VBUC 10.1.0 – External Release

C# Features Implementation

Three different features were implemented in order to improve the code generation when selecting C#.

Auto Default Structs

Any field or property not initialized by a constructor is automatically initialized by the compiler using its default value, with this change, the create instance is not necessary, allowing the VBUC to generate a cleaner and friendlier code.

VB6 Code:

```
Type Person
    Name As String
    LastName As String
    Age As Integer
End Type

Sub main()
    Dim P1 As Person
End Sub
```

C# Previous Code:

```
[Serializable]
5 references
public struct Person
{
    public string Name;
    public string LastName;
    public short Age;
    1 reference
    public static Person CreateInstance()
    {
        Person result = new Person();
        result.Name = String.Empty;
        result.LastName = String.Empty;
        return result;
    }
}

[STAThread]
0 references
public static void Main()
{
    Person P1 = Person.CreateInstance();
}
```

C# Current Code:

```
[Serializable]
3 references
public struct Person
{
    public string Name;
    public string LastName;
    public short Age;
    1 reference
    public Person()
    {
        Name = String.Empty;
        LastName = String.Empty;
    }
}

[STAThread]
0 references
public static void Main()
{
    Person P1 = new Person();
}
```

Global Usings

Global usings are intended to simplify and lighten C# code, by declaring only once a using keyword for a specific namespace in a project, so then you won't need to declare in another file which needs the same using.

To do this, a new file named "GlobalUsings" will be generated in the solution, in which these shared using declarations for the project will be located.

C# Previous Code:

```
using System;
using System.Net.Http;
using System.Xml;

namespace Project1
{
    0 references
    internal static class Module1
    {
```

```
using System;
using System.Net.Http;

namespace Project1
{
    0 references
    internal static class Module2
    {
```

C# Current Code:

```
global using System;
global using System.Windows.Forms;
global using System.Net.Http;
global using System.Xml;
```

Expression Bodied Members

Expression body definitions allow you to provide the implementation of a member in a concise and readable form when the member has a single statement.
Note: This feature applies to different members in the code; however, there is an upgrade option to disable this feature if you prefer to have the old style.

Methods

VB6 Code:

```
Function Add(P1 As Double, P2 As Double) As Double
    Add = P1 + P2
End Function
```

```
Sub Square(P1 As Double)
    P1 = P1 * P1
End Sub
```

C# feature turned off:

```
1 reference
internal static double Add(double P1, double P2)
{
    return P1 + P2;
}

1 reference
internal static void Square(ref double P1)
{
    P1 *= P1;
}
```

C# feature turned on:

```
1 reference
internal static double Add(double P1, double P2) => P1 + P2;


1 reference
internal static void Square(ref double P1) => P1 *= P1;
```

Events

VB6 Code:

```
Private Sub TreeView1_Collapse(ByVal Node As MSComctlLib.Node)
    'Event Collapse
End Sub
```

```
Private Sub TreeView2_Collapse(ByVal Node As MSComctlLib.Node)
    Debug.Print "Event Collapse"
End Sub
```

```
Private Sub TreeView3_Collapse(ByVal Node As MSComctlLib.Node)
    Node.BackColor = vbRed
End Sub
```

C# feature turned off:

```
1 reference
private void TreeView1_AfterCollapse(Object eventSender, TreeViewEventArgs eventArgs)
{
    _ = eventArgs.Node;
    //Event Collapse
}

1 reference
private void TreeView2_AfterCollapse(Object eventSender, TreeViewEventArgs eventArgs)
{
    _ = eventArgs.Node;
    Debug.WriteLine("Event Collapse");
}

1 reference
private void TreeView3_AfterCollapse(Object eventSender, TreeViewEventArgs eventArgs)
{
    TreeNode Node = eventArgs.Node;
    Node.BackColor = Color.Red;
}
```

C# feature turned on:

```
1 reference
private void TreeView1_AfterCollapse(Object eventSender, TreeViewEventArgs eventArgs) => _ = ev


1 reference
private void TreeView2_AfterCollapse(Object eventSender, TreeViewEventArgs eventArgs)
{
    _ = eventArgs.Node;
    Debug.WriteLine("Event Collapse");
}

1 reference
private void TreeView3_AfterCollapse(Object eventSender, TreeViewEventArgs eventArgs)
{
    TreeNode Node = eventArgs.Node;
    Node.BackColor = Color.Red;
}
```

Properties

VB6 Code:

```
Private m_name As String
Private m_age As String
```

```
Public Property Get Name() As String
    Name = m_name
End Property
```

```
Public Property Let Name(newName As String)
    m_name = newName
End Property
```

```
Public Property Get Age() As String
    Age = m_age
End Property
```

```
Public Property Let Age(newValue As String)
    Dim newValue2 As Integer
    newValue2 = CInt(newValue)
    If newValue2 > 0 And newValue2 < 12 Then
        m_age = "Enfant"
    ElseIf newValue2 >= 12 And newValue2 < 18 Then
        m_age = "Teenager"
    ElseIf newValue2 >= 18 And newValue2 < 60 Then
        m_age = "Adult"
    Else
        m_age = "Elder"
    End If
End Property
```

```
Public Property Get Nationality() As String
    Nationality = "Costa Rican"
End Property
```

C# feature turned off:

```
private string m_name = "";
private string m_age = "";

2 references
public string Name
{
    get
    {
        return m_name;
    }
    set
    {
        m_name = value;
    }
}

2 references
public string Age
{
    get
    {
        return m_age;
    }
    set
    {
        int newValue2 = Convert.ToInt32(Double.Parse(value));
        if (newValue2 > 0 && newValue2 < 12)
        {
            m_age = "Enfant";
        }
        else if (newValue2 >= 12 && newValue2 < 18)
        {
            m_age = "Teenager";
        }
        else if (newValue2 >= 18 && newValue2 < 60)
        {
            m_age = "Adult";
        }
        else
        {
            m_age = "Elder";
        }
    }
}

1 reference
public string Nationality
{
    get
    {
        return "Costa Rican";
    }
}
```

C# feature turned on:

```
private string m_name = "";
private string m_age = "";

2 references
public string Name
{
    get => m_name;
    set => m_name = value;
}

2 references
public string Age
{
    get => m_age;
    set
    {
        int newValue2 = Convert.ToInt32(Double.Parse(value));
        if (newValue2 > 0 && newValue2 < 12)
        {
            m_age = "Enfant";
        }
        else if (newValue2 >= 12 && newValue2 < 18)
        {
            m_age = "Teenager";
        }
        else if (newValue2 >= 18 && newValue2 < 60)
        {
            m_age = "Adult";
        }
        else
        {
            m_age = "Elder";
        }
    }
}

1 reference
public string Nationality => "Costa Rican";
```

Support for new libraries and components

Support for DataEnvironment files

Migration for DataEnvironment files is now supported, allowing to generate functional code when using this type of files in the VB6 projects.

Previous C# Code:

```
internal class DataEnvironment_deBank
    : Microsoft.VisualBasic.Compatibility.VB6.BaseDataEnvironment
{

    public System.Data.Common.DbConnection cnBank = null;
    public UpgradeHelpers.DB.ADO.ADORecordSetHelper rsCommand2 = null;
    private System.Data.Common.DbCommand m_Command2 = default(System.Data.Common.DbCommand);
    public UpgradeHelpers.DB.ADO.ADORecordSetHelper rsCommand3 = null;
    private System.Data.Common.DbCommand m_Command3 = default(System.Data.Common.DbCommand);
    1 reference
    public DataEnvironment_deBank()
        : base()
    {
        System.Data.Common.DbParameter par = default(System.Data.Common.DbParameter);


        cnBank = UpgradeHelpers.DB.AdoFactoryManager.GetFactory().CreateConnection();
        cnBank.ConnectionString = "PROVIDER=MSDataShape;DATA PROVIDER=MICROSOFT.JET.OLEDB.4.0;Data Source
        m_Connections.Add(cnBank, "cnBank", 0, 0);
        m_Command2 = UpgradeHelpers.DB.AdoFactoryManager.GetFactory().CreateCommand();
        rsCommand2 = new UpgradeHelpers.DB.ADO.ADORecordSetHelper("");
        m_Command2.Name = "Command2";
        m_Command2.CommandText = "select * from tbltransaction where Date=addate";
        m_Command2.CommandType = CommandType.Text;
        rsCommand2.CursorLocation = UpgradeHelpers.DB.ADO.CursorLocationEnum.adUseClient;
        rsCommand2.CursorType = UpgradeHelpers.DB.ADO.CursorTypeEnum.adOpenStatic;
        rsCommand2.LockType = UpgradeHelpers.DB.LockTypeEnum.LockReadOnly;
        rsCommand2.Source = m_Command2;
        m_Commands.Add(m_Command2, "Command2", null, null);
        m_Recordsets.Add(rsCommand2, "Command2", null, null);
    }
    1 reference
    public void Command2()
    {
        if (cnBank.State == ((int) System.Data.ConnectionState.Closed))
        {
            cnBank.Open(cnBank.ConnectionString, System.String.Empty, System.String.Empty, -1);
        }
        if (rsCommand2.State == ((int) System.Data.ConnectionState.Open))
        {
            rsCommand2.Close();
        }
        m_Command2.ActiveConnection = cnBank;
    }
```

Current C# Code:

```
public DataEnvironment_deBank()
    : base()
{
    System.Data.Common.DbParameter par = null;


    cnBank = UpgradeHelpers.DB.AdoFactoryManager.GetFactory().CreateConnection();
    cnBank.ConnectionString = "PROVIDER=MSDataShape;DATA PROVIDER=MICROSOFT.JET.OLEDB.4.0;Data Source
    m_Connections.Add("cnBank", cnBank);
    m_Command2 = UpgradeHelpers.DB.AdoFactoryManager.GetFactory().CreateCommand();
    rsCommand2 = new UpgradeHelpers.DB.ADO.ADORecordSetHelper();
    m_Command2.setName("Command2");
    par = m_Command2.CreateParameter();
    par.ParameterName = "addate";
    par.DbType = System.Data.DbType.String;
    par.Size = 510;
    par.Direction = System.Data.ParameterDirection.Input;
    m_Command2.Parameters.Add(par);
    rsCommand2.CursorLocation = UpgradeHelpers.DB.ADO.CursorLocationEnum.adUseClient;
    rsCommand2.CursorType = UpgradeHelpers.DB.ADO.CursorTypeEnum.adOpenStatic;
    rsCommand2.LockType = UpgradeHelpers.DB.LockTypeEnum.LockReadOnly;
    m_Command2.CommandText = "select * from tbltransaction where Date=addate";
    m_Command2.CommandType = System.Data.CommandType.Text;
    rsCommand2.Source = m_Command2;
    m_Commands.Add("Command2", m_Command2);
    m_Recordsets.Add("Command2", rsCommand2);
}
1 reference
public void Command2(object addate) => Command2_Typed(System.Convert.ToString(addate));

1 reference
private void Command2_Typed(string addate)
{
    if (cnBank.State == System.Data.ConnectionState.Closed)
    {
        cnBank.Open();
    }
    if (rsCommand2.State == System.Data.ConnectionState.Open)
    {
        rsCommand2.Close();
    }
    m_Command2.Connection = cnBank;
    m_Command2.Parameters["addate"].Value = addate;
    rsCommand2.Open();
}
```

Support for MSMQ

Microsoft Message Queue library support was added, you can now choose if you want to generate MSMQ as an Interop or using native .NET classes when upgrading this class.

Note: Option to generate native .NET elements is not available when selecting .NET 6 or higher as target.

VB6 Code:

```
Dim qInfo As MSMQ.MSMQQueueInfo
Dim q As MSMQ.MSMQQueue
Dim msg As MSMQ.MSMQMessage
Dim DefaultTimeToRec As Long


Private Sub Command1_Click()
    DefaultTimeToRec = 259200

' Initialize the queue information
    Set qInfo = New MSMQ.MSMQQueueInfo
    qInfo.PathName = ".\Private$\TestQueueName" ' Replace with your queue name

    ' Create the queue
    Set q = qInfo.Open(MQ_SEND_ACCESS, MQ_DENY_NONE)

    ' Create a message
    Set msg = New MSMQ.MSMQMessage
    msg.Body = "Hello, MSMQ simple without properties!"

    ' Send the message
     msg.Send q
     MsgBox "MSMQ simple without properties was sent"
```

Using Interop:

```
MSMQ.MSMQQueueInfo qInfo = null;
MSMQ.MSMQQueue q = null;
MSMQ.MSMQMessage msg = null;
int DefaultTimeToRec = 0;


1 reference
private void Command1_Click(Object eventSender, EventArgs eventArgs)
{
    int DELIVERY_PRIORITY = 0;
    int SECURE_DELIVERY = 0;
    DefaultTimeToRec = 259200;

    // Initialize the queue information
    qInfo = new MSMQ.MSMQQueueInfo();
    qInfo.PathName = ".\\Private$\\TestQueueName"; // Replace with your queue name

    // Create the queue
    q = (MSMQ.MSMQQueue) qInfo.Open((int) MSMQ.MQACCESS.MQ_SEND_ACCESS, (int) MSMQ.MQSHARE.MQ_DENY_NONE);

    // Create a message
    msg = new MSMQ.MSMQMessage();
    msg.Body = "Hello, MSMQ simple without properties!";

    // Send the message
    object tempRefParam = Type.Missing;
    msg.Send(q, ref tempRefParam);
    MessageBox.Show("MSMQ simple without properties was sent", AssemblyHelper.GetTitle(System.Reflection.Assembly.GetExecutingAssembly()));
```

Using Native .NET Classes:

```
System.Messaging.MessageQueue qInfo = null;
System.Messaging.MessageQueue q = null;
System.Messaging.Message msg = null;
int DefaultTimeToRec = 0;


1 reference
private void Command1_Click(Object eventSender, EventArgs eventArgs)
{
    System.Messaging.MessagePriority DELIVERY_PRIORITY = System.Messaging.MessagePriority.Lowest;
    int SECURE_DELIVERY = 0;
    DefaultTimeToRec = 259200;

    // Initialize the queue information
    qInfo = new System.Messaging.MessageQueue();
    qInfo.Path = ".\\Private$\\TestQueueName"; // Replace with your queue name

    // Create the queue
    //UPGRADE_WARNING: (2065) MSMQ.MSMQQueueInfo method qInfo.Open has a new behavior. More Information: https://docs.mobilize.net/vbuc/ewis#2065
    q = (System.Messaging.MessageQueue) (qInfo = new System.Messaging.MessageQueue(qInfo.Path, false, false, System.Messaging.QueueAccessMode.Send));

    // Create a message
    msg = new System.Messaging.Message();
    msg.Body = "Hello, MSMQ simple without properties!";

    // Send the message
    //UPGRADE_WARNING: (2065) MSMQ.MSMQMessage method msg.Send has a new behavior. More Information: https://docs.mobilize.net/vbuc/ewis#2065
    q.Send(msg);
    MessageBox.Show("MSMQ simple without properties was sent", AssemblyHelper.GetTitle(System.Reflection.Assembly.GetExecutingAssembly()));
```

Support for VBScript Regex

Microsoft VBScript Regular Expressions support was added, allowing to generate .NET native code using the System.Text.RegularExpressions class or using the Interop wrapper.

VB6 Code:

```
' Create a RegExp object
    Dim regex As RegExp
    Dim text As String
    Dim foundPattern As Boolean
    Dim foundUppercase As Boolean
    Dim foundLowercase As Boolean
    Dim foundNumerics As Boolean
    Dim foundSpecialCharacters As Boolean
    Dim outputString As String
    Dim matches As MatchCollection

    Set regex = New RegExp
    text = "Hello, world! Hello, there. 123 @"
    TextExp.text = text

    ' The string you want to search within
    Dim inputString As String
    inputString = Text1.text

    ' Set the pattern you want to match
    regex.Pattern = inputString

    ' Perform a global search (find all matches)
    regex.Global = True
    ' Clear the ListBox
    List1.Clear

    ' Execute the regular expression search
    Set matches = regex.Execute(text)

    If (matches.Count > 0) Then
        ' Loop through the matches and add them to the ListBox change by me
        Dim match As match
        For Each match In matches
            List1.AddItem "Match found at position " & match.FirstIndex & ": " & match.Value
        Next
    End If

    'it founds the pattern given ?
    foundPattern = regex.Test(text)
    If foundPattern Then
      MsgBox "Pattern : " + inputString + " matches the Expression. ", vbInformation, "Pattern Match"
    End If
```

Using Interop:

```
// Create a RegExp object

VBScript_RegExp_55.RegExp regex = new VBScript_RegExp_55.RegExp();
string text = "Hello, world! Hello, there. 123 @";
TextExp.Text = text;

// The string you want to search within
string inputString = Text1.Text;

// Set the pattern you want to match
regex.Pattern = inputString;

// Perform a global search (find all matches)
regex.Global = true;
// Clear the ListBox
List1.Items.Clear();

// Execute the regular expression search
VBScript_RegExp_55.MatchCollection matches = (VBScript_RegExp_55.MatchCollection) regex.Execute(text);

VBScript_RegExp_55.Match match = null;
if (matches.Count > 0)
{
    // Loop through the matches and add them to the ListBox change by me
    foreach (VBScript_RegExp_55.Match matchIterator in matches)
    {
        match = matchIterator;
        List1.AddItem($"Match found at position {match.FirstIndex.ToString()}: {match.Value}");
        match = default(VBScript_RegExp_55.Match);
    }
}

//it founds the pattern given ?
bool foundPattern = regex.Test(text);
if (foundPattern)
{
    MessageBox.Show($"Pattern : {inputString} matches the Expression. ", "Pattern Match", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

Using Native .NET with Helpers classes:

```
// Create a RegExp object

ScriptRegexHelper regex = new ScriptRegexHelper();
string text = "Hello, world! Hello, there. 123 @";
TextExp.Text = text;

// The string you want to search within
string inputString = Text1.Text;

// Set the pattern you want to match
regex.Pattern = inputString;

// Perform a global search (find all matches)
//UPGRADE_ISSUE: (2064) VBScript_RegExp_55.RegExp property regex.Global was not upgraded.
regex.setGlobal(true);
// Clear the ListBox
List1.Items.Clear();

// Execute the regular expression search
MatchCollection matches = (MatchCollection) regex.Execute(text);

Match match = null;
if (matches.Count > 0)
{
    // Loop through the matches and add them to the ListBox change by me
    foreach (Match matchIterator in matches)
    {
        match = matchIterator;
        List1.AddItem($"Match found at position {match.Index.ToString()}: {match.Value}");
        match = default(Match);
    }
}

//it founds the pattern given ?
bool foundPattern = regex.Test(text);
if (foundPattern)
{
    MessageBox.Show($"Pattern : {inputString} matches the Expression. ", "Pattern Match", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

Other Improvements include but are not limited to:

- Databases mappings
- Maps for VSIndex
- Upgrade Options reorganization.
- NEW GUI Colors
- Support for Visual Studio 2010, 2012, 2013, 2015 was removed.