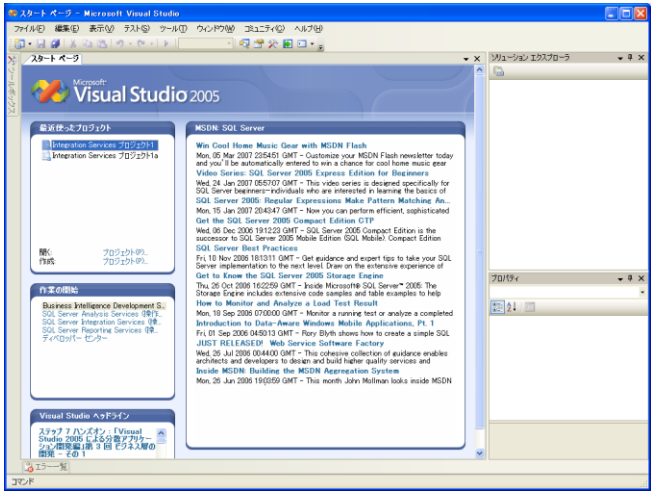
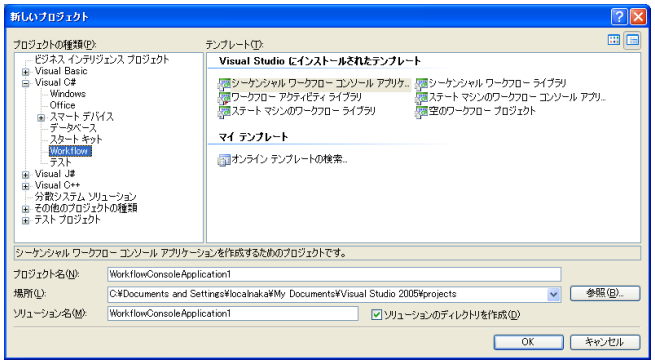
DEMO1

まずはやってみよう



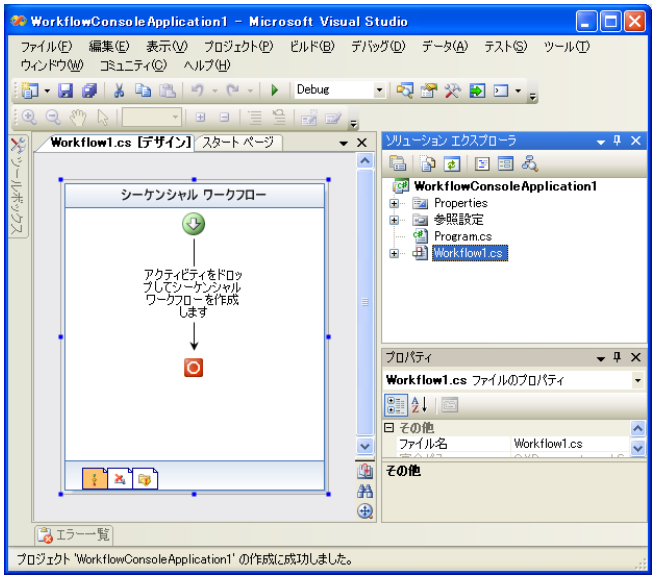
作成-プロジェクト

C# => Workflow

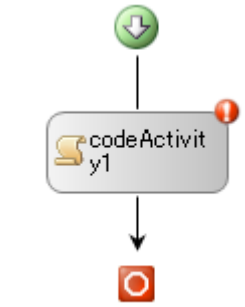


シーケンシャルと、ステートマシン、それぞれのコンソールアプリ、あとライブラリがある。

今回はシーケンシャルのコンソール

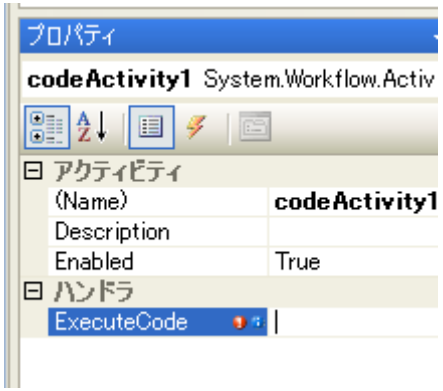


CodeActivity をぽとぺ



びっくりマークは足りていないあかし

プロパティをみると判別できます。



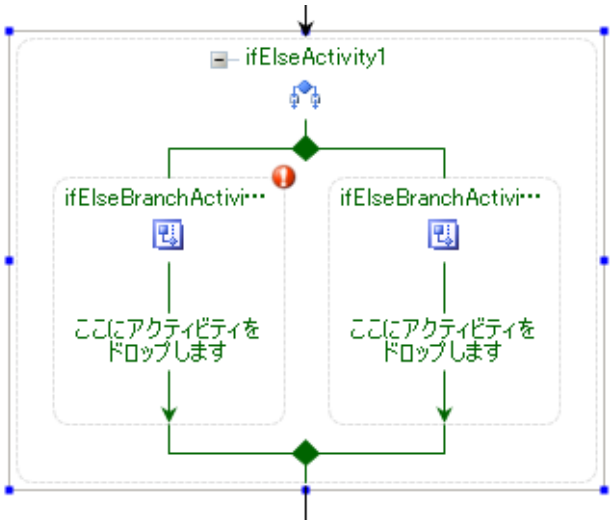
アクティビティをダブルクリック

```
namespace WorkflowConsoleApplication1
{
    public sealed partial class Workflow1: SequentialWorkflowActivity
    {
        public Workflow1()
        {
            InitializeComponent();
        }

        private void codeActivity1_ExecuteCode(object sender, EventArgs
        {

        }
    }

}
```

こんなコードを追加

```
string str = Console.ReadLine();
int コード= int.Parse(str);
this.UserData["Key"] =コード;
this.UserData["偶数フラグ"] = (コード% 2) == 0 ;
```

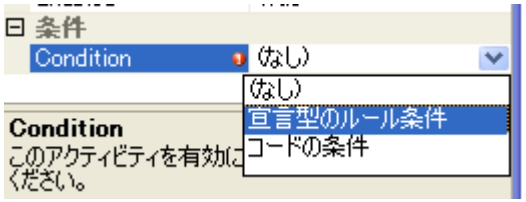
If-Else アクティビティをぽとぺ



左から順に評価されていきます。

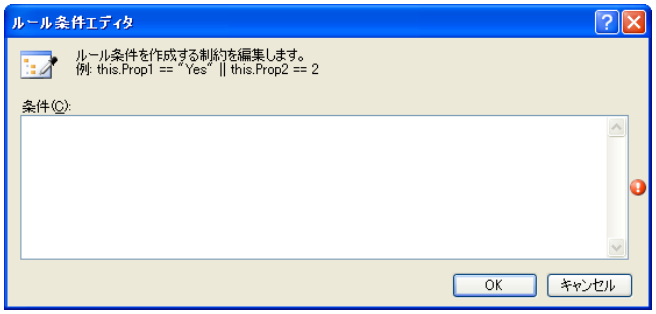
左のブランチの条件を設定します。

宣言型



ConditionName　に適当に設定

Expression　…



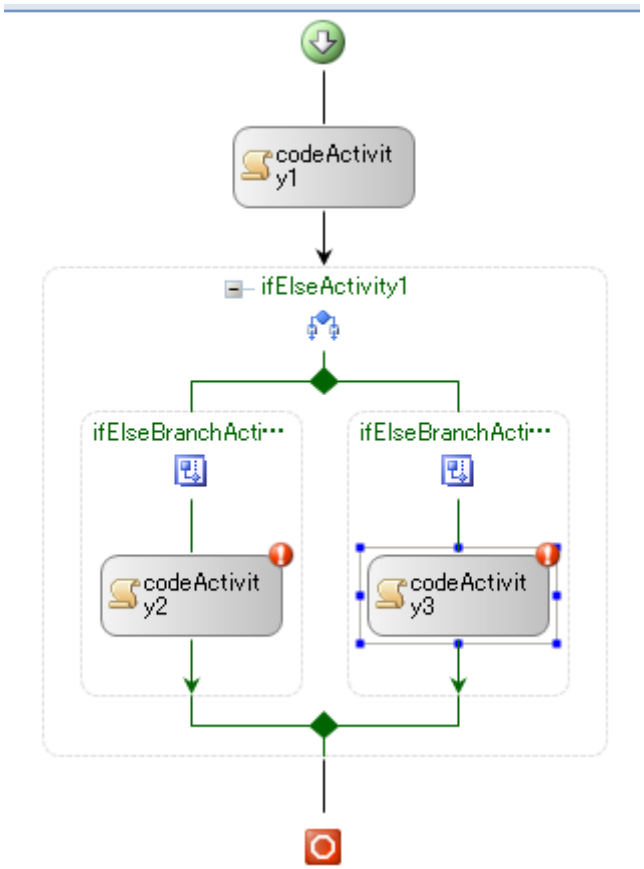
(bool)this.UserData["偶数フラグ"] == true

という条件をかく

これ Workflow1.rules に記述される。

```
naryOperatorExpression.Left)
deCastExpression TargetType="System.Boolean, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
0:CodeCastExpression.Expression)
 <ns0:CodeIndexerExpression)
     <ns0:CodeIndexerExpression.Indices)
         <ns0:CodePrimitiveExpression)
             <ns0:CodePrimitiveExpression.Value)
                 <ns1:String xmlns:ns1="clr-namespace:System;Assembly=mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">偶数フラグ</ns
             </ns0:CodePrimitiveExpression.Value)
         </ns0:CodePrimitiveExpression)
     </ns0:CodeIndexerExpression.Indices)
     <ns0:CodeIndexerExpression.TargetObject)
         <ns0:CodePropertyReferenceExpression PropertyName="UserData">
             <ns0:CodePropertyReferenceExpression.TargetObject)
                 <ns0:CodeThisReferenceExpression />
             </ns0:CodePropertyReferenceExpression.TargetObject)
         </ns0:CodePropertyReferenceExpression)
     </ns0:CodeIndexerExpression.TargetObject)
 </ns0:CodeIndexerExpression)
s0:CodeCastExpression.Expression)
odeCastExpression)
inaryOperatorExpression.Left)
naryOperatorExpression.Right)
dePrimitiveExpression)
0:CodePrimitiveExpression.Value)
 <ns1:Boolean xmlns:ns1="clr-namespace:System;Assembly=mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">true</ns1:Boolean>
s0:CodePrimitiveExpression.Value)
```

そして左右のブランチに CodeActivity を配置する




```
private void codeActivity2_ExecuteCode(object
sender, EventArgs e)
{

Console.WriteLine(this.UserData["Key"].ToString(
) + "だよ～～ん");
}

private void codeActivity3_ExecuteCode(object
sender, EventArgs e)
```

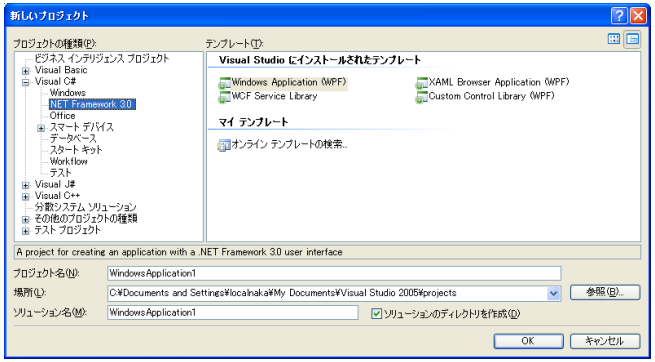
```
{

Console.WriteLine(this.UserData["Key"].ToString(
) + "ですよ！！");

}
```

一瞬で消えちゃうので、下に CodeActivity で、ReadLine をつける。
実行

奇数
偶数
試しましょう。

DEMO2

プロジェクト作成



追加で WF が選べないことを確認

いったんスライドに戻る

いったん終了

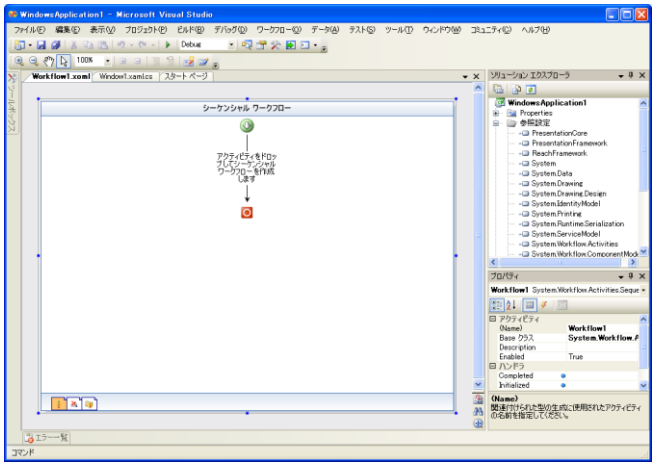
Csproj を開く

<ProjectTypeGuids>{14822709-B5A1-4724-98CA-57A101D1B079};

```
    <Reference
Include="System.Workflow.Activities" />
    <Reference
Include="System.Workflow.ComponentModel" />
    <Reference
Include="System.Workflow.Runtime" />

  <Import
Project="$(MSBuildExtensionsPath)¥Microsoft¥
Windows                          Workflow
Foundation¥v3.0¥Workflow.Targets" />
```

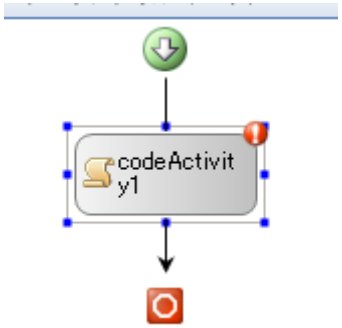
シーケンシャルワークフローを追加します。

ここでかならず(コード)を選ぶ。XOML だとまともに動かない。なぜかは今のところわかりません。



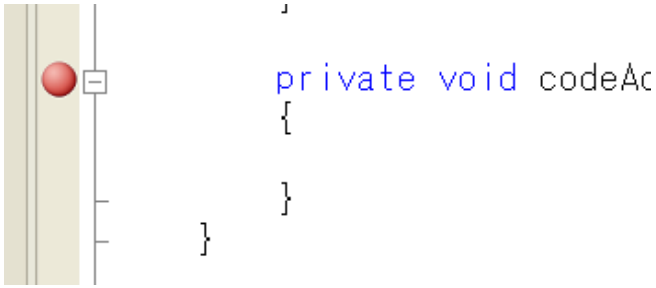
ワークフローの追加ができました。

コードアクティビティを追加します。



ダブルクリック

デバッグ用なので、ブレークポイントを置く



```
        }
    }
```

こんな XAML を追加する。

```
   FontSize="50">
    <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="*"/>
                    <RowDefinition Height="*"/>
                    <RowDefinition Height="*"/>
                </Grid.RowDefinitions>
                <Button Content="ワークフロー開始"
Click="開始" Grid.Row="1"/>
    </Grid>
</Window>
```

Cs にイベントを追加する。

```
        public void 開始(object sender,
RoutedEventArgs args)
        {
        }
```

Using は 5 個くらい

```
using System.Workflow;
using System.Workflow.Activities;
using System.Workflow.ComponentModel;
using System.Workflow.Runtime;
using System.Workflow.Runtime.Hosting;
```

ワークフローランタイムをメンバに追加

```
        WorkflowRuntime _wr = new
WorkflowRuntime();
```

コンストラクタで、ランタイム開始

```
            this._wr.StartRuntime();
```

開始にワークフローを開始するロジックを追加

```
        public void 開始(object sender,
RoutedEventArgs args)
        {
            WorkflowInstance wi =
this._wr.CreateWorkflow(typeof(Workflow1));
            wi.Start();
        }
```

テスト

ブレーク OK

DEMO3

画面の xaml をテキストボックスと、テキストブロックを追加した形に変更

```
<TextBox Text="{Binding Path=src}"/>
<Button Content="ワークフロー開始" Click="開始"
Grid.Row="1"/>
<TextBlock Text="{Binding Path=dest}" Grid.Row="2"/>
```

やってはいけないけど、データコンテキストに自身を指定

```
this.DataContext = this;
```

Using する

```
using System.ComponentModel;
```

インターフェイス実装する

```
, InotifyPropertyChanged
```

ファイアを追加する

```
        public event PropertyChangedEventHandler
PropertyChanged;
        private void FirePropertyChanged(string
propertyname)
        {
            if (this.PropertyChanged != null)
{ this.PropertyChanged(this, new
PropertyChangedEventArgs(propertyname)); }
        }
```

プロパティで src, dest を追加する

```
        private int _src;

        public int src
        {
            get { return _src; }
            set { _src = value;
this.FirePropertyChanged("src"); }
        }
        private int _dest;

        public int dest
        {
            get { return _dest; }
            set { _dest = value;
this.FirePropertyChanged("dest"); }
        }
```

パラメータ渡しするように変更

```
       public void 開始(object sender, RoutedEventArgs
args)
        {
            Dictionary<string, object> dic = new
Dictionary<string, object>();
            dic.Add("src", this.src);
            WorkflowInstance wi =
this._wr.CreateWorkflow(typeof(Workflow1), dic);
            wi.Start();
        }
```

Workflow側にもsrcを追加

```
        private int _src;

        public int src
        {
            get { return _src; }
            set { _src = value; }
        }
```

ここまででテスト

Destプロパティを追加

```
        private int _dest;

        public int dest
        {
            get { return _dest; }
            set { _dest = value; }
        }
```

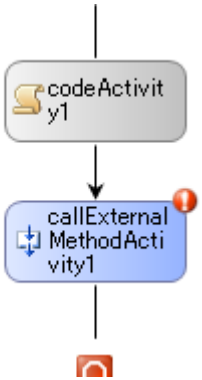
実装2倍にするだけ

```
        private void codeActivity1_ExecuteCode(object sender, EventArgs e)
        {
            this.dest = this.src * 2;
        }
```

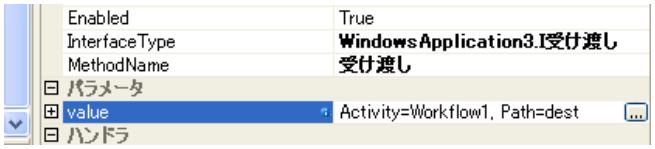
受け渡しインターフェイスを作成

```
    [ExternalDataExchange]
    public interface I受け渡し
    {
        void 受け渡し(int value);
    }
```

CallExternalMethod をぽとぺ



さっきのメソッドを指定する



受け渡し実装クラスの作成

```
    public class 受け渡しクラス : I受け渡し
    {
        #region I受け渡しメンバ

        public void 受け渡し(int value)
        {
            if (this.受け渡しデリゲート!= null)
            {
                this.受け渡しデリゲート(value);
            }
        }

        public Action<int> 受け渡しデリゲート;
        #endregion
    }
```

受け渡し実装クラスの登録

コンストラクタに追加

```
            ExternalDataExchangeService exservice = new ExternalDataExchangeService();
            this._wr.AddService(exservice);
            受け渡しクラス uke = new 受け渡しクラス();
            uke.受け渡しデリゲート= delegate(int value)
            {
                this.dest = value;
            };
            exservice.AddService(uke);
```

DEMO4

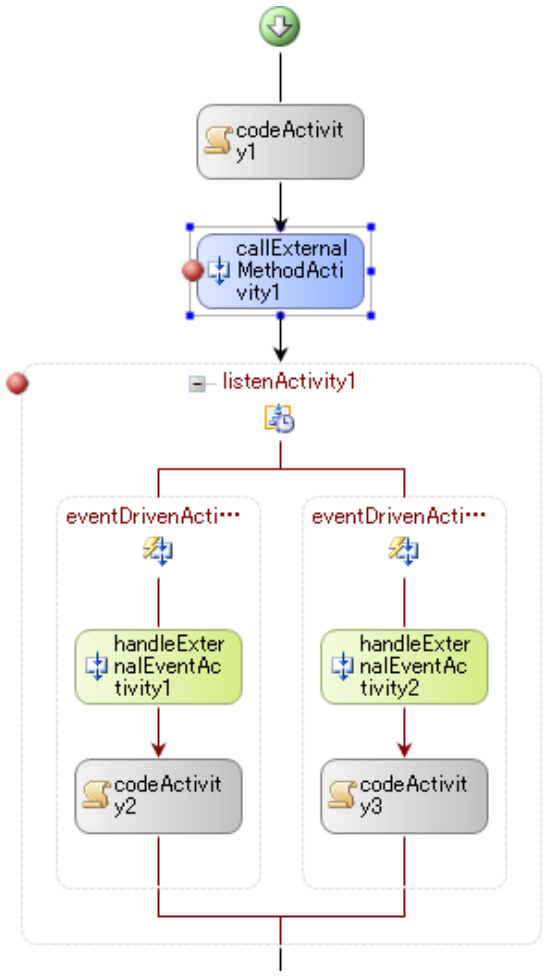
I 受け渡しを以下のように変更する。

```
[ExternalDataExchange]
public interface I受け渡し
{
    void 受け渡し(Guid guid, int value);
    event EventHandler<ExternalDataEventArgs> 許可;
    event EventHandler<ExternalDataEventArgs> 不許可;

}
```

Guid を持たせるのは、対話するためのインスタンス番号を戻さなくてはいけないため。

ワークフロー側にインスタンス GUID を取得するプロパティを追加する。

```
public Guid guid
{
    get
    {
        return WorkflowEnvironment.WorkflowInstanceId;
    }
}
```



左は許可、右は不許可

左のコードアクティビティは 4 倍、右は 1 倍とする。

受け渡しクラスは外に出す。

```
受け渡しクラス uke = new 受け渡しクラス();
```

受け渡しクラスは Guid の追加と、許可不許可およびそれぞれの Fire を追加する。

```
public class 受け渡しクラス : I受け渡し
{
    #region I受け渡しメンバ

    public void 受け渡し(Guid guid, int value)
    {
        if (this.受け渡しデリゲート!= null)
        {
            this.受け渡しデリゲート(guid, value);
        }
    }
    public delegate void 受け渡しdelegate(Guid guid, int value);

    public 受け渡しdelegate 受け渡しデリゲート;



    public event EventHandler<ExternalDataEventArgs> 許可;
    public void Fire許可(Guid guid)
    {
        if (this.許可!= null)
        {
            this.許可(null, new ExternalDataEventArgs(guid));
        }
    }
```

```
        public event
EventHandler<ExternalDataEventArgs> 不許可;
        public void Fire不許可(Guid guid)
        {
            if (this.不許可!= null)
            {
                this.不許可(null, new
ExternalDataEventArgs(guid));
            }
        }

        #endregion
    }
```

画面を編集する。

4行目を追加して、許可不許可とする。

```
                    <Grid Grid.Row="3">
                            <Grid.ColumnDefinitions>

         <ColumnDefinition Width="0.5*"/>

         <ColumnDefinition Width="0.5*"/>
                            </Grid.ColumnDefinitions>
                            <Button Grid.Column="0"
Content="許可" Click="許可"/>
                            <Button Grid.Column="1"
Content="不許可" Click="不許可"/>
                    </Grid>
```

画面のイベントを追加する。

```
        public void 許可(object sender, RoutedEventArgs
args)
        {
                uke.Fire許可(this.guid);
        }
        public void 不許可(object sender,
RoutedEventArgs args)
        {
                uke.Fire不許可(this.guid);
        }
```