The following event handler for the DocumentSavingAs event checks if the ProjectInformation Status parameter is empty, and if it is, cancels the SaveAs event. Note that if your application cancels an event, it should offer an explanation to the user.

Code Region 24-2: Canceling an Event

```
private void CheckProjectStatusInitial(Object sender, DocumentSavingAsEventArgs args)
{
        Document doc = args.Document;
        ProjectInfo proInfo = doc.ProjectInformation;

        // Project information is only available for project document.
        if (null != proInfo)
        {
                if (string.IsNullOrEmpty(proInfo.Status))
                {

                        // cancel the save as process.
                        args.Cancel = true;
                        MessageBox.Show("Status project parameter is not set.  Save is aborted.");
                }
        }
}
```

Note that although most event arguments have the Cancel and Cancellable properties, the DocumentChanged and FailuresProcessing events have corresponding Cancel() and IsCancellable() methods.

External Events

The Revit API provides an External Events framework to accommodate the use of modeless dialogs. It is tailored for asynchronous processing and operates similarly to the Idling event with default frequency.

To implement a modeless dialog using the External Events framework, follow these steps:

1. Implement an external event handler by deriving from the IExternalEventHandler interface
2. Create an ExternalEvent using the static ExternalEvent.Create() method
3. When an event occurs in the modeless dialog where a Revit action needs to be taken, call ExternalEvent.Raise()
4. Revit will call the implementation of the IExternalEventHandler.Execute() method when there is an available Idling time cycle.

IExternalEventHandler

This is the interface to be implemented for an external event. An instance of a class implementing this interface is registered with Revit, and every time the corresponding external event is raised, the Execute method of this interface is invoked.

The IExternalEventHandler has only two methods to implement, the Execute() method and GetName() which should return the name of the event. Below is a basic implementation which will display a TaskDialog when the event is raised.

Code Region: Implementing IExternalEventHandler

```
1. public class ExternalEventExample : IExternalEventHandler
2. {
3.     public void Execute(UIApplication app)
4.     {
5.         TaskDialog.Show("External Event", "Click Close to close.");
6.     }
7.
8.     public string GetName()
9.     {
10.        return "External Event Example";
11.    }
12. }
```

ExternalEvent

The ExternalEvent class is used to create an ExternalEvent. An instance of this class will be returned to an external event's owner upon the event's creation. The event's owner will use this instance to signal that the event should be called by Revit. Revit will periodically check if any of the events have been signaled (raised), and will execute all events that were raised by calling the Execute method on the events' respective handlers.

The following example shows the implementation of an IExternalApplication that has a method ShowForm() that is called from an ExternalCommand (shown at the end of the code region). The ShowForm() method creates a new instance of the external events handler from the example above, creates a new ExternalEvent and then displays the modeless dialog box which will later use the passed in ExternalEvent object to raise events.

Code Region: Create the ExternalEvent

```
public class ExternalEventExampleApp : IExternalApplication
{
    // class instance
    public static ExternalEventExampleApp thisApp = null;
    // ModelessForm instance
    private ExternalEventExampleDialog m_MyForm;

    public Result OnShutdown(UIControlledApplication application)
    {
        if (m_MyForm != null && m_MyForm.Visible)
        {
            m_MyForm.Close();
        }

        return Result.Succeeded;
    }

    public Result OnStartup(UIControlledApplication application)
    {
        m_MyForm = null;   // no dialog needed yet; the command will bring it
        thisApp = this;  // static access to this application instance

        return Result.Succeeded;
    }

    //   The external command invokes this on the end-user's request
    public void ShowForm(UIApplication uiapp)
    {
        // If we do not have a dialog yet, create and show it
        if (m_MyForm == null || m_MyForm.IsDisposed)
        {
            // A new handler to handle request posting by the dialog
            ExternalEventExample handler = new ExternalEventExample();

            // External Event for the dialog to use (to post requests)
            ExternalEvent exEvent = ExternalEvent.Create(handler);

            // We give the objects to the new dialog;
            // The dialog becomes the owner responsible for disposing them, eventually.
            m_MyForm = new ExternalEventExampleDialog(exEvent, handler);
            m_MyForm.Show();
        }
    }
}

[Autodesk.Revit.Attributes.Transaction(Autodesk.Revit.Attributes.TransactionMode.Manual)]
[Autodesk.Revit.Attributes.Regeneration(Autodesk.Revit.Attributes.RegenerationOption.Manual)]
public class Command : IExternalCommand
{
    public virtual Result Execute(ExternalCommandData commandData, ref string message, ElementSet elements)
    {
        try
        {
            ExternalEventExampleApp.thisApp.ShowForm(commandData.Application);
            return Result.Succeeded;
        }
        catch (Exception ex)
        {
            message = ex.Message;
            return Result.Failed;
        }
    }
}
```