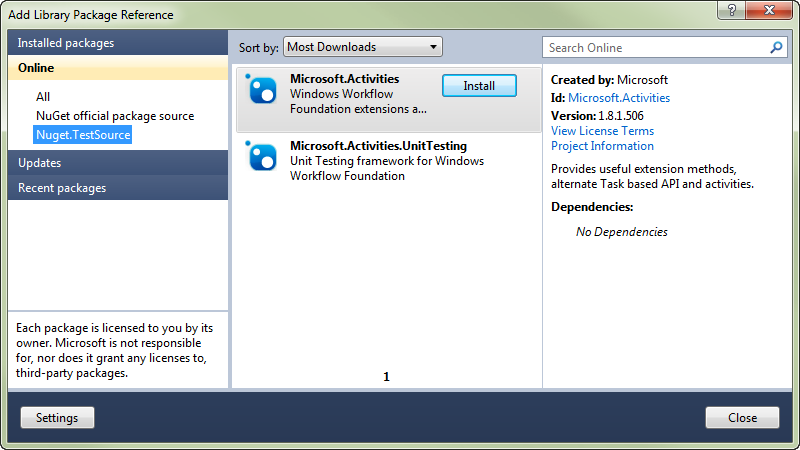
Microsoft.Activities Version 1.8.0

* 1. Microsoft.Activities.dll is a library of helpers, extension methods and activities that you can add to the .NET Framework library System.Activities when developing WF4 Workflows. This labs release is provided in both binary and source form on the WF4 CodePlex site <http://wf.codeplex.com> as example code for you to use in your projects.

## Using Microsoft.Activities.dll in your project (NuGet)

* 1. Install the NuGet Package Manager <http://nuget.codeplex.com>
  2. Right click on your project and select Add Library Package Reference…
  3. Search for Microsoft.Activities
  4. Click Install to add it to your project

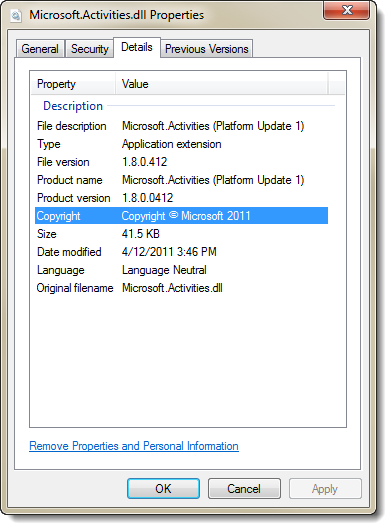


## Using Microsoft.Activities.dll in your project (CodePlex)

1. Download the binaries for the platform you are targeting. You can target .NET Framework 4 or .NET Framework 4 Platform Update 1.
2. Place the binaries into a folder that can be referenced by your project
3. Add a reference and browse to the location with the assembly
4. Add the necessary using or import statements to include types from the Microsoft.Activities namespace. Extension methods will not appear as an option in intellisense until you do this.

## How to tell which version of binaries you have

If you aren't sure which platform the assembly is targeting, just right click on the file in Explorer and select Properties. Then click on the details tab. Assemblies built for Platform Update 1 will contain the text (Platform Update 1) in the file description.



## Workflow Application Extensions

Microsoft.Activities includes a set of extension methods for WorkflowApplication which implement a task based model for running a workflow. Each time you run or resume the workflow is a Workflow Episode. When the episode completes it returns a result object that you can examine to determine what happened in the workflow episode. This model allows you to run an episode of work which simply means to run the workflow until

|  |  |  |
| --- | --- | --- |
| Workflow Outcome | ActivityInstanceState | WorkflowEpisodeResult Sub-Type Returned |
| Workflow completed | Closed | WorkflowCompletedEpisodeResult |
| Workflow faulted | Faulted | WorkflowAbortedEpisodeResult |
| Workflow idle | Executing | WorkflowIdleEpisodeResult |
| Timeout | A **TimeoutException** is thrown | |

## What about WorkflowInvoker?

[WorkflowInvoker](http://msdn.microsoft.com/en-us/library/system.activities.workflowinvoker.aspx) allows you to invoke a workflow in a similar way with an important restriction. It provides no method for dealing with a workflow that becomes idle. This makes it suitable only for workflows that run to completion.

Walkthrough – Hello WF Task API

In this walkthrough we will create a simple Workflow Console application that uses the task based API instead of WorkflowInvoker.

Task 1 – Create a new Workflow Application

* 1. Start Visual Studio and create a new Workflow Console Application named **HelloWFTaskAPI**
  2. Right click on the project and select **Add / New Folder**
  3. Name the folder **Reference Assemblies**
  4. Open Workflow1.xaml and drop a **Sequence** activity
  5. Drop a **WriteLine** activity in the Sequence
  6. Set the **Text** property to "Hello WF Task API"

Task 2 - Download and Reference Microsoft.Activities in your Project

* 1. Download the latest [Microsoft.Activities](http://wf.codeplex.com/releases) release
  2. Extract the zip file into the **Reference Assemblies** folder you created earlier
  3. Add a reference to **Microsoft.Activities.dll**
     1. **Where should I put Microsoft.Activities.dll?**
     2. You can put it in the GAC a central location or if you want to keep the binaries with your project, create a folder named "Reference Assemblies" and refer to it there.

Task 3 – Use the WorkflowApplication.RunEpisode method

* 1. Open **program.cs**
  2. Import the **Microsoft.Activities** namespace
     1. C#
     2. using Microsoft.Activities;
  3. Replace the main method with the following code which will run the episode synchronously
     1. C#
     2. static void Main(string[] args)
     3. {
     4. var workflowApplication = new WorkflowApplication(new Workflow1());
     5. var result = workflowApplication.RunEpisode();
     6. }
  4. Run the application.

Task 4 – Use the WorkflowApplication.RunEpisodeAsync method

* + 1. In the previous task you ran the workflow synchronously similar to the way you would with WorkflowInvoker. Now we will use a Task to control execution.
  1. Change the Main method to use RunEpisodeAsync
     1. C#
     2. static void Main(string[] args)
     3. {
     4. var workflowApplication = new WorkflowApplication(new Workflow1());
     5. var workflowTask = workflowApplication.RunEpisodeAsync();
     6. Console.WriteLine("I can do some other work here");
     7. // Get the result of the workflow episode
     8. var result = workflowTask.Result;
     9. if (result.State == ActivityInstanceState.Closed)
     10. {
     11. Console.WriteLine("The workflow completed successfully");
     12. }
     13. }

Microsoft.Activities.Statements

* 1. Microsoft.Activities includes a set of simple activities that you can use in your project as well.

Add the Activities to your Visual Studio Toolbox

* + 1. **Note:** If you install from NuGet (Library Package Reference) your toolbox will be updated automatically
  1. Right click on the toolbox and select Add Tab
  2. Name the new tab **Microsoft.Activities**
  3. Right click on the toolbox again and select Choose Items…
  4. Select the **System.Activities** components tab
  5. Browse to the location where you installed **Microsoft.Activities.dll** andselect it.
  6. Add the components you want to your toolbox

# Microsoft.Activities.Diagnostics

**Microsoft.Activities.ServiceModel**

* SilverlightFaultBehavior
* SilverlightFaultElement

Use these classes with WorkflowServices and Silverlight to get errors passed through to Silverlight client code

<system.serviceModel>

<serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true"/>

<extensions>

<behaviorExtensions>

<add name="silverlightFaultBehavior" type="Microsoft.Activities.ServiceModel.SilverlightFaultElement, Microsoft.Activities"/>

</behaviorExtensions>

</extensions>

<behaviors>

<serviceBehaviors>

<behavior>

<serviceDebug includeExceptionDetailInFaults="true"/>

<serviceMetadata httpGetEnabled="true"/>

<silverlightFaultBehavior/>

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

**Microsoft.Activities.Diagnostics**

* WorkflowServiceTraceBehavior
* WorkflowServiceTraceElement

Use these classes while developing to get workflow tracking output in Visual Studio debug output window.

<system.serviceModel>

<serviceHostingEnvironment aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true"/>

<extensions>

<behaviorExtensions>

<add name="workflowServiceTraceBehavior" type="Microsoft.Activities.Diagnostics.WorkflowServiceTraceElement, Microsoft.Activities"/>

</behaviorExtensions>

</extensions>

<behaviors>

<serviceBehaviors>

<behavior>

<serviceDebug includeExceptionDetailInFaults="true"/>

<serviceMetadata httpGetEnabled="true"/>

<workflowServiceTraceBehavior/>

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

Example tracking output in debug window

Activity <StateMachine> is scheduled child activity <Closed / Unlocked> at 12:26:31.3603

Activity <Closed / Unlocked> state is Executing at 12:26:31.3613

{

Arguments

EventManager: System.Activities.Statements.StateMachineEventManager

}

StateMachineStateRecord <StateMachine> Current State <Closed / Unlocked> at 12:26:31.3613

{

Data

stateMachine: StateMachine

currentstate: Closed / Unlocked

}

# WorkflowArguments

Use the C# **dynamic** keyword to create or access input and output dictionaries from the workflow. Notice how I'm using the names of the arguments like properties. Remember argument names are case sensitive.

public void WorkflowArgumentsCanPassToWorkflowInvoker()

{

// Arrange

var activity = new ArgTest();

dynamic input = new WorkflowArguments();

// Act

input.Num1 = 2;

input.Num2 = 3;

var output = WorkflowArguments.FromDictionary(WorkflowInvoker.Invoke(activity, input));

// Assert

Assert.AreEqual(5, output.Result);

}