How-to create a persistent settings database in Qt Quick (QML)

This article shows how to store your persistent QML application settings using the Offline Storage API.

02 Nov 2011

How-to create a persistent settings database in Qt Quick (QML)

As applications are opened and closed, it is often necessary to keep a few settings or data persistent, for example usernames and passwords, or simply configuration values. Qt Quick (QML) does not (yet?) have a way to access local files (this is doable through a plugin for example), but offers the Offline Storage API. This storage uses Sqlite databases to store data to disk.

Example Code

Here is an example on how to use this Offline Storage API (see QML Global Object) to create a simple settings database. Each setting is composed of a key (represented as a string) and a value (also a string).

Let's create a small javascript file containing the functions necessary to store and retrieve our settings. This file can then be imported in our QML documents to be used. We will have 3 main functions: initialize() to set up the necessary tables, setSetting(setting,value) to record a setting in the database and getSetting(setting) to retrieve a setting's value.

```
//storage.js
// First, let's create a short helper function to get the database connection
function getDatabase() {
     return openDatabaseSync("MyAppName", "1.0", "StorageDatabase", 100000);
}

// At the start of the application, we can initialize the tables we need if they haven't been created yet
function initialize() {
    var db = getDatabase();
    db.transaction(
        function(tx) {
            // Create the settings table if it doesn't already exist
            // If the table exists, this is skipped
            tx.executeSql('CREATE TABLE IF NOT EXISTS settings(setting TEXT UNIQUE, value TEXT)');
   });
}

// This function is used to write a setting into the database
function setSetting(setting, value) {
   // setting: string representing the setting name (eg: “username”)
   // value: string representing the value of the setting (eg: “myUsername”)
   var db = getDatabase();
   var res = "";
   db.transaction(function(tx) {
        var rs = tx.executeSql('INSERT OR REPLACE INTO settings VALUES (?,?);', [setting,value]);
              //console.log(rs.rowsAffected)
              if (rs.rowsAffected > 0) {
                res = "OK";
              } else {
                res = "Error";
              }
        }
  );
  // The function returns “OK” if it was successful, or “Error” if it wasn't
  return res;
}
// This function is used to retrieve a setting from the database
function getSetting(setting) {
   var db = getDatabase();
   var res="";
   db.transaction(function(tx) {
     var rs = tx.executeSql('SELECT value FROM settings WHERE setting=?;', [setting]);
     if (rs.rows.length > 0) {
          res = rs.rows.item(0).value;
     } else {
         res = "Unknown";
     }
  })
  // The function returns “Unknown” if the setting was not found in the database
  // For more advanced projects, this should probably be handled through error codes
  return res
}
```

Now, we can use this short library in our QML files. Here is a short example:

```
import Qt 4.7
import "storage.js" as Storage
Rectangle {
    width: 360
    height: 360
    id: screen
    Text {
        id: textDisplay
        anchors.centerIn: parent
    }
    Component.onCompleted: {
        // Initialize the database
        Storage.initialize();
        // Sets a value in the database
        Storage.setSetting("mySetting","myValue");
        // Sets the textDisplay element's text to the value we just set
```


(C) Copyright Nokia 2013. All rights reserved.

```
        textDisplay.text = "The value of mySetting is:\n" + Storage.getSetting("mySetting");
    }
}
```

--Slocan 23:40, 29 January 2011 (UTC)

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.