Task Invoker - Mobile Background Tasks Documentation

Online documentation can be found at: https://vuopaja.com/docs/background-tasks.html

General

This plugin makes it possible for an Unity mobile app to request additional background execution time to finish some task when the user or OS puts the app to the background. For example, while interacting with a server it can prevent the system from suspending your app while the operation is in progress.

The plugin consists of an Android .aar library, iOS .mm library and a C# implementation. The included C# TaskInvoker class handles starting and stopping the background tasks. Started tasks invoke a C# callback method periodically while Unity is in either the background or the foreground. This means that you can start doing work with Task Invoker tasks whenever and doing so ensures that the tasks keeps running in case the app is sent to the background. The C# callback methods are invoked on the Unity main thread so Unity components can be accessed as usual.

Note: These tasks are meant for light background work, not for running the whole game in the background. Tasks will not run indefinitely: the tasks will stop running if the app is killed by the user or the OS.

Usage

Using the TaskInvoker is simple. You can start a task by calling the StartTask method. It returns an integer that identifies the task. This ID can be used to stop the task later. Remember to always stop the task once it's finished to save device resources and battery. If the expired event is invoked the task is stopped by the plugin and no further call to stop the task is needed.

Here is the script reference for TaskInvoker C# class:

```
// Starts a task that invokes the callback repeatedly
// Parameters:
// delay - delay before each callback invoke in milliseconds
// onInvoke - callback used to do work while the app is in the background
// onExpire - optional callback that is called if the task is expired by the OS
// Returns: integer ID of the started task
public static int StartTask(int delay, TaskEvent onInvoke, TaskEvent onExpire = null);

// Stops a task with the given taskID
// Parameters:
// taskID - the ID of the task to stop
public static void StopTask(int taskID);

// Stops all running tasks
public static void StopAllTasks();

// Delegate for the callbacks
// Parameters:
// taskID - the ID of the task invoking this callback
public delegate void TaskEvent(int taskId);
```

Here are the steps to setup a simple usage scenario that logs the taskID once and then stops the task:

```
// Create a callback method that matches TaskEvent delegate and
// include the code to be run in the background
void onInvoke(int taskID) {
    // Log the taskID
    Debug.Log("Task invoked with taskID: " + taskID);

    // Stop the task
    Vuopaja.TaskInvoker.StopTask(taskID);
}


// Call StartTask method with the created callback
// This can be called wherever inside the Unity application
// Set the delay to 1000 milliseconds
Vuopaja.TaskInvoker.StartTask(1000, onInvoke);
```

Android Details

The .aar library consists of an interface to be used with AndroidJavaProxy and a class to start and stop background tasks. The background tasks run as long as the process is alive. The tasks will stop invoking if the user or the OS kills the process. The Expired event present in the ITaskInvoker will not be invoked on Android because the killing of the process cannot be catched so that an event could still be invoked.

iOS Details

The iOS plugin uses beginBackgroundTaskWithExpirationHandler to run the background tasks. You can find more details about it in that link. The expiration handler invokes the Expired event in the TaskInvoker C# class so that the case where the operation being run was not completed in time before expiration can be handled.

Examples

There are two example scenes included in the package. One example runs a counter that invokes once per second while the app is in background and displays the elapsed background time in the UI. The other example has an input field in the UI and a button to send a GET request to the link written in the input field. This is to demonstrate that you can use the Task Invoker to start all your web requests with a task so that they will complete even if the app is sent to the background.

Counter Example

MonoBehaviour's OnApplicationPause method is invoked when the Unity application is sent to the background. This is used to start a new Task Invoker task that runs once per second. Then the task is stopped when the app is returned to the foreground.

```
// Unity pause method - called when Unity app is paused
void OnApplicationPause(bool paused)
{
    // Start the counter on pause
    if (paused)
    {
        // Start a task that invokes once every second (1000 milliseconds)
        currentTaskID = TaskInvoker.StartTask(1000, onInvoke, onExpire);
    }
    else if (currentTaskID != -1)
    {

        // Stop the running task when entering foreground
        TaskInvoker.StopTask(currentTaskID);
    }
}
```

The onInvoke callback is simple. It counts the times it is invoked and calls a method to update the UI:

```
void onInvoke(int taskId)
{
    elapsedTime++;
    updateUI();
}
```

WebRequest Example

This example shows how you can use the included utility class WebRequestWrapper to start all your web requests with a task so that they will complete even if the app is sent to the background.

An instance of the WebRequestWrapper is created and callbacks are registered to the events:

```
// Create a new wrapper object and register for callbacks
webRequestWrapper = new WebRequestWrapper();
webRequestWrapper.Completed += onCompleted;
webRequestWrapper.Failed += onFailed;
```

Using the wrapper is simple:

```
// Create a WebRequest as usual
var request = UnityWebRequest.Get("https://www.vuopaja.com");

// Send the created request using the wrapper.
// The wrapper starts a TaskInvoker task so that if the user/OS sends the app to
// the background during this WebRequest it will continue running in the background.
// Takes milliseconds as a parameter that defines how often the wrapper
// checks for the progress of the request.
webRequestWrapper.Send(request, 100);
```

The wrapper invokes events when the request completes or fails:

```
void onCompleted(WrappedRequest wrappedRequest)
{
    // Handle completed WebRequest
    Debug.Log("Download completed");
}

void onFailed(WrappedRequest wrappedRequest, string reason)
{
    // Handle failure
    Debug.LogError(reason);
}
```

For more information and support check out: vuopaja.com