

**Service Bus Durable Task Framework**

**Project: Software Engineering**

**Submitted To:**

**Prof. Dr. Andreas Pech**

**Damir Dobric**

**Submitted By**:

**Md Faisal Amin**

**1099272**

**Submission Date:20.04.2016**

**Contents**

[**1** **Overview** 3](#_Toc448747446)

[**2** **Scope of the project** 3](#_Toc448747447)

[**3** **Description** 3](#_Toc448747448)

[**3.1** **Core Concepts** 3](#_Toc448747449)

[**3.1.1** **Task Hub** 3](#_Toc448747450)

[**3.1.2** **Task Activities** 3](#_Toc448747451)

[**3.1.3** **Task Orchestrations** 4](#_Toc448747452)

[**3.1.4** **Task Hub Worker** 4](#_Toc448747453)

[**3.1.5** **Task Hub Client** 4](#_Toc448747454)

[**4** **Module Design** 5](#_Toc448747455)

[**4.1** **Writing Task Orchestrations** 5](#_Toc448747456)

[**4.2** **Writing Task Activities** 6](#_Toc448747457)

[**4.3** **Orchestration Instance Management** 6](#_Toc448747458)

[**4.4** **Error Handling & Compensation** 6](#_Toc448747459)

[**4.5** **Task Hub Management** 6](#_Toc448747460)

[**5** **Scenarios** 7](#_Toc448747461)

[**5.1** **Upload Video on Dailymotion** 7](#_Toc448747462)

[**5.1.1** **User Case 1** 7](#_Toc448747463)

[**5.1.2** **User Case2** 9](#_Toc448747464)

[**6** **Conclusion** 10](#_Toc448747465)

**Document Version Control:**

| **Date** | **Version** | **Change Reference** | **Author** |
| --- | --- | --- | --- |
| 20.04.2016 | V1.0 | Final Draft | Md Faisal Amin |

# **Overview**

The Service Bus Durable Task Framework provides developers a means to write code orchestrations in C# using the .Net Task framework and the async/await keywords added in .Net 4.5.

Here are the key features of the durable task framework:

* Definition of code orchestrations in simple C# code
* Automatic persistence and check-pointing of program state
* Versioning of orchestrations and activities
* Async timers, orchestration composition, user aided check pointing

The framework itself is very light weight and only requires an Azure Service Bus namespace and optionally an Azure Storage account. Running instances of the orchestration and worker nodes are completely hosted by the user. No user code is executing ‘inside’ Service Bus.

# **Scope of the project [[1]](#footnote-1)**

Dailymotion is a video sharing site on which users can upload, watch and share videos. In this project videos are uploaded to the Dailymotion website using C# console application and then Durable Task Framework is applied to the code. Mainly two tasks are done for uploading video 1) Get access token task for authentication to Dailymotion and 2) Upload video task: Upload video to Dailymotion using that access token. Durable task frame work ensures the state of the execution of these two tasks is preserved durably.

# **Description [[2]](#footnote-2)**

The Service Bus Durable Task Framework allows users to write C# code and encapsulate it within ‘durable’ .Net Tasks. These durable tasks can then be composed with other durable tasks to build complex task orchestrations.

## **Core Concepts**

There are a few fundamental concepts in the framework.

### **Task Hub**

The Task Hub is a logical container for Service Bus entities within a namespace. These entities are used by the Task Hub Worker to pass messages reliably between the code orchestrations and the activities that they are orchestrating.

### **Task Activities**

Task Activities are pieces of code that perform specific steps of the orchestration. A Task Activity can be ‘scheduled’ from within some Task Orchestration code. This scheduling yields a plain vanilla .Net Task which can be (asynchronously) awaited on and composed with other similar Tasks to build complex orchestrations.

### **Task Orchestrations**

Task Orchestrations schedule Task Activities and build code orchestrations around the Tasks that represent the activities.

### **Task Hub Worker**

The worker is the host for Task Orchestrations and Activities. It also contains APIs to perform CRUD operations on the Task Hub itself.

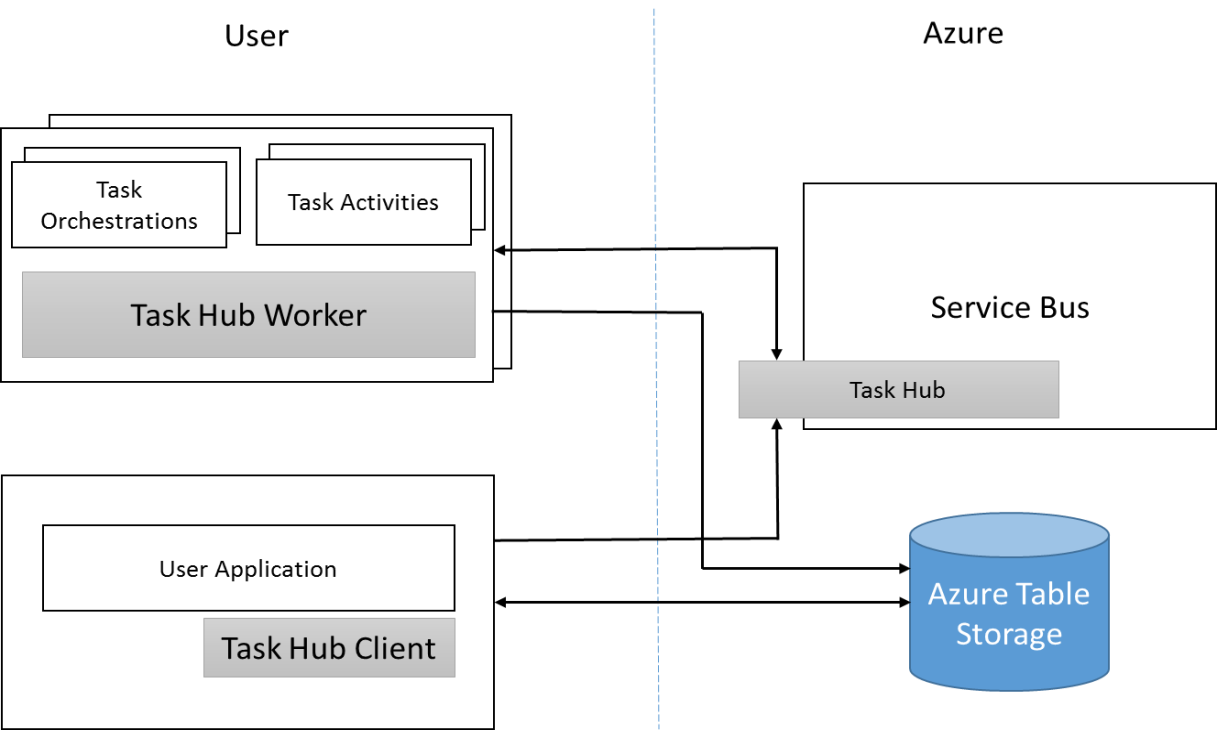
### **Task Hub Client**

The Task Hub Client provides:

* APIs for creating and managing Task Orchestration instances
* APIs for querying the state of Task Orchestration instances from an Azure Table

Both the Task Hub Worker and Task Hub Client are configured with connection strings connection strings for Service Bus and optionally with connection strings for a storage account.

Service Bus is used for storing the control flow state of the execution and message passing between the task orchestration instances and task activities. However Service Bus is not meant to be a database so when a code orchestration is completed, the state is removed from Service Bus. If an Azure Table storage account was configured then this state would be available for querying for as long as it is kept there by the user.



**In this project the bellow tasks are used to implement Durable Task Framework:**

**Task Orchestration:** TaskOrchestration.cs

**Task Activities:** GetAccesTokenTask.cs, UploadVideoTask.cs,

**User Applications: Dailymotion**

The framework provides TaskOrchestration and TaskActivity base classes which users can derive from to specify their orchestrations and activities. They can then use the TaskHub APIs to load these orchestrations and activities into the process and then start the worker which starts processing requests for creating new orchestration instances.

The TaskHubClient APIs are used to create new orchestration instances, query for existing instances and then terminate those instances if required.

# **Module Design**

Assume that user wants to build a code orchestration that will first request for an access token and using this access token user will be able to upload video in Dailymotion. To implement this using the Service Bus Durable Task Framework, the user will have to write two Task Activities, One for getting the access token for user authentication, another one is upload video task which will upload video to Dailymotion by using that access token and one Task Orchestration that orchestrates between the Task activities.

In this orchestration, the user is scheduling get access token task activity, and then waiting for the response and then scheduling the upload video task activity. The framework will ensure that the state of the execution is preserved durably. E.g., if the node hosting the task orchestration above crashed before scheduling the get access token task activity, on restart it will know to schedule this activity. If the node crashed after it had scheduled the activity but before the response came back, on restart it will be smart enough to know that the activity was already scheduled and it will directly start waiting for the response of the get access token task activity.

This is how users can load these orchestration and activity classes in a worker and start processing requests to create new orchestration instances. Multiple instances of these workers can be running concurrently against the same task hub to provide load balancing as required. The framework guarantees that a particular orchestration instance code would only be executing on a single worker at one time.

## **Writing Task Orchestrations [[3]](#footnote-3)**

Task orchestrations basically invoke Task Activities and define how the control flows from one activity to another. The code that can be written within an orchestration is plain C# but with a few constraints. These constraints exist because of how the framework replays the orchestration code. This is described in a nutshell below.

Every time new work needs to be processed by an orchestration (e.g. a Task Activity finished or a timer fired), the framework replays the user’s Task Orchestration code from scratch. Whenever this user code attempts to schedule a Task Activity, the framework intercepts this call and consults the ‘execution history’ of the orchestration. If it finds that the particular Task Activity had already been executed and yielded some result, it would replay that Activity’s result immediately and the Task Orchestration would continue. This would continue happening until the user code has executed to a point where either it is finished or it has scheduled a new Activity. If it is the latter case then the framework would actually schedule and execute the specified Activity. After this Activity is completed its result also becomes part of the execution history and the value would be used in subsequent replays.

The Task Orchestration code is always executed in a single thread. This means that if the code was awaiting multiple tasks and one of them completed followed immediately by another one, the framework is guaranteed to run the continuations for both of these tasks serially.

## **Writing Task Activities**

Task Activities are the ‘leaf’ nodes of an orchestration. This is the code which actually performs a unit of operation within the orchestration. This is plain C# code with no constraints. Task Activity code is guaranteed to be called at least once. However in error cases it might be invoked multiple times so idempotence is desirable.

## **Orchestration Instance Management**

The TaskHubClient API allows users to create new orchestration instances, query for the state of created orchestration instances and terminate these instances. The API for creating an orchestration instance will return the instance information. This information can be used in subsequent APIs to query for the state of the instance.

## **Error Handling & Compensation**

Any exception that is thrown in the TaskActivity code is marshalled back and thrown as a TaskFailedException in the TaskOrchestration code. Users can write the appropriate error handling and compensation code that suits their needs around this. Refer Failure scenario’s below for Error handling.

## **Task Hub Management**

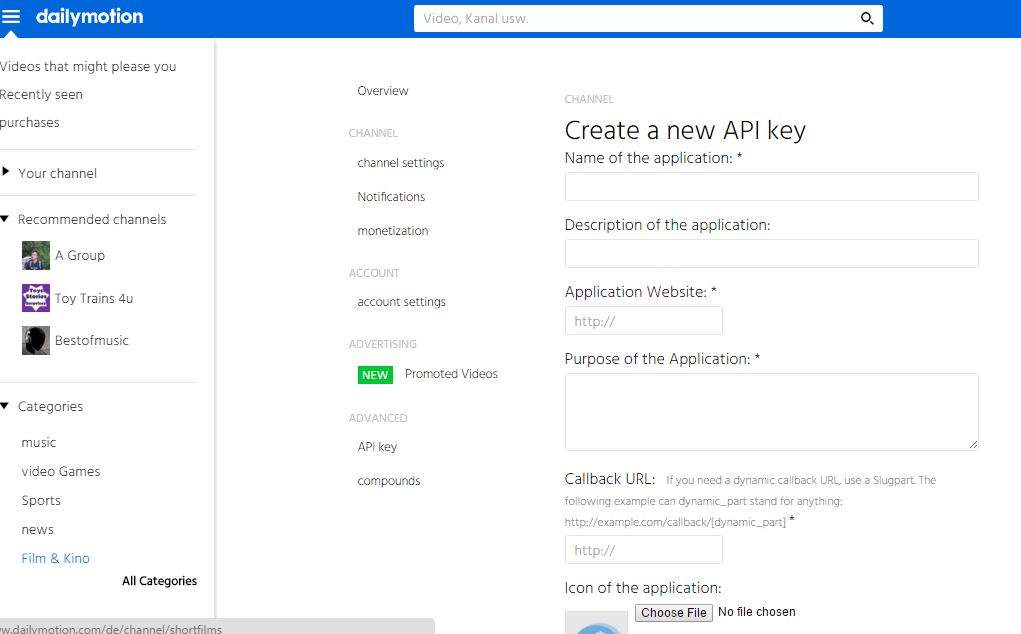
The TaskHubWorker has APIs that can be used to perform CRUD operations on the TaskHub itself. [[4]](#footnote-4)

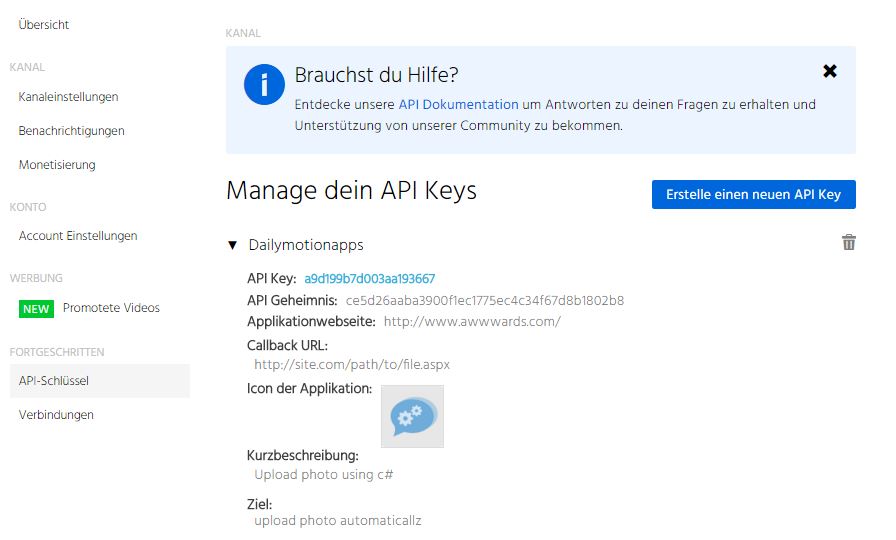
# **Scenarios**

## **Upload Video on Dailymotion**

### **User Case 1**

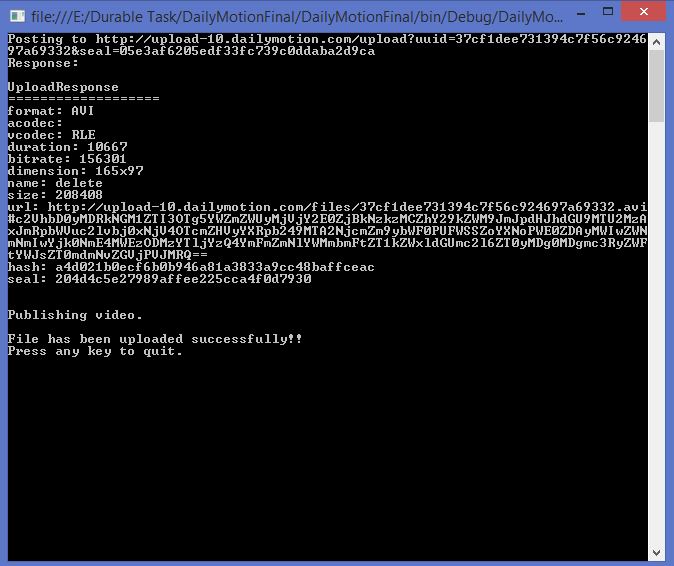
|  |  |
| --- | --- |
| User Case 1 | Create a API key |
| Description | While some basic features are available without authentication, you will need to register yourself as a developer in order to perform more elaborate API calls, authenticate users and act on their behalf. After requesting for a new API key you will get your API Secret, API Key and call back URL. You have to save this credential into a secret.xml file. Set the API key same in secret.xml file as you gave during the app creation. If the User enters the correct user name and password for Dailymotion, then the user will be successfully authenticated and forwarded to the CallBackurl (http://site.com/path/to/file.aspx in this case) which contains the requires access token. |





### **User Case2**

|  |  |
| --- | --- |
| User Case 2 | Successfully Upload Video |
| Description | After the successful authentication of the user, the input video should be uploaded to the Dailymotion website the response can be viewed from the screen. |



# **Conclusion**

With DTF Microsoft provides an easy option for building robust distributed and scalable services with build-in state persistence and program execution check points.[[5]](#footnote-5) Durable Task framework is used to stop repeating the same tasks. Because some tasks need very long time so in such type of cases repetition is not efficient for system. In this project I have implemented a console application which is used to upload video on Dailymotion. Durable task frame work is used to orchestrate the access token and uploading video task to make the system efficient.

**References:**

[1] Service Bus Durable Task Framework, p. 2-10. [Online]. Available: https://abhishekrlal.com/2013/06/27/durable-task-framework-preview-with-azure-service-bus/. Accessed: Oct. 21, 2015

[2] "Durable Task Framework – Episode III: Coded Orchestrations,". [Online]. Available: http://developers.de/blogs/damir\_dobric/archive/2015/09/15/durable-task-framework-episode-iii-coded-orchestrations.aspx. Accessed: Nov. 16, 2015.

1. Service Bus Durable Task Framework, p. 2. [Online]. Available: https://abhishekrlal.com/2013/06/27/durable-task-framework-preview-with-azure-service-bus/. Accessed: Oct. 21, 2015 [↑](#footnote-ref-1)
2. Service Bus Durable Task Framework, p. 3,4. [Online]. Available: https://abhishekrlal.com/2013/06/27/durable-task-framework-preview-with-azure-service-bus/. Accessed: Oct. 21, 2015 [↑](#footnote-ref-2)
3. Service Bus Durable Task Framework, p. 6,7. [Online]. Available: https://abhishekrlal.com/2013/06/27/durable-task-framework-preview-with-azure-service-bus/. Accessed: Oct. 21, 2015 [↑](#footnote-ref-3)
4. Service Bus Durable Task Framework, p. 8. [Online]. Available: https://abhishekrlal.com/2013/06/27/durable-task-framework-preview-with-azure-service-bus/. Accessed: Oct. 21, 2015 [↑](#footnote-ref-4)
5. "Durable Task Framework – Episode III: Coded Orchestrations,". [Online]. Available: http://developers.de/blogs/damir\_dobric/archive/2015/09/15/durable-task-framework-episode-iii-coded-orchestrations.aspx. Accessed: Nov. 16, 2015. [↑](#footnote-ref-5)