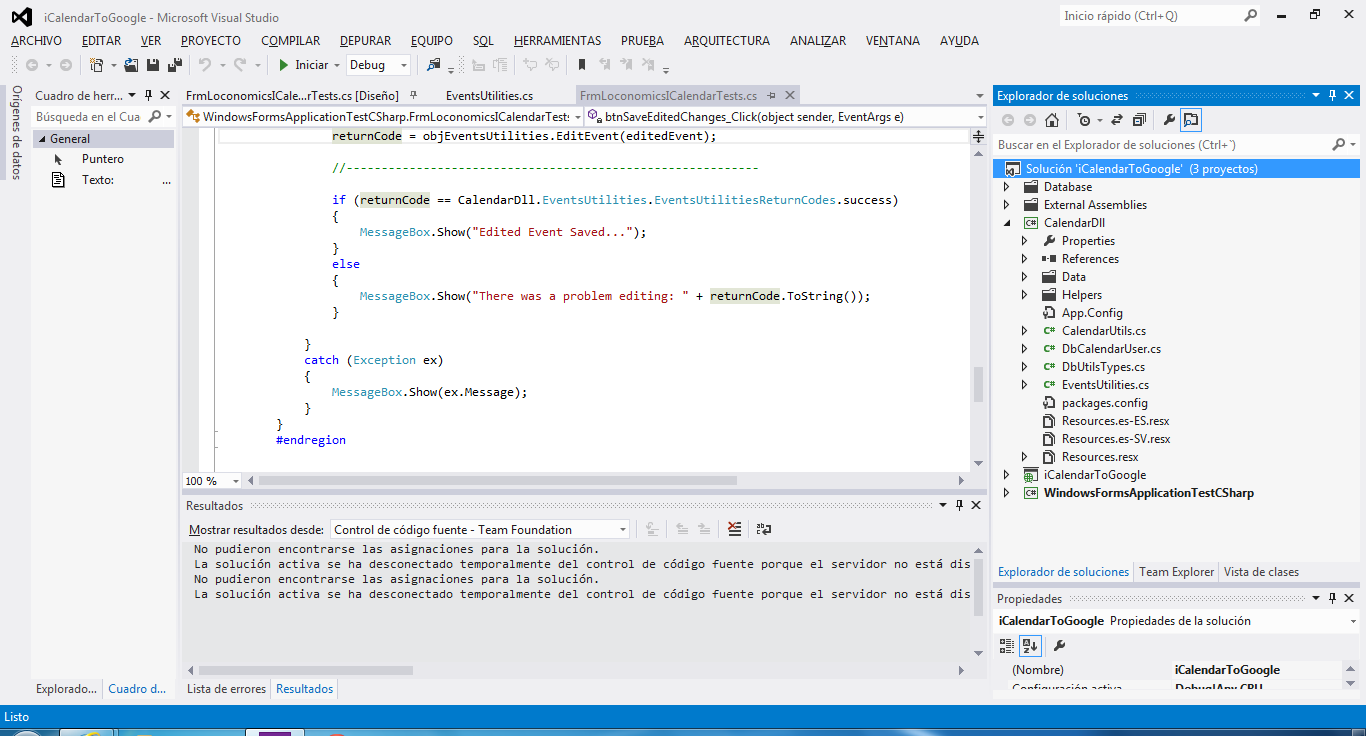
# CRUD Library for Loconomics iCalendar

In the version before this one, all CRUD operations were done directly from the Controllers in the Web Page to the Entity Framework model.

Because of the way Loconomics will work, the appointments should also be generated from a Library, and not interactively by a user.

That is why we are adding a CRUD library in Calendar.dll called *EventsUtilities.cs*.   
You can find it in the *Solution Explorer* (see image below)

We also added a Windows Forms Test project, which uses the Library, so you can test and see how the library is used. This test project is called *WindowFormsApplicationTestCSharp*



The library currently creates “raw” records in the Loconomics Database. It gets the content of all the fields and writes them as they come. In the future, more abstracted methods could be created that don’t use all of the available fields, or that interpret the contents of some fields in a more friendly manner.

## Functions in the EventsUtilities class

### Create Event

public EventsUtilitiesReturnCodes  
 **CreateEvent**(  
 CalendarDll.Data.CalendarEvents **newCalendarEvent**)

We pass inside the parameter *newCalendarEvent* of type CalendarDll.Data.CalendarEvents all the necessary fields to create an Event in Loconomics. As we mentioned about, currently we pass all the raw data, as it will be stored in the database.

The function returns an enumeration of *EventsUtilitiesReturnCodes* which tells if the function was succesful or not. For example, it tells if there was already another record with the same UID identifier.

### Read (Get) Event by UID

public CalendarDll.Data.CalendarEvents   
 **GetEventByUID**( string **UIDToReturn** )

We pass an *UID* identifier, and the function returns us an Event of the type *Public.CalendarDll.Data.CalendarEvents*. If not found, It returns *null*.

### Edit Event

public EventsUtilitiesReturnCodes  
 **EditEvent**(  
 CalendarDll.Data.CalendarEvents **calendarEventWithChanges**)

We pass a parameter *calendarEventWithChanges*, which is an event with the changed data for the Event to Edit. The system uses the UID field to search for the Event that will be changed.

It returns a return code of the *EventsUtilitiesReturnCodes* type, which shows if it was succesful, or there was a problem (like if the UID is not found)

### Delete Event

public EventsUtilitiesReturnCodes  
 **DeleteEvent**(  
 string **UIDToDelete**)

We pass the UID of the Event that we want to delete in the *UIDToDelete* parameter, and the function deletes the Event, returning a return code of the enumeration EventsUtilitiesReturnCodes. This return code can be of success or of a problem (like, if UID doesn’t exist)

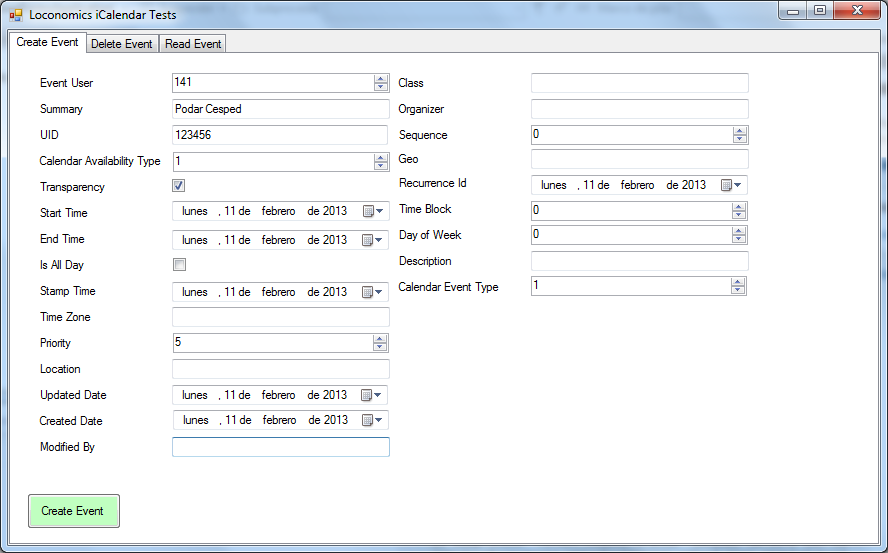
## Using the Test Program to test the CRUD functionality

Set the Start Project to *WindowsFormsApplicationTestCSharp* and run it.

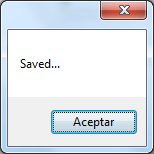
You’ll see the following form, which allows you to Create, Delete, and Read and Edit records for storing Events. These forms have code that use the *EventsUtilities* library, and You can use them as examples for your own code.

### Create Event

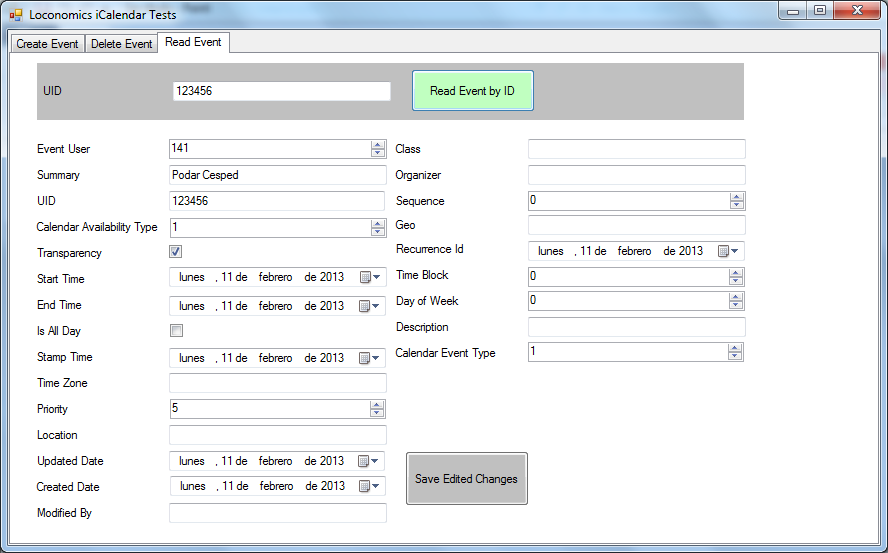
To *Create Event*, go to the tab with that name, enter some information, and then click on the *Create Event* button.



A confirmation Pop Up appears:



### Read Event



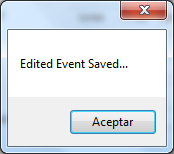
Go to the *Read Event* tab,   
and in the *UID* in the upper grey box enter the id you want to recover.   
Then click on the *Read Event by ID* button.

The information of the Event that was read should fill the controls below.

### Edit Event

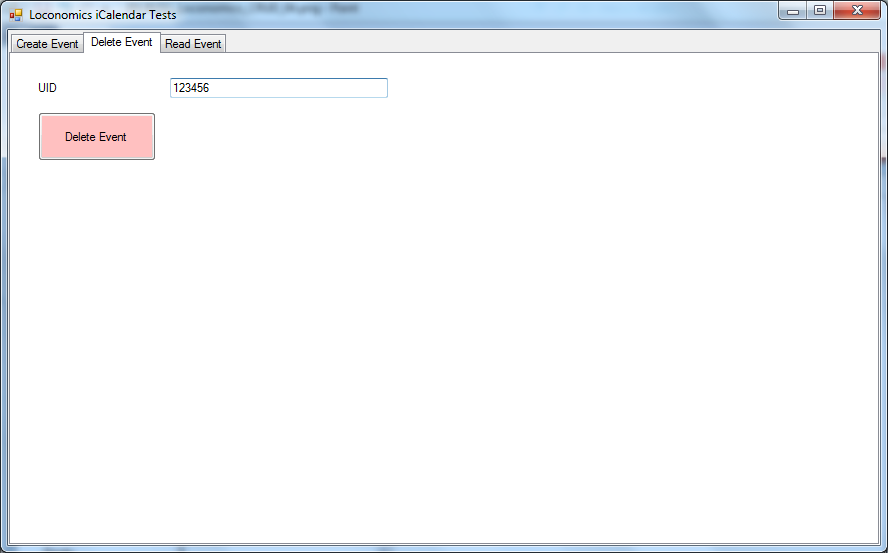
For editing an event, make some changes over the previously *Read* event,  
and then click on the button below: *Save Edited Changes*.

A confirmation message should appear:



### Delete Event

To delete an Event, go to *Delete Event* tab,   
enter the UID of the Event to be deleted,  
and click on the *Delete Event* button.



After deleted, a message appears:

