

How to

# Usando Web Services de Q-flow



*Versión: 1.1*

*Fecha de publicación 06-04-2011*

*Aplica a: Q-flow 3.0 y Q-flow 3.1*

## Índice

|  |    |
|--|----|
| Introducción.....                        | 3  |
| Los WebServices de Q-flow.....           | 3  |
| WebStart.....                            | 3  |
| WebLists y WebQueue.....                 | 4  |
| WebResponse .....                        | 4  |
| Integrando aplicaciones con Q-flow ..... | 5  |
| Iniciando workflows .....                | 5  |
| El objeto NewFlowMessage .....           | 7  |
| Listando las tareas pendientes .....     | 8  |
| El objeto Task .....                     | 8  |
| Métodos en WebLists .....                | 9  |
| Métodos en WebQueue.....                 | 10 |
| Respondiendo tareas .....                | 11 |
| El objeto TaskMessage.....               | 13 |

## Introducción

El objetivo de este documento es brindar una breve introducción a los principales WebServices de Q-flow, describiendo y ejemplificando el uso de los mismos para: el inicio de flujos, listado de tareas pendientes y la obtención de la información de una tarea, para contestarla mediante el mismo mecanismo. Para obtener un conocimiento más a fondo sobre este tema recomendamos consultar los manuales del producto, en los cuales se describen todos los WebServices y su forma de uso.

Se asume que el lector tiene conocimiento de cómo usar las tecnologías ASP.NET con el uso de WebServices y conocimientos básicos de Q-flow.

Como resultado de esta práctica, el lector será capaz de iniciar flujos, contestar y listar tareas pendientes sin utilizar el sitio de Q-flow, permitiendo una posible integración con sistemas desarrollados.

## Los WebServices de Q-flow

### WebStart

El WebService WebStart provee métodos que nos facilitan el inicio de flujos en la herramienta. Dentro de sus métodos destacamos dos, que son los que utilizaremos más adelante en los ejemplos que realicemos, ellos son:

- ✓ **GetNewFlowInfoCorrelativeId(long templateCorrelativeID)**  
Este método nos permite obtener un objeto de tipo *NewFlowMessage*, que posee la información requerida para el inicio de un flujo en particular dado el Id correlativo del *template* que queremos iniciar. Este tipo de datos no es más que un XML, el cual puede utilizarse, editándolo, para cargar los datos requeridos al inicio del flujo y luego utilizar el método *StartFlow* para iniciar un flujo nuevo.
- ✓ **StartFlow(NewFlowMessage flowMessage)**  
Este método nos permite iniciar un flujo mediante la información contenida dentro del *NewFlowMessage* que recibe como parámetro. A su vez, luego de iniciar, nos devuelve un objeto de tipo *Guid* que contiene el Id del flujo que acabamos de iniciar.

### WebLists y WebQueue

Estos WebServices hacen posible el listado de tareas pendientes de respuesta, de los flujos activos en el sistema. En el caso de WebLists lista las tareas pendientes asignadas a un usuario, por otro lado WebQueue nos posibilita listar tareas que pertenezcan a una cierta cola de trabajo. En este documento mostraremos los siguientes métodos:

- ✓ **GetCurrentUserTasks()**  
Método que devuelve una lista de tareas, *Task[]*, que tienen como destinatario al usuario que invoca el WebService. Con la información contenida en la tarea se puede componer para obtener la URL para responder la tarea.
- ✓ **GetCurrentUserTasksByTemplatesWithData(Guid[] templatesId, Guid[] dataToInclude)**  
Método que devuelve un *DataTable* con la información de las tareas de los *templates* seleccionados, asignadas al usuario invocador. También pueden mostrarse algunos datos de aplicación de esas tareas mediante la lista *dataToInclude*.
- ✓ **GetMyWorkQueueTasks(Guid workQueueId)**  
Dado un identificador de cola, nos devuelve la lista de tareas de esa cola, que fueron tomadas por el usuario que invoca el WebService.
- ✓ **GetUnassignedWorkQueueTasks(Guid workQueueId)**  
Dado un identificador de cola, nos devuelve la lista de tareas de esa cola, que no han sido tomadas por ningún usuario.

### WebResponse

WebResponse hace posible listar la información de una tarea y responderla. Entre los servicios que expone destacamos los siguientes:

- ✓ **GetTask(Guid FlowId, Guid StepId, Guid Told)**  
Este método permite obtener un objeto del tipo *TaskMessage* que contiene la información de la tarea y puede ser utilizado para responder la misma.
- ✓ **RespondTask(TaskMessage task)**  
Permite responder la tarea basado en un objeto de tipo *TaskMessage* que puede ser obtenido utilizando el método *GetTask* y modificado para responderla.
- ✓ **RespondTaskNow(Guid FlowId, Guid StepId, Guid Told, string ResponseKey)**  
Este WebService nos permite responder una tarea, con solamente la respuesta indicada en el parámetro *ResponseKey*.

### Integrando aplicaciones con Q-flow

Los WebServices de Q-flow nos proveen una gran facilidad para integrar nuestra aplicación web o de escritorio con la herramienta. A continuación mostraremos de qué forma utilizar los métodos nombrados anteriormente para interactuar con Q-flow.

### Iniciando workflows

Para el inicio de un flujo primero debemos obtener la información del mismo mediante el método *GetNewFlowInfoCorrelativeId* que nos devuelve un objeto que contiene los datos para el inicio del flujo. Luego cargamos los valores de esos datos, usando por ejemplo LINQ, para después utilizar el método *StartFlow* que inicie el flujo con el objeto que editamos. A continuación mostramos un ejemplo de cómo podría hacerse esto:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using QflowWSStart;

public partial class StartFlow : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected void cmdIniciar_Click(object sender, EventArgs e)
    {
        if (Page.IsValid)
        {
            QflowWSStart.WebStart proxy = new QflowWSStart.WebStart();
            proxy.Credentials = GetUserCredentials();
            var flow = proxy.GetNewFlowInfoCorrelativeId(235);
        }
    }
}
```

En el código anterior, vemos como se crea una instancia de *WebStart*, se le asignan las credenciales del usuario que inicia el flujo y se obtiene, con el método *GetNewFlowInfoCorrelativeId*, la información del *template* con id '235'. Este valor se puede obtener desde el sitio web de Q-flow bajo la opción *Templates*, en la columna ID nos da los correlativos de todos los flujos que están puestos en producción. Es importante para poder invocar a los WebServices de Q-flow, que las credenciales provistas sean las de un usuario válido de la herramienta, de lo contrario fallará con error: "Http 404, unauthorized". El usuario utilizado para invocar *StartFlow* es el que aparecerá como iniciador del flujo.

```
flow.FlowName = "Título del flow";  
SetFlowData(flow, "Nombre del dato 1", "valor del dato 1");  
SetFlowData(flow, "Nombre del dato 2", "valor del dato 2");
```

En el fragmento anterior cargamos el título del flujo y utilizamos la función auxiliar *SetFlowData* que su función es cargar, mediante LINQ, el valor del dato, en el nodo correspondiente al mismo para la información del flujo. Para el código del procedimiento auxiliar véase más abajo.

```
        Guid flowId = proxy.StartFlow(flow);  
    }  
}
```

Por último utilizando la variable *flow* a la cual se le agregaron los datos para el inicio, invocamos a *StartFlow* para iniciar el flujo. Este método nos devuelve el Id del flujo que se instancia.

Como vimos anteriormente, al ingresar los valores a los datos del XML del formulario de inicio, utilizamos una función, que mediante consultas LINQ carga los valores deseados. De forma análoga podría hacerse lo mismo para los roles cargando los usuarios. Esta función es a modo de ejemplo ya que cualquier procedimiento utilizado para modificar el XML es válido.

```
protected void SetFlowData(NewFlowMessage flow, string dataName, string dataValue)  
{  
    var data = (from d in flow.Data where d.Name == dataName select d)  
                .SingleOrDefault();  
    if (data != null)  
        data.Values = new string[] { dataValue };  
}
```

Cabe aclarar que los datos que marcados como requeridos al inicio del flujo deberán ser ingresados, de lo contrario *StartFlow* dará un error en tiempo de ejecución, pues los datos requeridos son obligatorios para iniciar un flujo.

## El objeto `NewFlowMessage`

El elemento `NewFlowMessage` devuelto por el método `GetNewFlowInfoCorrelativeId` es un objeto del tipo XML que tiene la siguiente estructura, en la cual podemos notar que comienza con cierta información del flujo como el nombre del flujo, descripción, nombre del `template`, entre otros. Luego un nodo `Data`, en el cual aparecen los datos de aplicación que deben ser cargados para instanciar un flujo, algunos de ellos serán requeridos, otros no. Por último vemos los roles que son editables al inicio.

```
<?xml version="1.0" encoding="utf-8" ?>
<NewFlowMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://urudata.qflow.com/">
  <Importance>Normal</Importance>
  <Progress>0</Progress>
  <Flag />
  <FlowName />
  <FlowDescription />
  <TemplateId>6d110831-4176-4c85-b446-57b726e1adfc</TemplateId>
  <VersionId>439d52c1-7c3b-4ec6-9c36-2c3d9710758e</VersionId>
  <TemplateName>Integracion con Qflow</TemplateName>
  <TemplateDescription />
  <TemplateVersionName>1.0</TemplateVersionName>
  <TemplateVersionDescription>1.0</TemplateVersionDescription>
  <Data>
    <DataMessage>
      <DataId>ae49d070-bb7c-4f26-bde7-4b531118fb7f</DataId>
      <Name>Dato 1</Name>
      <IsArray>>false</IsArray>
      <DataType>System.Boolean</DataType>
      <Values />
    </DataMessage>
    <DataMessage>
      <DataId>c4149a7c-36a9-4793-9d5c-0c20d6bb29f4</DataId>
      <Name>Dato 2</Name>
      <IsArray>>false</IsArray>
      <DataType>System.Decimal</DataType>
      <Values />
    </DataMessage>
  </Data>
  <Lines />
  <Roles>
    <RoleMessage>
      <Id>70893cec-cd0b-40cd-847e-f46b056bea83</Id>
      <Name>Rol 1</Name>
      <IsMultiUser>>false</IsMultiUser>
      <Members />
    </RoleMessage>
  </Roles>
  <Attachments />
</NewFlowMessage>
```

### Listando las tareas pendientes

A continuación mostraremos como obtener las tareas pendientes del sistema mediante la invocación a WebServices. También mostraremos el contenido del objeto *Task*, que es el que se devuelve al invocar algunos de los servicios mencionados.

#### El objeto Task

Los WS de Q-flow utilizan este objeto para devolver la información de la tarea para desplegarla en una lista. Los servicios devuelven una lista de tareas, *Task[]*, la cual contiene información mínima, que después puede ser utilizada para generar el link que nos lleve al formulario de respuesta de la tarea. Esta información incluye el id del Flujo, el id de la tarea y el id de la tarea del usuario (si una tarea está dirigida a varios usuarios hay varias tareas con distinto 'Told'), además incluye el asunto de la misma para ser desplegada. A continuación vemos un ejemplo de un *Task[]*:

```
<?xml version="1.0" encoding="utf-8" ?>
<ArrayOfTask xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://tempuri.org/">
  <Task FlowId="9d708831-a095-4592-8db4-e90fc3ba8b10"
    TaskId="df981529-9b02-4faf-a0f0-ddc12b85bd9a"
    ToId="dac998f1-6ed8-43d2-a286-92c3957d9610">
    <TaskSubject>Cancelar el pedido nro. 3586</TaskSubject>
  </Task>
  <Task FlowId="f6d4a5bc-1a48-4d1c-8516-744f776b4f63"
    TaskId="9b143efc-3648-400f-b47f-a4b83d8f9d4b"
    ToId="ebbccf5e-343e-4413-a442-0659228f73ee">
    <TaskSubject>Dar de baja la tarjeta nro. 1234-56789</TaskSubject>
  </Task>
</ArrayOfTask>
```

De esta forma, si quisiéramos acceder al formulario para responder la primera tarea desde Q-flow, podríamos generar el link de la siguiente forma:

```
string.Format(
"http://WebServerName/QflowWebSite/CustomForms/TaskResponseWithGroups.aspx
?FlowID={0}&FlowStepID={1}&FlowStepToID={2}", flowId, flowStep, flowStepTo);
```

Donde las 3 variables son obtenidas del Webservice invocado anteriormente. Esto nos dirige al formulario que Q-flow contiene para responder la tarea, de la misma manera que si el ingreso lo hubiésemos hecho mediante el sitio de Q-flow.

## Métodos en WebLists

WebLists, nos brinda métodos para acceder de forma fácil a las tareas de un usuario. La forma más sencilla de obtener la información de las tareas pendientes del usuario es utilizar el método *GetCurrentUserTasks* el cual devuelve la lista de tareas del usuario que invoca el servicio, esto puede hacerse de la siguiente forma:

```
WebLists webLists = new WebLists();
Task[] tasks = webLists.GetCurrentUserTasks();
```

Luego se puede recorrer *tasks* para obtener la información de las tareas pendientes del usuario y eventualmente ingresar a responder alguna de ellas. La lista devuelta por este método, está ordenada por la fecha en la cual se generó la tarea de forma descendente, es decir que el primer elemento de esta lista, es la tarea más nueva que se le generó al usuario invocador.

Otra forma, no tan sencilla pero si mucho más completa, que nos brinda más información de la tarea y además nos posibilita seleccionar datos de aplicación para ser desplegados, es el método *GetCurrentUserTasksByTemplatesWithData*, el cual nos permite elegir los *templates* para los cuales se listan las tareas y los datos de esas tareas que se desean mostrar en la tabla. Además de los datos elegidos, el método nos devuelve cierta información adicional del flujo, la cual listaremos a continuación los más importantes:

- ✓ **FlowId, TaskId, TaskTold:** Similar a el objeto *Task*, tienen la información que identifica la tarea
- ✓ **TemplateName, TemplateVersionName:** Nombre y versión del *template* al que pertenece la tarea
- ✓ **FlowName, FlowFlag, FlowStartDate:** Nombre, bandera actual y fecha de comienzo del flujo
- ✓ **TaskName, TaskDescription:** Nombre y descripción de la tarea
- ✓ **TaskSubject, TaskTimeStarted:** Asunto y fecha de comienzo de la tarea

Luego de estos campos, se agrega una lista con cada dato de aplicación solicitado, donde las columnas se nombran como *CustomApplicationData\_ID* donde ID representa el *Guid* del dato de aplicación seleccionado. A continuación vemos cómo se puede invocar a este *WebService* y para el caso del ejemplo cargar un *DataGridView* de un *WindowsForm* que despliega los datos:

```
public ShowDataOnWinForm()
{
    WebLists w1 = new WebLists();
    w1.Credentials = CredentialCache.DefaultCredentials;

    List<Guid> templates = new List<Guid>();
    templates.Add(new Guid("7bf18afe-dec5-4e63-9c1d-f242786f3f2b"));
    templates.Add(new Guid("aa056346-a937-4b1f-88e9-ca5f6df786fc"));
}
```

```
List<Guid> data = new List<Guid>();
data.Add(new Guid("3a68692f-98e2-46aa-ab75-2fe032d11a6a"));
data.Add(new Guid("f01dc00d-706b-4a98-86e3-11821c52f15f"));

formGrid.DataSource =
wl.GetCurrentUserTasksByTemplatesWithData(templates.ToArray(), data.ToArray());
}
```

### Métodos en WebQueue

De la misma manera que con WebLists, se puede hacer algo similar en el caso de que se utilicen colas de trabajo, invocando los métodos en WebQueue. Si queremos la lista de tareas de una cola que no fueron tomadas por ningún usuario, podemos utilizar el método *GetUnassignedWorkQueueTasks* el cual toma como parámetro el id de la cola de trabajo de la que se quiere obtener la lista de tareas. Un ejemplo de cómo invocar a este Webservice es el siguiente:

```
Public void TareasDeCola()
{
    WebQueue wq = new WebQueue();
    wq.Credentials = CredentialCache.DefaultCredentials;

    Guid queueId = new Guid("f01dc00d-706b-4a98-86e3-11821c52f15f");
    Task[] queueTasks = wq.GetUnassignedWorkQueueTasks(queueId);
}
```

Este método devuelve una lista de objetos *Task*, los cuales como vimos antes, poseen la información para poder acceder a la tarea. El otro método que mencionamos de WebQueue, *GetMyWorkQueueTasks*, nos posibilita listar las tareas que pertenecen a la cola de trabajo seleccionada y que fueron tomadas por el usuario invocador. Los parámetros y el resultado de invocar a este método, es el mismo que al que vimos anteriormente, por lo cual no ahondaremos en ejemplos.

### Respondiendo tareas

Los métodos de `TaskResponse` presentados anteriormente nos permiten contestar una tarea identificándola con los tres id que, como mencionamos, identifican una tarea, el *FlowId*, *TaskId* y el *TaskTold*. Además de estos parámetros debemos proveer la respuesta a la tarea, la cual se pasa como parámetro mediante un *string* que indica el estado de la tarea. Al igual que en la web de Q-flow, una tarea puede ser contestada con tres estados: no iniciada, en progreso y finalizada. Para esto basta con pasarle al parámetro *ResponseKey* alguno de los siguientes valores, según corresponda: "Not Started" para marcar que la tarea todavía no ha comenzado a realizarse, "In Progress" para el caso que sea necesario indicar que la tarea está en proceso y "Finished" que indica que la tarea ha sido completada.

Los métodos que veremos a continuación, nos brindan la posibilidad de responder las tareas de dos formas. La primera y más sencilla de ellas es utilizar el método *RespondTaskNow* en el cual identificando la tarea y pasándole un *string* con la respuesta, la misma queda contestada, con el estado especificado en la respuesta. Este método tiene sus limitaciones, las tareas que requieren el ingreso o modificación de algún dato de aplicación, no pueden ser contestadas de este modo, ya que no acepta parámetros para los datos de aplicación. Por otro lado, la simpleza de este mecanismo, nos da una forma rápida de contestar tareas en las que el asunto de la misma me basta para responderla, que no requieren el ingreso ni la visualización de datos. Veamos a continuación un ejemplo de ejecución donde se obtiene la lista de tareas pendientes y luego se contesta la primera de la lista.

```
public void ContestarPrimerTarea()
{
    WebLists webLists = new WebLists();
    webLists.Credentials = CredentialCache.DefaultCredentials;

    WebResponse webResponse = new WebResponse();
    webResponse.Credentials = CredentialCache.DefaultCredentials;

    Task[] pendientes = webLists.GetCurrentUserTasks();
    Task tarea = pendientes.First();

    //Posibles respuestas: Not Started , In Progress , Finished
    webResponse.RespondTaskNow(tarea.FlowId, tarea.TaskId, tarea.ToId, "Finished");
}
```

En este ejemplo utilizamos `WebLists` para obtener la lista de tareas pendientes del usuario, luego tomamos el primer elemento de la lista, la tarea más nueva. De la tarea obtenida, tomamos el *FlowId*, *TaskId* y *Told*, y con el *string* de la respuesta, en este caso "Finished", invocamos el `WebService` para dar por contestada y finalizada a la tarea.

La otra posibilidad para responder a una tarea es invocar dos operaciones: *GetTask* para obtener la información de la tarea, un objeto de tipo *TaskMessage* el cual veremos más adelante, para luego de manera similar a *StartFlow*, responder la tarea mediante el método *RespondTask*. Si bien es un poco

## Patterns & Practices

más trabajosa esta idea de utilizar dos métodos para contestar la tarea, esto nos permite visualizar los datos que son visibles para la tarea, así como editar o ingresar los datos que sean editables o requeridos. De esta forma la información en Q-flow facilita la tarea del usuario al contestar la tarea, quien además puede agregar información de la tarea al flujo. El siguiente fragmento de código, similar al anterior, es de una función que devuelve el *TaskMessage* para la primera tarea de la lista de pendientes:

```
public static TaskMessage ObtenerInfoTarea()
{
    WebLists webLists = new WebLists();
    webLists.Credentials = CredentialCache.DefaultCredentials;

    WebResponse webResponse = new WebResponse();
    webResponse.Credentials = CredentialCache.DefaultCredentials;

    Task[] pendientes = webLists.GetCurrentUserTasks();
    Task tarea = pendientes.First();

    return webResponse.GetTask(tarea.FlowId, tarea.TaskId, tarea.ToId);
}
```

Con esta información eventualmente se desplegaría la información obtenida, el usuario podría ingresar o modificar la información de la tarea y luego la misma sería respondida, de forma similar al método anterior, con un *string* que oficia de clave e indica el estado de la tarea. Luego cuando el usuario confirma la acción, la tarea se contesta invocando *RespondTask* y pasándole como parámetro el *TaskMessage* que fue editado por el usuario. El siguiente código nos muestra cómo debería ser esto:

```
public static void ResponderTarea(TaskMessage taskInfo)
{
    WebResponse webResponse = new WebResponse();
    webResponse.Credentials = CredentialCache.DefaultCredentials;

    webResponse.RespondTask(taskInfo);
}
```

De esta forma la tarea queda contestada, con la información que el usuario ingresó o modificó dentro del objeto *TaskMessage*.

## El objeto `TaskMessage`

Lo último que nos resta por ver para poder responder todo tipo de tareas es el contenido de `TaskMessage`. Este objeto es el contenedor de toda la información de la tarea y el que utilizaremos para contestarla cuando invoquemos `RespondTask`. A continuación veremos la estructura del objeto:

```
<?xml version="1.0" encoding="utf-8" ?>
<GetTaskResponse xmlns="http://tempuri.org/">
  <GetTaskResult>
    <FlowStepId>ac86f5db-1b6a-4885-8f79-b85c19ee5425</FlowStepId>
    <FlowStepToId>83fdf5c5-dd91-4923-9b08-0c4b0b06e05a</FlowStepToId>
    <ResponseKey />
    <Progress />
    <TaskName>Ingreso de boleta</TaskName>
    <TaskDescription />
    <TaskSubject>Ingresar la boleta del Cliente</TaskSubject>
    <FlowId>a4762c8f-f79a-4c1c-a52b-09678d7f3349</FlowId>
    <FlowCorrelativeId>303</FlowCorrelativeId>
    <FlowName>Declaración de Gastos</FlowName>
    <FlowDescription />
    <TemplateId>6d110831-4176-4c85-b446-57b726e1adfc</TemplateId>
    <VersionId>439d52c1-7c3b-4ec6-9c36-2c3d9710758e</VersionId>
    <TemplateName>Declaraciones</TemplateName>
    <TemplateDescription />
    <TemplateVersionName>1.0</TemplateVersionName>
    <TemplateVersionDescription>1.0</TemplateVersionDescription>
    <Data>
      <DataMessage>
        <DataId>63254999-171b-41af-8166-a97ceadc577c</DataId>
        <Name>Dirección</Name>
        <IsArray>false</IsArray>
        <DataType>System.String</DataType>
        <Values xsi:nil="true" />
      </DataMessage>
      <DataMessage>
        <DataId>7a987d12-90e1-418e-90eb-778102e359cd</DataId>
        <Name>Nombre completo</Name>
        <IsArray>false</IsArray>
        <DataType>System.String</DataType>
        <Values xsi:nil="true">José Ángel Fuentes</Values>
      </DataMessage>
    </Data>
    <Roles />
  </GetTaskResult>
</GetTaskResponse>
```

Vemos como aparece la información del flujo y de la tarea en si, además de una lista que contiene los datos de aplicación y eventualmente una conteniendo los roles visibles de la tarea. Para modificar o ingresar los datos de aplicación utilizaremos una función como la utilizada para cargar los datos de aplicación al inicio del flujo, que consulte mediante LINQ la lista de datos e ingrese el deseado. Un posible código de función podría ser el siguiente:

## Patterns & Practices

```
protected void SetTaskData(TaskMessage task, string dataName, string dataValue)
{
    var data = (from d in task.Data where d.Name == dataName select d)
                .SingleOrDefault();
    if (data != null)
        data.Values = new string[] { dataValue };
}
```

Algo similar podría hacerse con los Roles si hubiera alguno editable. Luego de modificar e ingresar los datos de aplicación, nos interesa marcar la tarea con algún estado ya sea no iniciada, finalizada o en proceso. Para esto utilizamos la información de la tarea, si *task* es un objeto de tipo *TaskMessage* veamos cómo sería:

```
task.ResponseKey = "Finished";
```

De esta forma marcamos la tarea como finalizada, y luego la responderíamos con el método visto anteriormente. De manera análoga podría contestarse como en progreso, salvo que en este caso, podemos ponerle un número que indica que porcentaje de la tarea fue realizado o completado:

```
task.ResponseKey = "In Progress";
task.Progress = 40;
```

Asignarle el valor '40' a *Progress* y marcarla como en progreso significa que la tarea se está llevando a cabo y que se completó un 40% de la misma.