain no parameters.

We will also contain a private field of the data type string that will contain the message that we are going to return. In the constructor, we will set the value of this field to “Hello World!”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorld
{
    public class HelloWorld
    {
        private string _helloWorldString;
    }
}
```

```
public HelloWorld()  
{  
    _helloWorldString = "Hello World!";  
}  
}
```

2.3. Properties

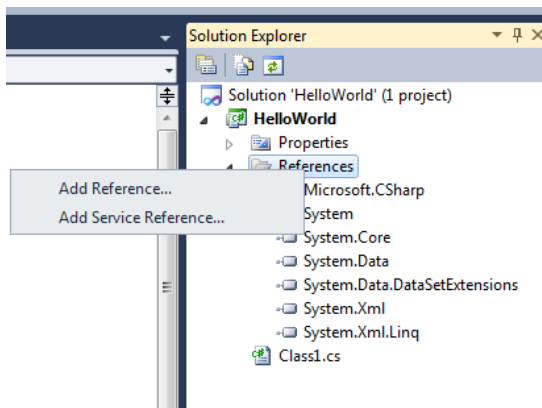
Properties allow you to read or write values from private fields. Properties can be read by defining a get method and can be written to using a set method. In this case, we want to add a public property to our class that will return the contents of the string `_helloWorldString`, and we will define both a get and set methods, so that we can read from and write to this property.

```
public string Message  
{  
    get  
    {  
        return _helloWorldString;  
    }  
    set  
    {  
        _helloWorldString = value;  
    }  
}
```

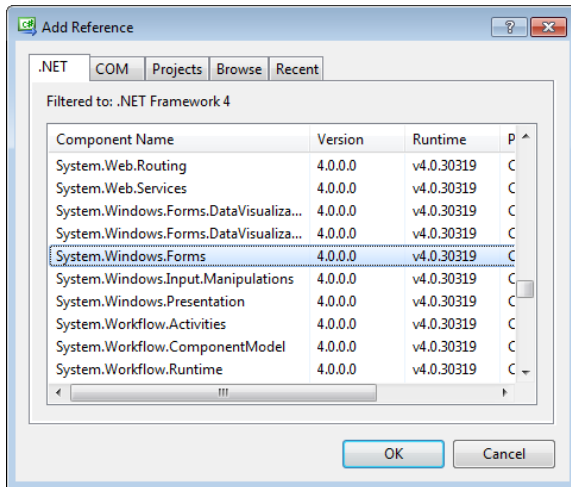
2.4. Methods

A method is a group of statements that perform a task. Similar to a cicode function, a method has a series of parameters that are passed into it and can optionally return value.

In this example, we will create a method that will display a message box that contains the contents of the property "Message". To display the message box, we will call a method from the Dot Net framework. To use this functionality, we will need to add a reference to our project. To do this, right click on "References" in solution explorer and select "Add Reference..."



In the “Add Reference” windows, click on the “.Net” tab, and scroll down until you find “System.Windows.Forms”, then press “OK”.



Now that we have added a reference to System.Windows.Forms, we will use the “using” statement to allow our code to access all the members of that namespace.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

Finally, we can use create our method.

```
public void DisplayMessage()
{
    MessageBox.Show(Message);
}
```

We can now build our assembly by right clicking on your solution in solution explorer, and selecting “Build Solution”. This will output the file HelloWorld.dll into the directory \bin\Release\

3. Hello World (Cicode)

Now that we have built a C# assembly, we can now write some cicode to access it. I'll show two examples of how this can be done. The first example will create an instance of our HelloWorld class, get the "Message" property and then use the cicode function "Message" to display a message box displaying "Hello World!" The second example will also create an instance of the class, HelloWorld, will set the value of the Message property to "Hello World!!!!!" and will call the DisplayMessage method to display the message box. Both of these functions have some similar code.

The full cicode for both of these examples can be found in Appendix C: "Hello World!" cicode.

3.1. Creating an instance of the class

Both functions need to create an instance of the class "HelloWorld" using DllClassCreate. This function requires the full path to the assembly being called, and the name of the class. As the class's constructor contains no input parameters, we don't need to pass any additional parameters into DllClassCreate. Once this has been done, we need to check if the class has been created successfully using DllClassIsValid.

```
FUNCTION HelloWorld1()  
  
    //Handle to the instance of our class, hello world.  
    OBJECT hHelloWorld;  
    ...  
    //Create an instance of our class  
    hHelloWorld = DllClassCreate(PathToStr("[RUN]:HelloWorld.dll"), "HelloWorld");  
    //If this class is valid  
    IF DllClassIsValid(hHelloWorld) = 1 THEN  
        ...
```

3.2. Getting a property

Property values can be retrieved from this class by calling DllClassGetProperty. As this property returns as string, the value can be returned into a cicode string (sMessage).

```
IF DllClassIsValid(hHelloWorld) = 1 THEN  
    //retrieve the contents of the Message property  
    sMessage = DllClassGetProperty(hHelloWorld, "Message");  
    Message("Property from .Net class", sMessage, 0);
```

3.3. Setting a property

Property values can be set in this object by calling `DllClassSetProperty`. In this case, we will set the value of the `Message` property to be equal to the value of the string `sMessage`.

```
IF DllClassIsValid(hHelloWorld) = 1 THEN
    //Set the value of the message property
    DllClassSetProperty(hHelloWorld, "Message", sMessage);
```

3.4. Calling a method

Methods can be called from our class by calling `DllClassCallMethod`, and passing the name of the method into it. In this example, we can display the message box using the method "DisplayMessage".

```
//If this class is valid
IF DllClassIsValid(hHelloWorld) = 1 THEN
    //Set the value of the message property
    DllClassSetProperty(hHelloWorld, "Message", sMessage);
    //and display the message by calling the method DisplayMessage
    DllClassCallMethod(hHelloWorld, "DisplayMessage");
```

Note that in this case, cicode will block on the `DllClassCallMethod` call until the message box has been closed.

3.5. Limitations

The .Net functions do not support static methods and classes, and do not support events. For this reason, it is not recommended that you do not use the .Net Functions to create user interfaces from within cicode.

4. Calling predefined functions in windows

Not all cases will require you to create a c# assembly to use the Dot Net cicode functions. For simple operations, you can directly call the methods from the dot net framework. In this section, I will use functionality that is built into the .Net Framework to measure the CPU Usage on my machine. I will use the "PerformanceCounter" class from the namespace System.Diagnostics to measure the PC's CPU usage using the performance counter "Processor / % Processor Time". This class can be used to measure the same set of data as windows performance monitor.

```
OBJECT oPerformanceCounter;

REAL FUNCTION GetPercentProcessorTime ()

    STRING sDllPath;
    STRING sCategoryName = "Processor";
    STRING sCounter = "% Processor Time";
    STRING sInstance = "_Total";
    REAL rProcessorTime = -1;
    STRING sCounterHelp;
    INT iError;

    //Enable error checking
    ErrSet(1);
    sDllPath = "C:\Windows\Microsoft.NET\Framework64\v4.0.30319\system.dll";
    //Create an instance of the PerformanceCounter class from the namespace System.Diagnostics
    //You do not need to include the class's namespace
    oPerformanceCounter=DllClassCreate(sDllPath, "PerformanceCounter", sCategoryName, sCounter, sInstance);
    //If the class is valid
    IF DllClassIsValid(oPerformanceCounter) THEN
        //Call the required method
        rProcessorTime = DllClassCallMethod(oPerformanceCounter, "NextValue");
        //Get the value of a property
        sCounterHelp = DllClassGetProperty(oPerformanceCounter, "CounterHelp");
        ErrLog(sCategoryName+"\\"+sCounter+"\\"+sInstance+": "+RealToStr(rProcessorTime, 4, 1));
        ErrLog(sCounterHelp);
    ELSE
        ErrLog("GetPercentProcessorTime: Class is not valid");
    END
    //check for error codes;
    iError = IsError();
    IF iError <> 0 THEN
        ErrLog("GetPercentProcessorTime: Error code "+iError:"#+" was returned.");
    END

    DllClassDispose(oPerformanceCounter);

    RETURN rProcessorTime;

END
```

4.1. Creating an object

In the above example, `DllClassCreate` is called to create an instance of the “PerformanceCounter” class from the namespace `System.Diagnostics`. This is done using the cicode:

```
//Create an instance of the PerformanceCounter class from the namespace System.Diagnostics
//You do not need to include the class's namespace
oPerformanceCounter=DllClassCreate(sDllPath, "PerformanceCounter", sCategoryName, sCounter, sInstance);
```

Referring to the MSDN, the `PerformanceCounter` class has six possible constructors:

- `PerformanceCounter()`
- `PerformanceCounter(string categoryName, string counterName)`
- `PerformanceCounter(string categoryName, string counterName, bool readOnly)`
- `PerformanceCounter(string categoryName, string counterName, string instanceName)`
- `PerformanceCounter(string categoryName, string counterName, string instanceName, bool readOnly)`
- `PerformanceCounter(string categoryName, string counterName, string instanceName, string machineName)`.

Unlike other similar functions, such as `DllOpen` or `TaskNew`, `DllClassCreate` (and `DllClassCallMethod`) supports a *variable* number of function arguments. This means that you no longer need to concatenate all of a functions arguments together into one string, and use the carat character (^) to delimit any string arguments.

When using `DllClassCreate` simply pass the same number of arguments as the class’s constructor i.e. to instantiate the `PerformanceCounter` class using each of the above constructors, call the following cicode:

- `oPerformanceCounter=DllClassCreate(sDllPath, "PerformanceCounter");`
- `oPerformanceCounter=DllClassCreate(sDllPath, "PerformanceCounter", sCategoryName, sCounter);`
- `oPerformanceCounter=DllClassCreate(sDllPath, "PerformanceCounter", sCategoryName, sCounter, 1);`
- `oPerformanceCounter=DllClassCreate(sDllPath, "PerformanceCounter", sCategoryName, sCounter, sInstance);`
- `oPerformanceCounter=DllClassCreate(sDllPath, "PerformanceCounter", sCategoryName, sCounter, sInstance, 1);`
- `oPerformanceCounter=DllClassCreate(sDllPath, "PerformanceCounter", sCategoryName, sCounter, sInstance, sMachineName);`

`DllClassCreate`’s path parameter needs to refer to the actual assembly containing the class that you want to instantiate. If you are planning on calling .Net assemblies directly, then this needs to refer to the .Net assembly containing this class. For more information on where this is located, please refer to the MSDN.

Remember, that for each instance of a class that you create, you must call `DllClassDispose` to dispose of that object. Failure to do this may result in memory leaks or other undefined behavior.

4.2. Supported Data Types

The .Net framework supports a large number of numeric data types, much more than are supported by cicode. The following table shows how .Net data types can be cast into cicode data types. These conversions apply to DIIClassCallMethod, DIIClassGetProperty and DIIClassSetProperty.

	Int	Real
Bool	True	False
Byte	True	False
Decimal	False	False
Double	False	False
Float	False	True
Int	True	False
Long	True	False
SByte	True	False
UINT	True	False
Ulong	True	False
Ushort	True	False
Int16	True	False
Int32	True	False
UInt16	True	False
UInt32	True	False
UInt64	False	False
short	True	False

The .Net assembly functions do not support casting a .Net numeric type into a string data type in cicode. To convert numeric types into strings, use the cicode functions IntToStr() or RealToStr().

5. Example: Creating an alarm paging system

This section will explain how the .Net assembly functions can be used to create an alarm paging system. This section will detail:

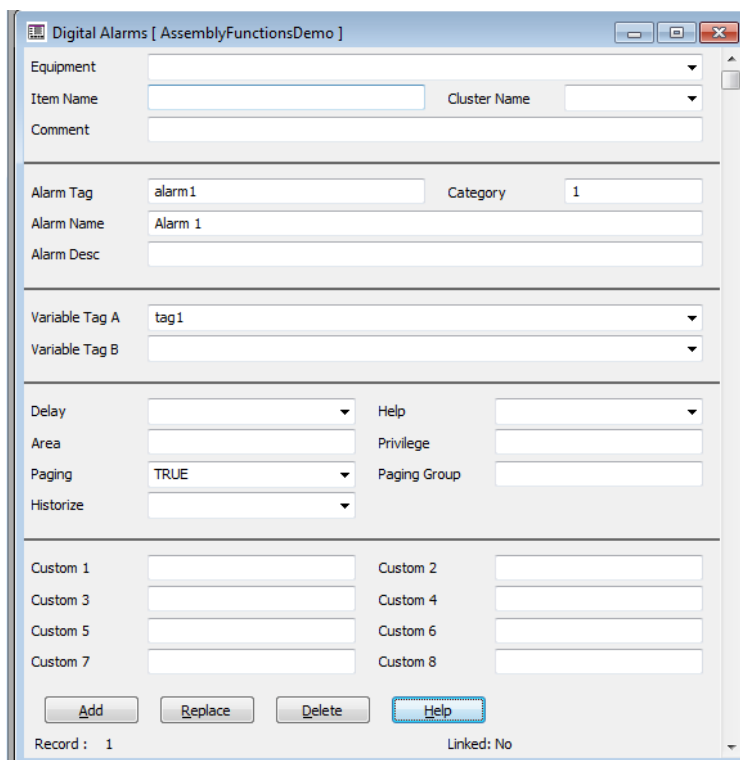
- Configuring the paging and paging group fields
- Setting up the alarm event queue.
- Sending alarm notifications by email.

5.1. Configuring the Paging and Paging Group fields

Every type of alarm in Vijeo Citect contains two fields, which can be configured to track how each alarm is paged.

- Paging – A true/false value that indicates whether the alarm will be pages.
- Paging Group – a string field that is used to indicate which paging group the alarm belongs. You could use this field to control what set of users to page.

Both of these fields appear on the extended form of all alarms types. In this example, the paging group will be not be used.



The screenshot shows a software window titled "Digital Alarms [AssemblyFunctionsDemo]". The window contains several input fields and dropdown menus for configuring an alarm. The fields are organized into sections:

- Equipment Section:** Includes "Equipment" (text input), "Item Name" (text input), "Cluster Name" (dropdown menu), and "Comment" (text input).
- Alarm Tag Section:** Includes "Alarm Tag" (text input with value "alarm1"), "Category" (text input with value "1"), "Alarm Name" (text input with value "Alarm 1"), and "Alarm Desc" (text input).
- Variable Tag Section:** Includes "Variable Tag A" (dropdown menu with value "tag1") and "Variable Tag B" (dropdown menu).
- Configuration Section:** Includes "Delay" (dropdown menu), "Area" (text input), "Paging" (dropdown menu with value "TRUE"), "Historize" (dropdown menu), "Help" (dropdown menu), "Privilege" (text input), and "Paging Group" (text input).
- Custom Fields Section:** Includes eight custom fields labeled "Custom 1" through "Custom 8", each with a text input.

At the bottom of the window, there are four buttons: "Add", "Replace", "Delete", and "Help". Below the buttons, it shows "Record : 1" and "Linked: No".

5.2. Alarm Event Queue

The alarm event queue is an existing Vijeo Citect feature that allows to be read as they are triggered by the alarm server. The alarm event queue can be enabled by setting the parameter:

- [Alarm] EventQue = 1

The format of the alarm event queue can be controlled by setting the parameter:

- [Alarm] EventFmt =
{Name,48},{Time,12},{Category,3},{STATE,16},{PAGING,4},{PAGINGGROUP,80}

For more details on these parameters, refer to the online help.

To process the alarm event queue, we write a cicode function that will run on the alarm server. This will read the contents of the alarm event queue, parse the contents of the queue, and then either send an email or an SMS based on certain criteria:

- Paging = 1, State = ON, PagingGroup = Email
- Paging = 1, State = ON, PagingGroup = SMS

The alarm event queue can be processed using the following functions:

- QueOpen
- QueRead
- QueLength

For details on these functions please refer to the online help. The full contents of the cicode that is used to read and parse the alarm event queue can be found in Appendix A: ProcessEventQue cicode function.

```
WHILE 1 DO
//Use QueOpen to open the Alarm Event Que
hQueue= QueOpen("EventQue",0)
Sleep(1);
IF hQueue<> -1 THEN
//If the queue is not empty
iQueLength = QueLength(hQueue);
IF iQueLength > 0 THEN
iQueStatus = 0;
WHILE iQueStatus = 0 DO
//Read each entry from the queue
iQueStatus = QueRead(hQueue,iRecord, sAlarmFmt,1)
```

In this example the variable sAlarmFmt will return a string in the format that is specified in the parameter [Alarm] EventQueFmt. You will now need to parse this string using the string functions to retrieve the various field information.

Once the relevant information has been read from the queue, and the appropriate checks have been made, the alarm information be send via either email or SMS.

```

IF ((sPaging = "TRUE") AND (iCategory = 2) AND (sState = "ON")) THEN
    SendEmail("graeme.davey@schneider-electric.com",sAlarmName,"TIME: "+sTime+" STATE: "+sState);
END

```

5.3. Sending an alarm notification via email

This example will explain how to use the Dot Net assembly functions to send an email from a web based email account, such as gmail, yahoo or outlook.com. This will be done by directly calling assemblies from the Dot Net Framework.

```

INT
FUNCTION SendEmail(STRING sTo, STRING sSubject, STRING sMessage, STRING sFrom, STRING sPassword, STRING
sSMTPServer)
    STRING sPath = "C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.0\Profile\Client\System.dll";
    OBJECT hCredentialsProxy;
    OBJECT hSmtpClientProxy;
    INT result;
    INT iError = 0;
    hSmtpClientProxy = DllClassCreate(sPath, "SmtpClient", sSMTPServer);

    IF DllClassIsValid(hSmtpClientProxy) = 1 THEN
        DllClassSetProperty(hSmtpClientProxy, "Port", 587)
        DllClassSetProperty(hSmtpClientProxy, "UseDefaultCredentials", 0)
        DllClassSetProperty(hSmtpClientProxy, "EnableSsl", 1);
        hCredentialsProxy = DllClassCreate(sPath, "NetworkCredential", sFrom, sPassword);
        IF DllClassIsValid(hCredentialsProxy) THEN
            DllClassSetProperty(hSmtpClientProxy, "Credentials", hCredentialsProxy);
            result= DllClassCallMethod(hSmtpClientProxy, "Send", sFrom, sTo, sSubject, sMessage);
            iError = IsError();
        END
    END
    DllClassDispose(hSmtpClientProxy);
    DllClassDispose(hCredentialsProxy);

    RETURN iError;
END

```

This function sends an email by creating an instance of the class, SMTPClient from the namespace System.Net.Mail and a NetworkCredentials object from System.Net to handle the user name and password. Help for these classes can be found at:

NetworkCredentials: [http://msdn.microsoft.com/en-us/library/system.net.networkcredential\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.net.networkcredential(v=vs.110).aspx)

SMTPClient: [http://msdn.microsoft.com/en-us/library/system.net.mail.smtpclient\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/system.net.mail.smtpclient(v=vs.110).aspx)

In this example, I've first created an SMTPClient using DllClassCreate, and passed SMTP server address to it. This creates an instance of this class, that I can then set the relevant properties, so that I can establish communications to the SMTP server.

```

hSmtpClientProxy = DllClassCreate(sPath, "SmtpClient", sSMTPServer);

```

Once the class has been created, then next step is check if the class has been created successfully or not. This can be done using the function `DllClassIsValid`. This function will return 1, if the instance of the class has been created.

```
IF DllClassIsValid(hSmtplibClientProxy) = 1 THEN
```

Now that I know that the class is valid. I can now set some property values, using `DllClassSetProperty`. In this example, I will set four properties. The first three properties are simple data types, so they can be set directly using `DllClassSetProperty`. i.e.

```
DllClassSetProperty(hSmtplibClientProxy, "Port", 587)  
DllClassSetProperty(hSmtplibClientProxy, "UseDefaultCredentials", 0)  
DllClassSetProperty(hSmtplibClientProxy, "EnableSsl", 1);
```

Property	Data Type	Value
EnableSsl	Boolean	True
Port	Int	587
UseDefaultCredentials	Boolean	False
Credentials	System.Net.NetworkCredential	*

The fourth property “Credentials” needs to be set to an instance of a class, in this case, a `System.Net.NetworkCredentials` object. To set this property, I create an instance of this class using `DllClassCreate`, passing the relevant user name and password into the function. After checking that this is valid, the handle to this object can be used within `DllClassSetProperty`.

```
hCredentialsProxy = DllClassCreate(sPath, "NetworkCredential", sFrom, sPassword);  
IF DllClassIsValid(hCredentialsProxy) THEN  
    DllClassSetProperty(hSmtplibClientProxy, "Credentials", hCredentialsProxy);  
...  
...
```

The email can now be sent using the “Send” method, passing in the relevant information by using `DllClassCallMethod`

```
result= DllClassCallMethod(hSmtplibClientProxy, "Send", sFrom, sTo, sSubject, sMessage)
```

Finally, we need to dispose of the `SMTPClient` and `NetworkCredential` objects, by calling `DllClassDispose`.

```
DllClassDispose(hSmtplibClientProxy);  
DllClassDispose(hCredentialsProxy);
```

Using the Dot Net Assembly functions is to directly call into assemblies from the Dot Net framework is only recommended for simple operations. For most operations, it will be more efficient to write your own Dot Net class,

and call the relevant methods and properties from that class using the Dot Net Assembly functions. This will be more efficient as:

- It will allow you to write less total code
- Easier Debugging.
- It will allow you to access a wider variety of classes and data types.
- It will allow you to access static classes and methods.

6. Logging

Often while configuring and debugging your C# code, issues will occur. Your properties may not return the correct values, incorrect parameters may be passed into methods or exceptions may occur.

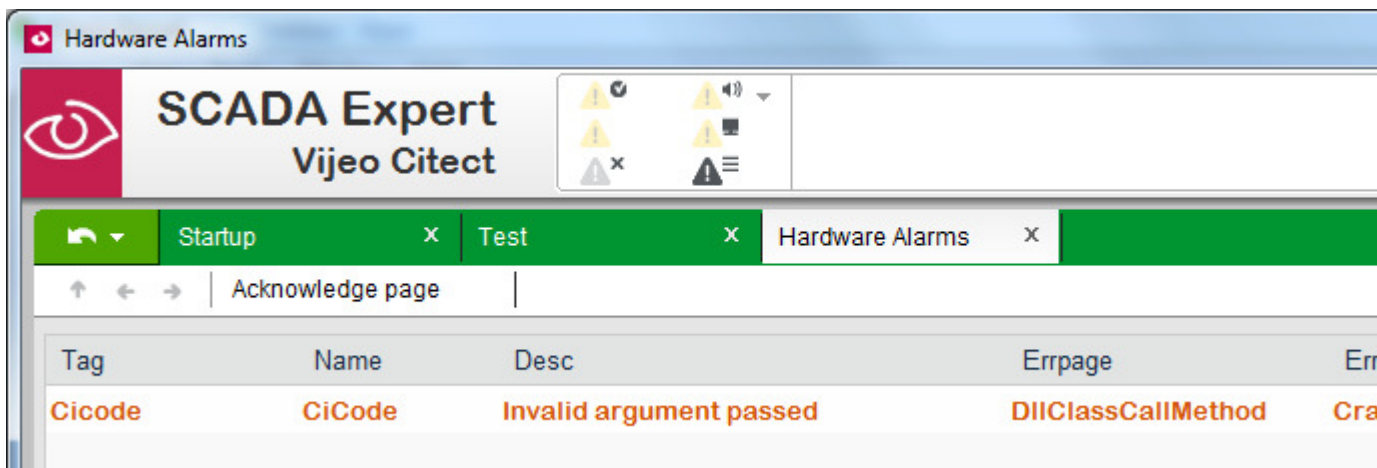
To troubleshoot these types of issues, logging can be enabled that will catch all calls to the Dot Net Assembly functions. This logging can be enabled for setting the following parameters in your Citect.ini file. *It is recommended that you only set these parameters will configuring and testing your project, and not on a running system.*

- [Debug] CategoryFilter = ManagedUtil
- [Debug] CategoryFilterMode = 0
- [Debug] Critical, Error, Information, Verbose, Warning
- [Debug] Priority = 30

By configuring these settings, detailed information on every call that is made to your dot net assembly will be logged to the tracelog.dat file, which is contained in your \Logs directory.

6.1. Unhandled Exceptions

When an exception is thrown by your assembly, the exception will be caught by Vijeo Citect 2015 and an "Invalid Argument passed" hardware alarm should be raised.



The tracelog.dat file, will also contain some useful information regarding the exception.

```
... ManagedUtil    DIIProxyManager::CreateProxyTasks dllPath=C:\Program Files (x86)\Schneider Electric\Vijeo Citect  
7.50\Bin\CrashTest.dll className=CrashTest hProxy=0
```

```
... ManagedUtil    DIIProxy::DLLProxy method=Crash ex=System.Reflection.TargetInvocationException: Exception has been  
thrown by the target of an invocation. ---> System.Exception: This exception should not cause runtime to crash
```

```

... ManagedUtil      at CrashTest.CrashTest.Crash()
... ManagedUtil      --- End of inner exception stack trace ---
... ManagedUtil      at System.RuntimeMethodHandle.InvokeMethod(Object target, Object[] arguments, Signature sig, Boolean
constructor)
... ManagedUtil      at System.Reflection.RuntimeMethodInfo.UnsafeInvokeInternal(Object obj, Object[] parameters, Object[]
arguments)
... ManagedUtil      at System.Reflection.RuntimeMethodInfo.Invoke(Object obj, BindingFlags invokeAttr, Binder binder, Object[]
parameters, CultureInfo culture)
... ManagedUtil      at System.RuntimeType.InvokeMember(String name, BindingFlags bindingFlags, Binder binder, Object
target, Object[] providedArgs, ParameterModifier[] modifiers, CultureInfo culture, String[] namedParams)
... ManagedUtil      at SE.SCADA.ManagedUtil.DllProxy.MethodCall(String method, List`1 argList, Int16& error)
... ManagedUtil      DllProxyManager::MethodCallTasks hProxy=0 method=Crash result=

```

6.2. Unsupported data types

When calling a method or accessing a property that returns an unsupported data type, the results of the call can be viewed in the `tracelog.dat` file. i.e. If I attempt to read the minimum value of a Long in C# by calling `Long.Min()`, this will return `-9223372036854775808`. As a long in C# is 64 bit, but cicode INT's are only 32-bits, this data will not be able to be passed back into a cicode variable. The value can however be seen in the `tracelog.dat` file:

```

... ManagedUtil      DllProxyManager::CreateProxyTasks dllPath=D:\ProgramData\Schneider Electric\Vijeo
Citect 7.50\User\DotNetDataTypes\DotNetDataTypeTests.dll className=DotNetDataTypeTests hProxy=17
... ManagedUtil      DllProxyManager::GetProxyPropertyTasks hProxy=17 propertyName=LongMin result=-
9223372036854775808

```

7. Conclusion

By giving Citect users the ability to access Microsoft's dot net framework, Citect users now have can easily pass data to and from external systems. The dot net framework is powerful and well documented. Example code can easily be found to solve any number of problems, meaning that solutions to complex problems can be engineered much more quickly.

For example, web services exist that provide information on everything from weather services, energy prices, or that allow users to send SMS's via web based SMS gateways. By doing a little research, and writing a small amount of C# code, these services can quickly be prototyped and integrated into Citect solutions.

8. Appendix A: “Hello World!” C# code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace HelloWorld
{
    public class HelloWorld
    {
        private string _helloWorldString;
        public HelloWorld()
        {
            _helloWorldString = "Hello World!";
        }
        public string Message
        {
            get
            {
                return _helloWorldString;
            }
            set
            {
                _helloWorldString = value;
            }
        }
        public void DisplayMessage()
        {
            MessageBox.Show(Message);
        }
    }
}
```

9. Appendix B: “Hello World!” cicode

```
// -----  
// File: HelloWorld.ci  
// -----  
//  
// -----  
// Hello World! examples for calling dot net code from cicode  
//  
// -----  
// HelloWorld1  
// -----  
//  
// Description:      Creates an instance of the HelloWorld class, retrieves  
//                  Message property and displays that using cicode function  
//                  Message to display the message in a message box.  
// -----  
FUNCTION HelloWorld1()  
  
    //Handle to the instance of our class, hello world.  
    OBJECT hHelloWorld;  
    STRING sMessage;  
    //Create an instance of our class  
    hHelloWorld = DllClassCreate(PathToStr("[RUN]:HelloWorld.dll"), "HelloWorld");  
    //If this class is valid  
    IF DllClassIsValid(hHelloWorld) = 1 THEN  
        //retrieve the contents of the Message property  
        sMessage = DllClassGetProperty(hHelloWorld, "Message");  
        Message("Property from .Net class", sMessage, 0);  
    END  
    //dispose our instance of the class  
    DllClassDispose(hHelloWorld)  
END  
  
// -----  
// HelloWorld2  
// -----  
//  
// Description:      Creates an instance of the HelloWorld class, sets the  
//                  Message property and then calls the method DisplayMessage  
//                  to display the message in a message box.  
// -----  
FUNCTION HelloWorld2()  
  
    //Handle to the instance of our class, hello world.  
    OBJECT hHelloWorld;  
    STRING sMessage = "Hello World !!!!!";  
    //Create an instance of our class  
    hHelloWorld = DllClassCreate(PathToStr("[RUN]:HelloWorld.dll"), "HelloWorld");  
    //If this class is valid  
    IF DllClassIsValid(hHelloWorld) = 1 THEN  
        //Set the value of the message property  
        DllClassSetProperty(hHelloWorld, "Message", sMessage);  
        //and display the message by calling the method DisplayMessage  
        DllClassCallMethod(hHelloWorld, "DisplayMessage");  
    END  
END
```

```
END
//dispose our instance of the class
DllClassDispose(hHelloWorld)
    END
```

10. Appendix C: ProcessEventQuee()

```
FUNCTION ProcessEventQueuee ()
```

```
    INT hQueue;  
    INT iRecord;  
    STRING sAlarmFmt;  
    STRING sRemaining;  
    STRING sAlarmName;  
    STRING sTime;  
    INT iCategory;  
    STRING sState;  
    INT iSendCategory = 2;  
    INT iQueStatus;  
    INT iQueLength;  
    STRING sPaging;  
    STRING sPagingGroup;
```

```
    WHILE 1 DO
```

```
        hQueue= QueOpen("EventQuee",0)
```

```
        Sleep(1);
```

```
        IF hQueue<> -1 THEN
```

```
            iQueLength = QueLength(hQueue);
```

```
            IF iQueLength > 0 THEN
```

```
                iQueStatus = 0;
```

```
                WHILE iQueStatus = 0 DO
```

```
                    iQueStatus = QueRead(hQueue,iRecord, sAlarmFmt,1)
```

```
                    //sAlarmFmt will now return a string in the format that was specified in
```

```
                    //the parameter [Alarm]EventQueeFmt
```

```
                    //In this example, this will be
```

```
                    // {fill this in later when your debugging}
```

```
                    IF sAlarmFmt <> "" THEN
```

```
                        sAlarmName = StrLeft(sAlarmFmt ,StrSearch(0, sAlarmFmt,""));
```

```
                        sRemaining = StrRight(sAlarmFmt,StrLength(sAlarmFmt)-StrSearch(0,
```

```
sAlarmFmt,"")-1);
```

```
                        sTime= StrLeft(sRemaining ,StrSearch(0, sRemaining ,","));
```

```
sRemaining ,",")-1);
```

```
                        sRemaining = StrRight(sRemaining ,StrLength(sRemaining)-StrSearch(0,
```

```
sRemaining ,",")-1);
```

```
                        iCategory = StrToInt(StrLeft(sRemaining ,StrSearch(0, sRemaining ,",")));
```

```
                        sRemaining = StrRight(sRemaining ,StrLength(sRemaining)-StrSearch(0,
```

```
sRemaining ,",")-1);
```

```
                        sState = StrLeft(sRemaining ,StrSearch(0, sRemaining ,","));
```

```
                        sRemaining = StrRight(sRemaining ,StrLength(sRemaining)-StrSearch(0,
```

```
sRemaining ,",")-1);
```

```
                        sPaging = StrLeft(sRemaining ,StrSearch(0, sRemaining ,","));
```

```
                        sPagingGroup = StrRight(sRemaining ,StrLength(sRemaining)-StrSearch(0,
```

```
                        IF ((sPaging = "TRUE") AND (iCategory = 2) AND (sState = "ON")) THEN
```

```
                            SendEmail("me@myemail.com",sAlarmName,"TIME: "+sTime+" STATE:
```

```
                            "+sState);
```

```
                        END
```

```
                    END
```

```
                END
```

```
            END
```

```
        END
```

```
    END
```

```
END
```

11. References

C# Terminology [https://msdn.microsoft.com/en-us/library/ms173231\(v=vs.90\).aspx](https://msdn.microsoft.com/en-us/library/ms173231(v=vs.90).aspx)

Visual Studio 2010 Express http://www.visualstudio.com/downloads/download-visual-studio-vs#DownloadFamilies_4

NetworkCredential

SMTPClient

PerformanceCounter

Namespaces <https://msdn.microsoft.com/en-us/library/z2kcy19k.aspx>

Cicode references

DllclassCreate

DllClassCallMethod

DllClassSetProperty

dllclassGetProperty

DllClassDispose

AlarmEventQue

