HTML Viewer

Contents

1 SoXipHtmlViewer 2
  1.1 Objective 2
  1.2 Core 2
  1.3 Node 3
  1.4 HTML and JavaScript 4
2 SoXipIvQtUiViewer 5
  2.1 Objective 5
  2.2 Core 5
  2.3 Node 6
  2.4 UI and Qt Script files 6
3 Standalone application 9

1 SoXipHtmlViewer

1.1 Objective

The HTML viewer allows loading a render area of a scene graph into any customized HTML file and manipulating the scene graph through HTML controls and JavaScript functions. Main objective in implementing this node was to bring the functionality of the SoRadHtmlView node formerly used in RadBuilder into XIP Builder. Instead of MFC and IE6, this approach uses QtWebKit which provides its own web browser based on the Open Source Webkit engine (also used by Apple's Safari web browser).

It can be used as a node in XipBuilder as well as in a standalone application.

1.2 Core

XipHtmlViewerCore.lib contains the whole functionality for the HTML viewer. Both the standalone application and the htmlviewer.dll link against it statically.

QXipIvHtmlWidget is the render area. It is basically identical to QXipBuilderIvWidget in the XIP builder which is responsible for drawing the scene graph and handling events.

QXipHtmlIvWrapperWidget is parent object of the QXipHtmlIvWidget object. In order to display an object of this class in the HTML page it has to be registered in `xiphtmlviewer_init()` with `qRegisterMetaType<QXipHtmlIvWrapperWidget>()`.

QXipHtmlJSInterfaceWidget is responsible for the interaction between JavaScript and the scene graph. In class QXipWebView an object of this class ("widget") is exposed to Javascript, from which the slots `getParam(QString string)` and `setParam(QString string)` are called to find the specified node's field and either set or return its value. `setParam(QString string)` can also be used to add or remove a sensor to/from a node or add a child to a node.

- `setParam("nodeName.fieldName,value")`
  sets the value of a node's field.

- `setParam(”AddFieldSensor,nodeName.fieldName”)`
  adds a sensor to a node’s field.
- `setParam(”RemoveFieldSensor,nodeName.fieldName”)`
  removes the sensor from a field.
- `setParam(”AddChild,parentNodeName.childNodeName”)`
  adds a child to a node.

The signal `stateChanged(const QString &value, const QString &nodeFieldName)` can be connected to any JavaScript function and is emitted whenever a field value has changed, given that a sensor has been added to the field. The slot `connectJSSlot` can be called to connect this signal to a function in JavaScript.

QXipWebPage and QXipWebView are derived from QtWebKit’s classes QWebPage and QWebView. QWebView acts as a view onto a web page, represented by an instance of QWebPage, which provides access to the document structure in a page. The class QWebFrame includes the bridge to the JavaScript window object. Every QWebPage object has at least one object of QWebFrame as its main frame.

- **QXipWebPage** handles the plugin creation. In `createPlugin(...)` the class name of the object that will be embedded in the HTML page is specified.
- **QXipWebView** connects the signal `javaScriptWindowObjectCleared()` with the slot `addToJavaScriptWindowObject(”widget”, mQtObject)`, which exposes an object of QXipHtmlJSInterfaceWidget to JavaScript. `javaScriptWindowObjectCleared()` is emitted whenever the global window object of the JavaScript environment is cleared (e.g. before reloading the page).

1.3 Node

SoXipHtmlViewer is the actual node. In order to load it in XipBuilder, it has to be specified in the extensions.xml. It has the fields `file`, `width`, `height`, a trigger `show` and a member variable `mView` of QXipWebView. On show it will popup a window and load the specified `file` into the web view. The size of the window is indicated by `width` and `height`. There can be multiple SoXipHtmlViewer nodes in a scene graph. Regardless of where in the scene graph the node is located, it will always render its subgraph. If it has multiple children a Separator will be inserted internally inbetween the SoXipHtmlViewer node and its children and set as root node.

1.4 HTML and JavaScript

Embedding the render area
The render area can be embedded with the following code:

```
<object type="application/x-qt-plugin"
    classid="QXipHtmlIvWrapperWidget" name="renderer"
    height="100\%" width="100\%"></object>
```

Accessing scene graph
In QXipWebView an object of QXipHtmlJSInterfaceWidget was exposed to JavaScript, ”widget”. In order to interact with the scene graph we need to call its slots `getParam` and `setParam`:

```
function setParam(p, v)
{
        widget.setParam(p + ',' + v);
}

function getParam(param)
{
        return widget.getParam(param);
}
```

These functions need to be defined in the main HTML file which is specified in the `file` field of SoXipHtmlViewer.

JavaScript callback
As mentioned above, QXipHtmlJSInterfaceWidget has a signal `stateChanged-(const QString &value, const QString &nodeFieldName)` that is emitted when the value of a (sensor-controlled) field has changed. The signal passes the name of the node,the name of the field and its new value on to the JavaScript function that is connected to it. To connect the signal to a function, use

```
widget.connectJSSlot('stateChanged(const QString &, const
    QString &)', 'window.onStateChanged');

window.onStateChanged = function onStateChanged(v, p)
{
        if (p == 'NodeName.FieldName')
        {
                ...
        }
}
```

HTML adjustments for standalone viewer
To tell the standalone viewer which .iv file and possible additional dlls to load, a comment needs to be added in the HTML file:

```
<!--LoadIVFile scenegraph.iv-->
<!--LoadDLL xipivcorepro,xipivutils,xipivvtk-->
<!--LoadDLLConfigFile extensions.xml-->
```

2 SoXipIvQtUiViewer

2.1 Objective

Objective is to make Xip capable of wrapping a GUI around a scene graph to display it and interact with it through Qt controls. Qt's QUiLoader allows creating interfaces during runtime using the information stored in .ui files which can easily be created with the QtDesigner. Qt Script is used to connect the controls and access the scene graph.

2.2 Core

XipIvQtUiViewerCore.lib contains the functionality that is provided to SoXipIvQtUiViewerNode and the standalone viewer, which both link against it statically.

QXipIvWidget is the render area. It is basically identical to QXipBuilderIvWidget in the XIP builder which is responsible for drawing the scene graph and handling events.

QXipIvWrapperWidget is parent object of the QXipIvWidget object. It contains the slots `getParam(QString string)` and `setParam(QString string)` that are called from Qt Script to find the specified node's field and either set or return its value. `setParam(QString string)` can also be used to add or remove a sensor to/from a node or add a child to a node.

- `setParam(''nodeName.fieldName,value'')`
  sets the value of a node's field.

- `setParam(''AddFieldSensor,nodeName.fieldName'')`
  adds a sensor to a node's field.

- `setParam(''RemoveFieldSensor,nodeName.fieldName'')`
  removes the sensor from a field.

- `setParam(”AddChild,parentNodeName.childNodeName”)`
  adds a child to a node.

The signal `stateChanged(const QString &value, const QString &nodeFieldName)` can be connected to any Qt Script function and is emitted whenever a field value has changed, given that a sensor has been added to the field.

QXipIvUiInterface evaluates the script files and creates the widget from the .ui file. When creating the UI elements, the widget with the classname "QXipIvRenderWidget" is caught and replaced with an object of "QXipIvWrapperWidget".

2.3 Node

SoXipIvQtUiViewer is the name of the actual node. In order to load it in XipBuilder it has to be listed in the extensions.xml. It has 3 String fields, `ui`, `script` and `scriptExtensions`, and a trigger `show`. If you wish to load any script extensions, you can enter their names in the `scriptExtensions` field. `script` and `scriptExtensions` are multi fields, so it is possible to load multiple script files. If the node has any children, they are rendered in the QXipWrapperWidget. Otherwise it will only provide access to the scene graph.

2.4 UI and Qt Script files

Creating UI files with QtDesigner

In order to be able to load the scenegraph and interact with it, it is necessary to add a widget to the .ui file and promote it to "QXipIvRenderWidget" with the objectname "widget". It will later be replaced with on object of "QXipIvWrapperWidget". Even if you do not want to display the scene graph in the GUI, it is important to promote this widget, to have access to the scenegraph. If the actual node in the scene graph has no children, the widget will not be displayed, neither will it take up any space in the GUI.

Qt Script

A basic introduction to Qt Script can be found at `http://doc.trolltech.com/4.4/qtscript.html`.

There are two ways to interact with the scenegraph. One would be to call the slots `setParam` and `getParam` like in the HTML viewer:

```
Form.prototype.setParam = function(p,v)
{
    this.ui.widget.setParam(p+','+v);
}

Form.prototype.getParam = function(p)
{
    return this.ui.widget.getParam(p);
}
```

Another way would be to call the following slots to recieve pointers the fields and nodes:

```
node = this.ui.widget.getNode('WindowLevelManip');

field = this.ui.widget.getField('WindowLevelManip', 'level');

trigger = this.ui.widget.getTrigger('Popup', 'show');
```

To get or set a value of a field we can call the slots `field.getCurStr()` and `field.setStr(value)`. To get a nodes name call `node.getName()` and to trigger a trigger field call `trigger.trigger()`.

The syntax for connecting to and disconnecting from signals in Qt Script is different from the C++ syntax. To connect a signal, you reference the relevant signal as a property of the sender object, and invoke its connect() function.

So instead of

```
connect(obj1, SIGNAL(somethingChanged()), obj2,
   SLOT(doSomething()));
```

in Qt Script we write

```
obj1.somethingChanged.connect(obj2, "doSomething");
```

To connect the widget's Qt signal `stateChanged` to a script function, you write

```
this.ui.widget["stateChanged(QString,QString)"].connect(this,
    "stateChanged");
```

or

```
field["stateChanged(QString,QString)"].connect(this,
    "stateChanged");
```

and define the function `stateChanged(v,p)` in Qt script:

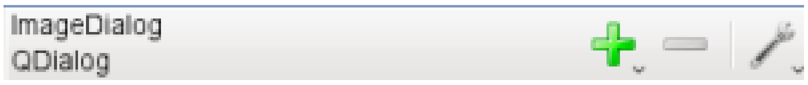
```
Form.prototype.stateChanged = function(v, p)
{

    if(p == 'RenderArea.current'){
        ...
    }
    if(p == 'WindowLevelManip.window'){
        ...
    }
    if(p == 'WindowLevelManip.level'){
        ...
    }
}
```

Adjustments for Standalone viewer
To tell the standalone viewer which IV file and script files to load, we add property tags called "ivFile" and "scriptFiles" to the main widget in the ui file:

```
<property name="ivFile" stdset="0">
   <string>mpr_multiple_scripts.iv</string>
</property>
<property name="scriptFiles" stdset="0">
   <string>ui/renderWidget_1.js, ui/renderWidget_2.js</string>
</property>
```

Dynamic properties can be added in Qt Designer by clicking the plus icon in the property editor tool bar and choosing "String".



You can make additional functionality available to the scripts by using Qt Script extensions. Information about how to write your own extensions can be found at `http://doc.trolltech.com/4.5/qtscriptextensions.html`. In order to load the extensions enter their names in the scriptExtensions field.

The Qt Script Generator (`http://labs.trolltech.com/page/Projects/QtScript/Generator`) provides extensions to access substantial portions of the Qt API. Note that the "qt.core" extension apparently does not expose the function QObject::findChild(QString name) to the script.

3 Standalone application

The standalone application views the scene graph in a GUI defined by either an HTML or a UI file, that is given as an argument by running Standaloneviewer.exe. If it is run without an argument it will try to find and open "html/main.html" by default.

Besides the dlls that are specified in the extensions.xml additional dlls can be loaded, either by providing their single names or an xml file (like extensions.xml) inside the HTML file.

Also in the HTML or UI file, the scene graph which is to be loaded, has to be specified. The standalone application can only handle .iv files. The UI files also have to specify the Qt Script file(s) that ought to be loaded.

The scene graph itself does not need to contain a SoXipHtmlViewer or SoXipIvQtUiViewer node as it wrapps the functionality of the core of either one around the whole scene graph.