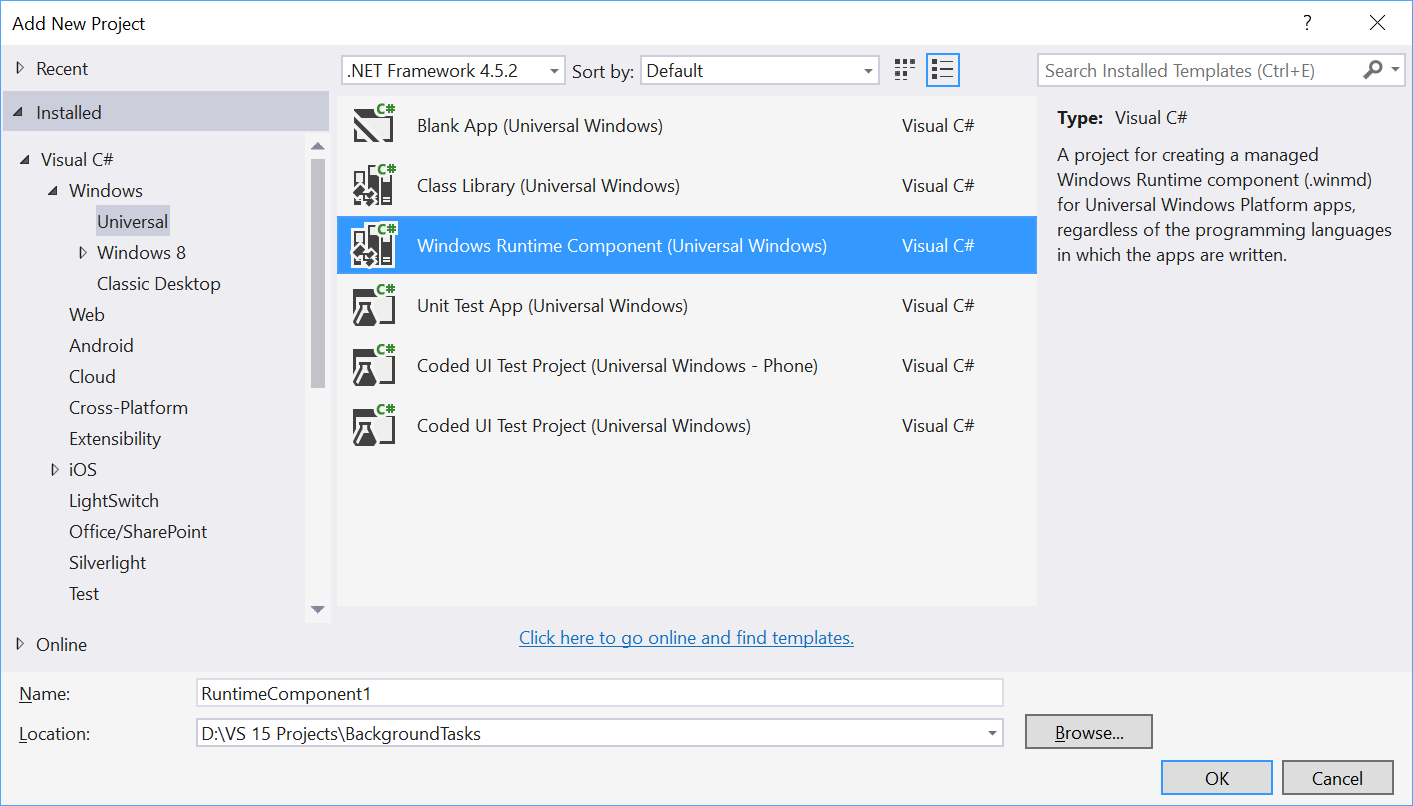
Background Task Execution

Going to start with an out of process background task. This requires a runtime component to be created and registered to do the work in a similar way to a service.

Create a UWP project. When the project creation is completed, add a RuntimeComponent project to the solution.



Your solution should now contain two projects, one for the UI and one for the RuntimeComponent where the background task will be implemented. You will need to add the background task project as a reference to the UI so that it can access it and register it when running.

The background task will have a Run method which is the entry point for the task. Code should look something like this…

namespace myRuntimeComponent

{

public sealed class myBackgroundClass : IBackgroundTask

{

BackgroundTaskDeferral \_deferral= null;

public async void Run(

IBackgroundTaskInstance taskInstance)

{

The background task must implement the IBackgroundTask interface. If you run any asynchronous code in your background task, then your background task needs to use a deferral. If you don't use a deferral, then the background task process can terminate unexpectedly if the Run method completes before your asynchronous method call has completed. Don’t forget the Deferral.Complete statement to ensure proper management of resources by the system.

Find out if the background task is already registered by iterating through the [**BackgroundTaskRegistration.AllTasks**](https://msdn.microsoft.com/library/windows/apps/br224787) property. This step is important; if your app doesn't check for existing background task registrations, it could easily register the task multiple times, causing issues with performance and maxing out the task's available CPU time before work can complete.

foreach(var task in BackgroundTaskRegistration.AllTasks)

{

if( task.Value.Name == BackgroundTaskName)

{

taskRegistered = true;

break;

}

}

If the background task is not already registered, use [**BackgroundTaskBuilder**](https://msdn.microsoft.com/library/windows/apps/br224768) to create an instance of your background task. The task entry point should be the name of your background task class prefixed by the namespace. Register the background task by calling the Register method on the [**BackgroundTaskBuilder**](https://msdn.microsoft.com/library/windows/apps/br224768)object. Store the [**BackgroundTaskRegistration**](https://msdn.microsoft.com/library/windows/apps/br224786) result so it can be used in the next step.

if( taskRegistered != true)

{

BackgroundAccessStatus backgroundAccessStatus =

await BackgroundExecutionManager.RequestAccessAsync();

var builder = new BackgroundTaskBuilder();

builder.Name = BackgroundTaskName;

builder.TaskEntryPoint = "myBackgroundClass.Run";

builder.SetTrigger(new TimeTrigger(15, false));

BackgroundTaskRegistration task = builder.Register();

task.Completed += new

BackgroundTaskCompletedEventHandler(OnCompleted);

To ensure that your Universal Windows app continues to run properly after you release an update, use the **ServicingComplete** (see [SystemTriggerType](https://msdn.microsoft.com/library/windows/apps/br224839)) trigger to perform any post-update configuration changes such as migrating the app's database and registering background tasks. It is best practice to unregister background tasks associated with the previous version of the app (see [**RemoveAccess**](https://msdn.microsoft.com/library/windows/apps/hh700471)) and register background tasks for the new version of the app (see [**RequestAccessAsync**](https://msdn.microsoft.com/library/windows/apps/hh700485)) at this time.

为确保您的通用Windows应用程序在发布更新后继续正常运行，使用ServicingComplete（请参阅SystemTriggerType）触发器来执行任何更新后配置更改，例如迁移应用程序的数据库和注册后台任务。最好的做法是注销与以前版本的应用程序相关联的后台任务（请参阅RemoveAccess）并注册新版本的应用程序的后台任务（请参阅RequestAccessAsync）。

You should register a method with the [**BackgroundTaskCompletedEventHandler**](https://msdn.microsoft.com/library/windows/apps/br224781), so that your app can get results from the background task. When the app is launched or resumed, the mark method will be called if the background task has completed since the last time the app was in the foreground. (The OnCompleted method will be called immediately if the background task completes while your app is currently in the foreground.)

private async void OnCompleted(IBackgroundTaskRegistration task, BackgroundTaskCompletedEventArgs e)

{

Before your app can run background tasks, you must declare each background task in the app manifest. If your app attempts to register a background task with a trigger that isn't listed in the manifest, the registration will fail.

在您的应用程序可以运行后台任务之前，您必须在应用程序清单(app manifest)中声明每个后台任务。如果您的应用尝试使用清单中未列出的触发器注册后台任务，则注册将失败。

The purpose of the background task is to read the calendars that are available on the user’s account and provide information about these through the UI. Initially, write these to a setting for the application (or a file if you prefer) and the UI will read them and display them.

后台任务的目的是阅读用户帐户上可用的日历，并通过UI提供有关这些日历的信息。

最初，将这些写入应用程序的设置（如果您愿意的话），UI将会读取并显示它们。

It is possible to access the calendars and retrieve the entries. See this link for details

<https://docs.microsoft.com/en-us/windows/uwp/contacts-and-calendar/>

Brief Tutorial on creating background tasks with toast notification.

Create a background task class and register it to run when your app is not in the foreground. You can run code in the background by writing classes that implement the **IBackgroundTask**interface. The following sample code shows a very basic starting point for a background task class.

public sealed class MyBackgroundTask : IBackgroundTask {

public void Run(IBackgroundTaskInstance taskInstance){

// write code

}

}

You can request access for background task as follows.

var access = await BackgroundExecutionManager.RequestAccessAsync();

switch (access) {

case BackgroundAccessStatus.Unspecified:

break;

case BackgroundAccessStatus.AllowedMayUseActiveRealTimeConnectivity:

break;

case BackgroundAccessStatus.AllowedWithAlwaysOnRealTimeConnectivity:

break;

case BackgroundAccessStatus.Denied:

break;

default:

break;

}

To build and register the background task, use the following code.

var task = new BackgroundTaskBuilder {

Name = "My Task",

TaskEntryPoint = typeof(BackgroundStuff.MyBackgroundTask).ToString()

};

var trigger = new ApplicationTrigger();

task.SetTrigger(trigger);

task.Register();

await trigger.RequestAsync();

Let us understand a simple example of background task by following all the below given steps.

* Create a new blank UWP project **‘UWPBackgroundDemo’** and add one button in the XAML file.

<Page

x:Class = "UWPBackgroundDemo.MainPage"

xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local = "using:UWPBackgroundDemo"

xmlns:d = "http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc = "http://schemas.openxmlformats.org/markup-compatibility/2006"

mc:Ignorable = "d">

<Grid Background = "{ThemeResource ApplicationPageBackgroundThemeBrush}">

<Button x:Name = "button" Content = "Button"

HorizontalAlignment = "Left" Margin = "159,288,0,0"

VerticalAlignment = "Top" Click = "button\_Click"/>

</Grid>

</Page>

* Given below is the **button click** event implementation in which the background task is registered.

using System;

using Windows.ApplicationModel.Background;

using Windows.UI.Xaml;

using Windows.UI.Xaml.Controls;

// The Blank Page item template is documented at

http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

namespace UWPBackgroundDemo {

/// <summary>

/// An empty page that can be used on its own or navigated to within a Frame.

/// </summary>

public sealed partial class MainPage : Page {

public MainPage() {

this.InitializeComponent();

}

private async void button\_Click(object sender, RoutedEventArgs e) {

var access = await BackgroundExecutionManager.RequestAccessAsync();

switch (access){

case BackgroundAccessStatus.Unspecified:

break;

case BackgroundAccessStatus.AllowedMayUseActiveRealTimeConnectivity:

break;

case BackgroundAccessStatus.AllowedWithAlwaysOnRealTimeConnectivity:

break;

case BackgroundAccessStatus.Denied:

break;

default:

break;

}

var task = new BackgroundTaskBuilder {

Name = "My Task",

TaskEntryPoint = typeof(BackgroundStuff.MyBackgroundTask).ToString()

};

var trigger = new ApplicationTrigger();

task.SetTrigger(trigger);

var condition = new SystemCondition(SystemConditionType.InternetAvailable);

task.Register();

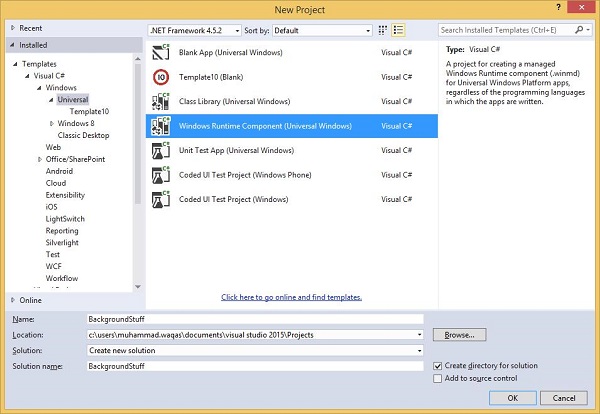
await trigger.RequestAsync();

}

}

}

* Now create another project, but this time select Windows Runtime Component (Universal Windows) from the menu and give the name **Background stuff** to this project.



* Given below is the C# code. which contains **MyBackgroundTask** class implantation and it will run the background task.

using Windows.ApplicationModel.Background;

using Windows.UI.Notifications;

namespace BackgroundStuff {

public sealed class MyBackgroundTask : IBackgroundTask {

public void Run(IBackgroundTaskInstance taskInstance) {

SendToast("Hi this is background Task");

}

public static void SendToast(string message) {

var template = ToastTemplateType.ToastText01;

var xml = ToastNotificationManager.GetTemplateContent(template);

var elements = xml.GetElementsByTagName("Test");

var text = xml.CreateTextNode(message);

elements[0].AppendChild(text);

var toast = new ToastNotification(xml);

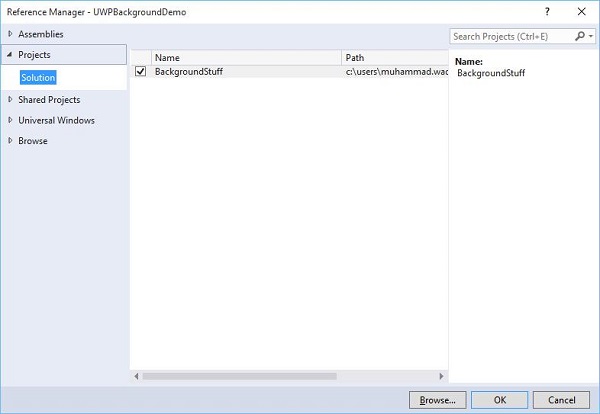
ToastNotificationManager.CreateToastNotifier().Show(toast);

}

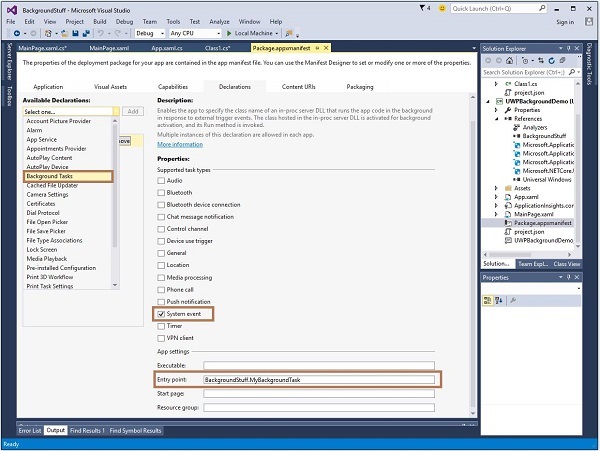
}

}

* To make this project accessible in the **UWPBackgroundDemo** project, right click on **References > Add References** in Solution Explorer and add **BackgroundStuff** project.



* Now, let us go to the **Package.appxmanifest** file of **UWPBackgroundDemo** project and add the following information in Declarations tab.



* First build the Background stuff project, then build and execute the **UWPBackgroundDemo** project.

When you click the **button**, it will run the background task and will show a notification at the right end of your window.