RadioFlex - Editor

--- **JunLeeTerry**

1.RadioflexEditor 2

2.RESourceViewerConfiguration 4

3.RColorProvider 5

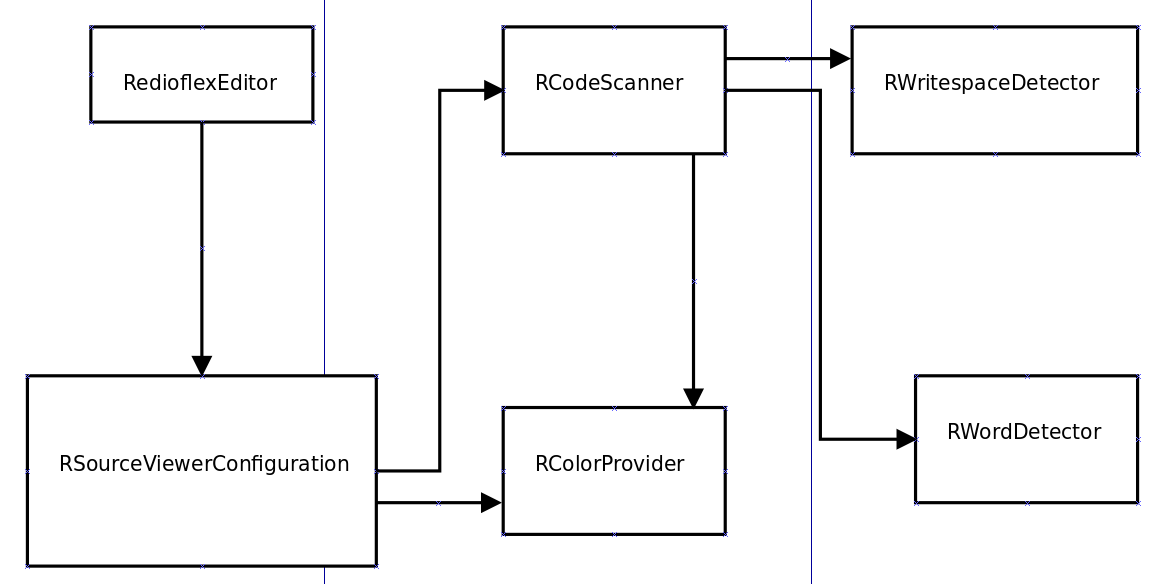
4.RCodeScanner 6

5.RWhitespaceDetetor 7

6.RWordDetector 7

In order to achieve a syntax highlighting for editor we need to create some class.

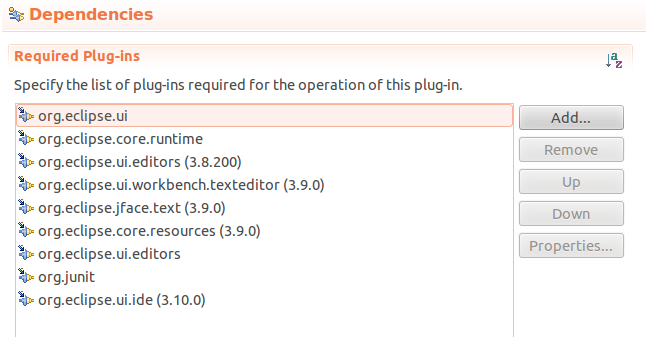
RedioflexEditor,RSourceViewerConfiguration,RCodeScanner,RColorProveder,RWritespaceDetector,RWordDetector.Picture 1.0 shows their call relationship.



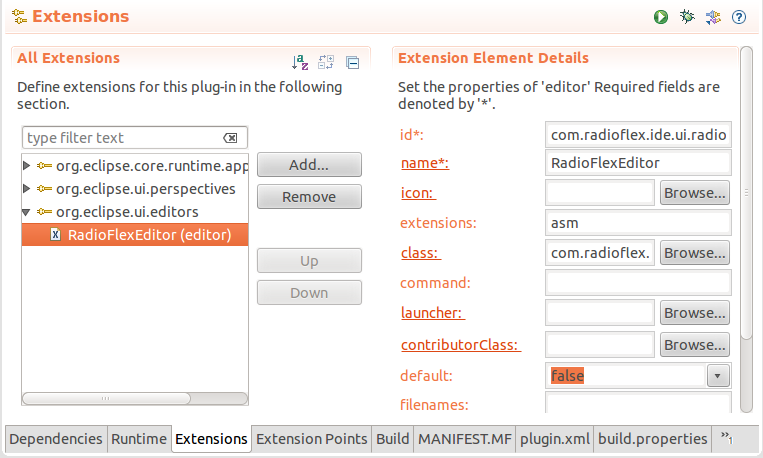
1.0

**1.RadioflexEditor**

1.1 First,some dependencies we need to add.



1.2 we need to add editor extensions.



1.3 create RadioflexEditor.java

|\_ EditorPart

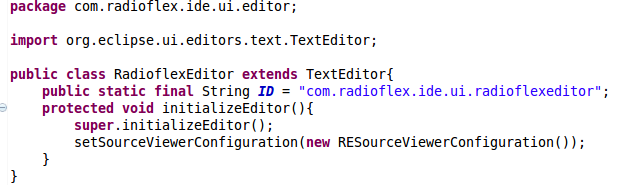
|\_ AbstractTextEditor

|\_ StatusTextEditor

|\_ AbstractDecoratedTextEditor

|\_ TextEditor

|\_ RadioflexEditor



initlizes this editor.This method configures the editor but does not define a SourceViewerConfiguration. When only interested in providing a custom source viewer configuration, subclasses may extend this method.

We want to realize syntax highlighting ,we must configurating a source viewer.So we must override the method initealizeEditor().Use setSourceViewerConfiguration().Its para-meter is the instantiation of the class RESourceViewConfiguration

The RadioflexEditor calls the RESourceViewerConfiguration.

**2.RESourceViewerConfiguration**

2.1 If you get ready to realize syntax hightlighting you need to know reconciler model before coding.

As the user modifies text in an editor, parts of the editor must be redisplayed to show the changes.  Computing the text that must be redisplayed is known as computing **damage**.  When syntax coloring is involved, the amount of damage caused by an editing operation becomes more extensive, since the presence or absence of a single character could change the coloring of the text around it.

Damagers ([**IPresentationDamager**](http://help.eclipse.org/luna/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/presentation/IPresentationDamager.html)) determine the region of a document's presentation which must be rebuilt because of a document change. A presentation damager is assumed to be specific to a particular document content type (or region). It must be able to return a damage region that is valid input for a presentation repairer ([**IPresentationRepairer**](http://help.eclipse.org/luna/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/presentation/IPresentationRepairer.html)).  A repairer must be able to derive all of the information it needs from a damage region in order to successfully describe the **repairs** that are needed for a particular content type.

**Reconciling** describes the overall process of maintaining the presentation of a document as changes are made in the editor.  A presentation reconciler ([**IPresentationReconciler**](http://help.eclipse.org/luna/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/presentation/IPresentationReconciler.html)) monitors changes to the text through its associated viewer.  It uses the document's regions to determine the content types affected by the change and notifies a damager that is appropriate for the affected content type.  Once the damage is computed, it is passed to the appropriate repairer which will construct repair descriptions that are applied to the viewer to put it back in sync with the underlying content.

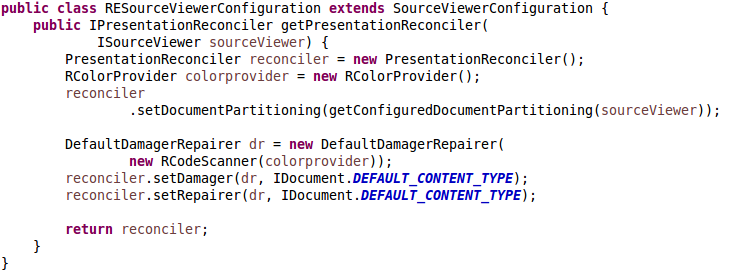
The classes in [**org.eclipse.jface.text.reconciler**](http://help.eclipse.org/luna/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/reconciler/package-summary.html) define additional support classes for synchronizing a document model with external manipulation of the document.

Presentation reconcilers should be provided with a repairer and damager pair for each content type to be found in the document.  It is up to each editor to determine the appropriate implementation for a presentation reconciler.  However, the platform provides support in[**org.eclipse.jface.text.rules**](http://help.eclipse.org/luna/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/rules/package-summary.html) for using rule-based document scanners to compute and repair damage.  Default damagers and repairers are defined in this package.  They can be used along with the standard reconcilers in [**org.eclipse.jface.text.presentation**](http://help.eclipse.org/luna/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/text/presentation/package-summary.html) to implement syntaxcoloring by defining scanning rules for the document.

2.2 create RESourceViewerConfiguration

|\_ SourveViewerConfiguration

|\_ RESourveViewerConfiguration



Override the method getPresentationReconciler.The method returns the presentation reconciler ready to be used with the given source viewer.

So first we new PresentationReconciler(); The class RcolorProvider is Where we place the color information.

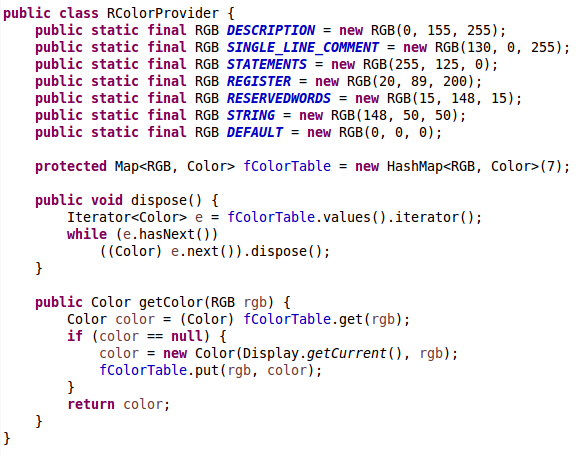
Then,set document partitioning.

Finially,add the damager-repairer-reconciler model.New a DefaultDamagerRepairer.Its parameter is RCodeScanner another class we created.A presentation reconciler provided with a repairer and a damager pair.Set damager and repairer.Their first parameter is I

DefaultDamagerRepairer,second parameter is contentType(String).we use Idocument.DEFAULT\_CONTENT\_TYPE as contentType.That means The identifier of the default partition content type.

**3.RColorProvider**

This is a place where we dentify the imformation of the color we will use.The class doesn't have parent class.



In assembly language,there are six kinds of words to color.they are description,single line comment,statements,register,reserved words and string.So we need seven colors,the other color is 'default'.

In the class,we store the RGB and the corresponding color in Map<RGB,Color> fColorTable.We also have method getColor() to get Color of RGB.

**4.RCodeScanner**

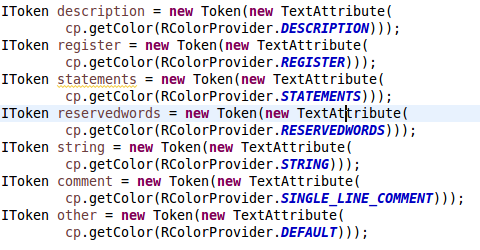
|\_ RuleBasedScanner

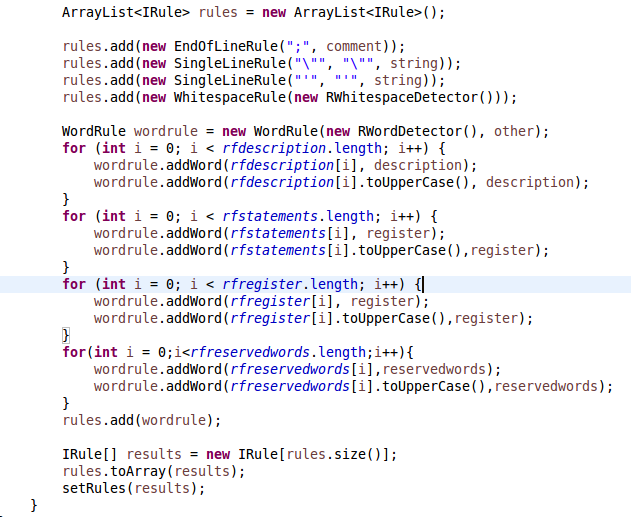
|\_ RCodeScanner

4.1 Define the keywords in String array like the next picture.

A description...

4.2 In the constructor,define the Itoken.

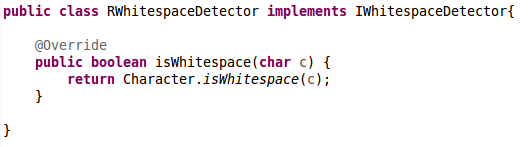
The Token's parameter is TextAttribute(*Creates a text attribute for the given foreground color, no background color and with the SWT normal style.*) .Its parameter is Color convert form the RGB we defined in the class RcolorProvider.

4.3 

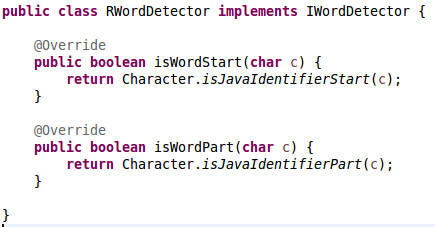
Define the ArrayList to store Irule.(*Defines the interface for a rule used in the scanning of text for the purpose of document partitioning or text styling.*)The EndLineRule is the rule for the given starting sequence which, if detected, will return the specified token.The SingleLineRule is the rule for the given starting and ending sequence which,if detected,will return the specified token.And the WhitespaceRule need writespace detector,and we need create it.The WordRule is an implementation of IRule capable of detecting words. A word rule also allows to associate a token to a word. That is, not only can the rule be used to provide tokens for exact matches, but also for the generalized notion of a word in the context in which it is used. A word rule uses a word detector to determine what a word is.

We add words and their token into the wordrule.

**5.RWhitespaceDetector**

The class is implements IwhitespaceDetector and realize our whitespace detector.Override the method isWhitespace,return Character.isWhitespace.

**6.RWordDetector**

The class RwordDetector is similar with the class RwhiteDetector.

Override the method isWordStart and isWordPart.