Jan 30 2015 pyqgis qgis

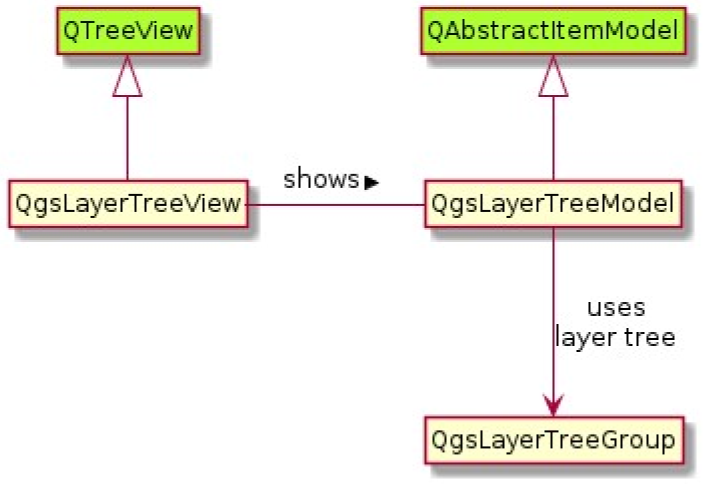
Recent blog posts

QGIS Layer Tree API (Part 3)

In the two previous blog posts we have learned how to query a layer tree and how to modify it. All these new APIs are available since QGIS 2.4 release. Today we will look into how to make layer trees available in GUI and connect them with a map canvas.

Model/View Overview

As we have seen earlier, a layer tree is a usual hierarchical data structure composed from nodes of two types – layers and groups. They do not provide any GUI functionality as they live in the QGIS core library. In order to visualize a layer tree, we will use Model/View approach from Qt framework. Readers not familiar with those concepts are recommended to read the Qt Model/View overview first.



There is `QgsLayerTreeModel` class (derived from `QAbstractItemModel`) which as you may have guessed provides a model to access a layer tree. Instance of this class may be used in any `QTreeView` class, however it is recommended to use it together with `QgsLayerTreeView` because of the extra convenience functionality offerred by the custom view class.

Here is an example how to create another view of current project's layer tree (try that in QGIS Python console):

```
from qgis.gui import *

root = QgsProject.instance().layerTreeRoot()
model = QgsLayerTreeModel(root)
view = QgsLayerTreeView()
view.setModel(model)
view.show()
```

Any changes that happen to the layer tree structure are automatically monitored by the model/view classes. After running the example above, try changing a layer's name or add/remove/reorder some layers – all those actions will be immediately visible in all views.

As you can see, the layer tree view is just one way how to visualize the underlying layer tree – in the future it may be possible to have different ways to show the layer tree, for example using Qt's QML framework (with all sorts of animated transitions known from mobile apps).

Plugin developers can access the layer tree view in the main window of QGIS through the following interface call:

```
view = iface.layerTreeView()
```

More About the Model

The model is meant to be flexible and it is possible to use it in various contexts. For example, by default the model also provides legend for the layers in the tree. This however may not be wanted in some cases. Similarly, sometimes the layer tree should be read-only while in other context it is desired that the user can reorder layers or change their names. These preferences can be passed to the model with flags:

```
model = QgsLayerTreeModel(root)
```

```
model.setFlag(QgsLayerTreeModel.AllowNodeReorder)
model.setFlag(QgsLayerTreeModel.AllowNodeChangeVisibility)
```

The `setFlag()` method has optional second parameter "enabled" (`True` by default). Flags can be also set all at once with `setFlags()` method which expects a combination of flags joined by binary OR operator. There are also `flags()` and `testFlag()` methods to query the current flags.

The `QgsLayerTreeModel` class also provides routines for conversion between `QgsLayerTreeNode` instances and corresponding `QModelIndex` objects used by Qt Model/View framework – `index2node()` and `node2index()` may come handy especially when working with views.

There is also some functionality related to legend display – it has been greatly extended in QGIS 2.6 release and we will try to cover that in a future blog post.

More About the View

As mentioned earlier, it is possible to use any `QTreeView` instance in combination with `QgsLayerTreeModel` to show the layer tree, but it is highly recommended to use `QgsLayerTreeView` class (a subclass of `QTreeView`) because of the additional functionality it provides (and more may be added in the future):

- Easier handling of selection. Normally one needs to work with selection through view's selection model. The `QgsLayerTreeView` class adds higher level methods that operate with `QgsLayerTreeNode` objects like `currentNode()` or with `QgsMapLayer` objects like `currentLayer()`.

```
def onChange(layer):
  QMessageBox.information(None, "Change", "Current Layer: "+str(layer))

# connect to the signal
view.currentLayerChanged.connect(onChange)
# change selection to the top-most layer (onChange will be also called)
view.setCurrentLayer( iface.mapCanvas().layers()[0] )
```

- Display of context menu. It is possible to assign a menu provider to the view (subclass of `QgsLayerTreeViewMenuProvider`) – its `createContextMenu()` implementation will return a `QMenu` object with custom actions whenever user right-clicks in the view. Additionally, there is a factory class `QgsLayerTreeViewDefaultActions` that can create commonly use actions for use in the menu, such as "Add Group", "Remove" or "Zoom to Layer". The following example shows how to create a provider with one action that shows the current layer's extent:

```
class MyMenuProvider(QgsLayerTreeViewMenuProvider):
  def __init__(self, view):
    QgsLayerTreeViewMenuProvider.__init__(self)
    self.view = view

  def createContextMenu(self):
    if not self.view.currentLayer():
      return None
    m = QMenu()
    m.addAction("Show Extent", self.showExtent)
    return m

  def showExtent(self):
    r = self.view.currentLayer().extent()
    QMessageBox.information(None, "Extent", r.toString())
provider = MyMenuProvider(view)
view.setMenuProvider(provider)
```

Interaction with Canvas

The layer tree classes and map canvas class are separate components that are not dependent on each other. This is a good thing and a great step forward, because until QGIS 2.2 there was big internal monolithic `QgsLegend` view class that handled everything and it was directly connected to map canvas and various other components, making it impossible to reuse it elsewhere. With the new layer tree API this has been solved, now it is possible use map canvas without an associated layer tree view or vice versa – to use a layer tree view without map canvas. It is even possible to get creative and use one layer tree with several map canvas instances at once.

Layer tree and map canvas can be connected via `QgsLayerTreeMapCanvasBridge` class. It listens to signals from the given layer tree hierarchy and updates canvas accordingly. Let's see how we could create a new map canvas that would show the same layers as the main canvas does:

```
canvas = QgsMapCanvas()
root = QgsProject.instance().layerTreeRoot()
bridge = QgsLayerTreeMapCanvasBridge(root, canvas)
canvas.zoomToFullExtent()
canvas.show()
```

That's it! We have tied the new canvas with project's layer tree. So any actions you do in the layer tree view (for example, add or remove layers, enable or disable layers) are automatically passed to the new canvas. And of course this does not work just with project's layer tree – you could use any custom layer tree in your layer tree model/view or canvas.

The bridge class has advanced functionality worth mentioning. By default the ordering of layers in canvas is according to the order in the layer tree (with first layer in the tree being at the top), though there are API methods to override the default order.

For convenience, the bridge by default also configures some settings of the canvas (this can be disabled if necessary):

- enable on-the-fly reprojections if layers have different coordinate reference system (CRS)
- setup destination CRS and map units when first layers are added
- zoom to full extent when first layers are added

Summary

I hope you have enjoyed the three blog posts introducing QGIS layer tree API. They should cover everything you need to know in order to start using the new functionality. In a future post we will have a look at the new legend API that nicely complements the layer tree API – stay tuned!

Posted by Martin Dobias

Tweet G+1 2 Like Share One person likes this. Be the first of your friends.