

Revision Analyser

Technical Background

29th of June, 2011

Written by:
Johan van der Geest
Mark Ettema

University of Groningen

Contents

- 1 Project plug-in system..... 3**
 - 1.1 Auto-load system..... 3
 - 1.2 Accessing the project types 3
 - 1.3 Project class 4
- 2 Task system 5**
 - 2.1 Task class 5
 - 2.2 Starting a task batch 5
- 3 Global variables 6**
 - 3.1 Getting a global variable..... 6
 - 3.2 Setting a global variable manually..... 6
 - 3.3 Getting or deleting (all) variables 6
- 4 RevisionSet control 7**
 - 4.1 Revision class 7
 - 4.2 RevisionSet properties..... 7
 - 4.3 RevisionSet methods 7
 - 4.4 Updated event listener 7
- 5 RevisionSlider control 8**
 - 5.1 Attaching a RevisionSet 8
 - 5.2 RevisionSelected event listener..... 8
- 6 SolidSX control..... 9**
 - 6.1 SolidSX methods 9
 - 6.2 Loaded event listener 9

1 Project plug-in system

One great feature of Revision Analyser is its modular project plug-in system. It allows developers to create new project types for Revision Analyser and ship them as a DLL file to customers.

1.1 Auto-load system

On startup, Revision Analyser will scan the “Plug-ins” sub-folder for DLL files that end with “Project.dll”. Using .NET’s assembly features, a project DLL file will automatically be loaded into the memory and will get access to the global namespaces of Revision Analyser. During run-time, the application can show a list of loaded plug-ins via the About dialog (Figure 1).

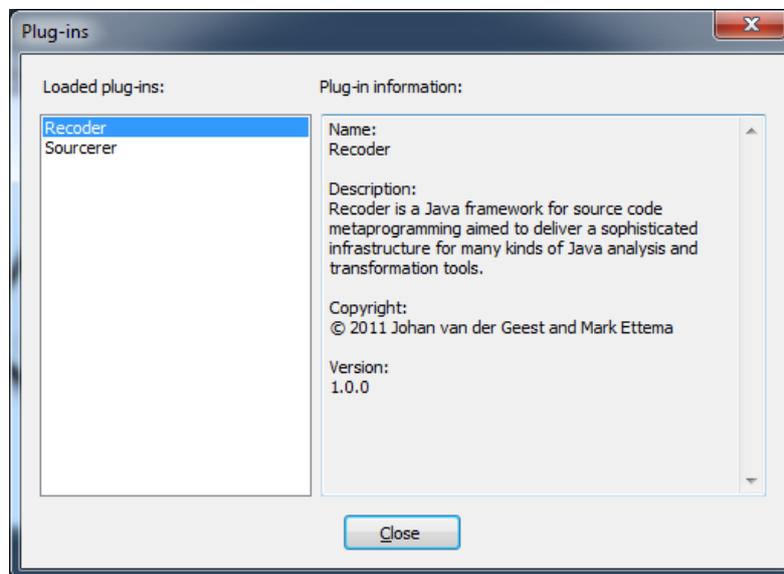


Figure 1: Plug-ins dialog.

1.2 Accessing the project types

The `RevisionAnalyser.Global.ProjectTypes` class contains a list of the project types that are currently loaded in the system. You can access it anywhere within Revision Analyser using the following code:

```
using RevisionAnalyser.Global.ProjectTypes;  
List<ProjectType> types = ProjectTypes.Instance.GetProjectTypes();  
ProjectType type = ProjectTypes.Instance.GetType("Example");
```

You can then create an instance of that project by calling the **GetProjectInstance** method on the `ProjectType` class:

```
Project project = type.GetProjectInstance();
```

1.3 Project class

The Project class is an abstract class that contains the following abstract methods:

- **FirstRun:** this method will be called on the first run of a project.
- **Opened:** each time the project is opened, this method will be called (also when the project is opened for the first time).
- **InitMenu:** in this method you can initialize the Project menu. You will get a reference to the ToolStripMenuItem object of the Project menu, such that you can easily add menu items to it.

It also contains the following properties:

- **ProjectFolder:** the location of the project folder.
- **ProjectFile:** the location of the project XML file (.raproj extension).
- **XmlDoc:** reference to the XmlDocument object for the project XML file.
- **RevisionSet:** reference to the RevisionSet object (see chapter 4 for more information).

2 Task system

Revision Analyser has an advanced task system that allows developers to run (time-consuming) tasks in a separate thread from the main thread. This allows you to continuously inform the end user about the progress while the task is performed.

2.1 Task class

The Task class is an abstract class that contains just one abstract method, named **Run**. This method will be called when the task is going to be executed. When the task is finished, you should call the **TaskFinished** method, otherwise the task will never finish. It is a good practice to have the following structure in your Run() method, to make sure the task will always finish:

```
public override void Run()
{
    try
    {
        // Normal execution code here
    }
    catch (Exception ex)
    {
        // Handle exceptions here
    }
    finally
    {
        TaskFinished();
    }
}
```

You should use the **AddLog** method of the Task class to add a message to the log. Another method that is available is **InsertTask**, which allows you to add a task to the task batch on-the-fly.

2.2 Starting a task batch

You can start a batch with tasks (or just one task) by passing a list with Task objects to the constructor of the TasksForm class, like shown below:

```
TasksForm frmTasks = new TasksForm(
    new List<Task>
    {
        new TaskA(),
        new TaskB()
    },
    false
);
```

3 Global variables

Revision Analyser can share global variables between projects, such as the paths of the Java Runtime Environment and SolidSX2. This way, you do not have to store these variables in individual projects. The user manual of Revision Analyser explains how this functionality works for the end user.

3.1 Getting a global variable

You can get the value of a global variable with the following code:

```
string var = GlobalVariables.Instance.GetVariable("KEY");
```

If the global variable does not exist, a form will popup for the end user that allows him or her to set the variable. Setting the variable is mandatory, there is no way to close this dialog for the end user without setting the variable.

You can also pass a second parameter to the **GetVariable** method containing a default value to show in the form when a variable has not been set before.

3.2 Setting a global variable manually

You can set a global variable manually with the use of the following code:

```
GlobalVariables.Instance.SetVariable("KEY", "value");
```

It is not mandatory to pass the key as upper-case string. The **SetVariable** method will transform the key to upper-case by default.

3.3 Getting or deleting (all) variables

The **GetVariables** method will return a <String, String> Dictionary containing all global variables that are currently set.

The **DeleteVariable** method will remove the global variable that you pass in its first parameter. The **DeleteVariables** method will remove all global variables.

4 RevisionSet control

The RevisionSet control offers a neat way for storing details about revisions within the Revision Analyser application. This could be any kind of revision, such as CVS, SVN or GIT.

4.1 Revision class

The RevisionSet control stores Revision objects. Before we start to discuss the functionalities of the RevisionSet control, we first want to introduce to you the Revision class.

The Revision class is a very simple abstract class that only stores these two properties:

- **ID:** a unique ID for the revision (a long / 64-bit signed integer).
- **Time:** a DateTime object that tells when the revision was committed.

As a developer for a custom project type, you must override the Revision class. You can then add your own custom properties. For example, the Recoder project type adds the Author and LogMessage properties in its own SvnRevision class.

4.2 RevisionSet properties

The RevisionSet control contains the following properties:

- **FirstRevision:** the first revision that is stored in the set. Returns 0 if the set is empty.
- **LastRevision:** the last revision that is stored in the set. Returns 0 if the set is empty.
- **RevisionCount:** the number of revisions stored in the set.

4.3 RevisionSet methods

The RevisionSet control contains the following methods:

- **AddRevision:** adds a Revision object to the set. You can optionally pass a *true* as second parameter to force the Updated event to be fired (see the next paragraph).
- **GetRevision:** returns the Revision object for a given revision ID.
- **GetRevisionDictionary:** returns a Dictionary containing all revision ID's and objects.
- **GetRevisionList:** returns a List containing just the Revision objects.
- **ContainsRevision:** checks whether the specified revision ID exists in the set.
- **DeleteRevision:** deletes a specified revision ID from the set. You can optionally pass a *true* as second parameter to force the Updated even to be fired.
- **ClearRevisions:** removes all revisions from the set.
- **Update:** fires the Updated event.

4.4 Updated event listener

You can attach an event listener to **Updated** to get notified of changes to the set.

5 RevisionSlider control

The RevisionSlider is an advanced GDI+ control that can be linked to a RevisionSet to display its revisions on a timeline.

5.1 Attaching a RevisionSet

In order to make the RevisionSlider functional, you must attach a RevisionSet to it. You can do this through code (by changing the RevisionSet property), or through the designer in the Visual Studio IDE, as can be seen in Figure 2.

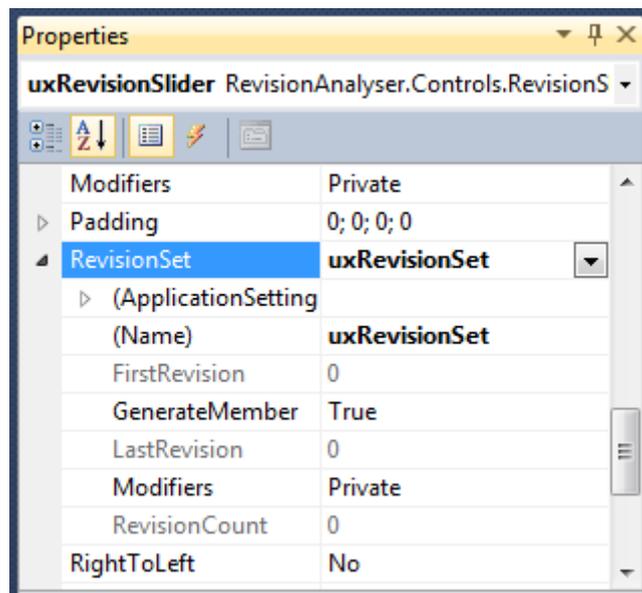


Figure 2: Attaching a RevisionSet through the designer.

The RevisionSlider will add an event listener to the RevisionSet control after assigning it, so that it will automatically update itself whenever a revision is added or deleted.

5.2 RevisionSelected event listener

The RevisionSelected even listener allows you to get notified whenever an user selects another revision in the timeline. Revision Analyser uses this event listener to update the SolidSX2 control after an user selects a new revision.

6 SolidSX control

The SolidSX control is a control that can automatically launch SolidSource SolidSX2, capture the required window handles, and render it flawlessly within Revision Analyser. This allows us to make use of the great external visualization tool SolidSX, while still offering good usability by integrating it within Revision Analyser.

6.1 SolidSX methods

The SolidSX control has the following public methods:

- **OpenSolidSX**: launches SolidSX and opens the given database file.
- **CloseSolidSX**: closes SolidSX.
- **SelectRevision**: selects the given revision in SolidSX.
- **ExpandAll**: expands all nodes in SolidSX.
- **CollapseAll**: collapses all nodes in SolidSX.
- **ExpandLevel**: expands to a given level in SolidSX (for example: 0, 1, 2, etc.).

6.2 Loaded event listener

You can attach an event listener to **Loaded** to get notified when SolidSX is finished loading.