Public Support - Support Request #6949

Documentation for Filter Editor qml

2019-04-24 16:51 - hidden

Status:	Closed		
Priority:	Normal		
Category:			
Customer:	VW	**Product Issue Numbers:**	https://www.cip.audi.de/jira/browse/ACORE-8895
Department:		**Affected Products:**	ADTF 3.5.0
Requester's Priority:	Normal	**Platform:**	Windows 10 64bit
Support Level:	3rd Level	**Topic:**	ADTF::FilterSDK
Resolution:	Known Problem	**FAQ Links:**	

Description

Supportanfrage

ähnlich wie beim ADTFDAT File Player möchte ich über das Kontextmenü ("Create pins from playback file") eine Datei einlesen und Pins erzeugen. In der Datei "${ADTF_DIR}/bin/adtf_playback_input_filtereditor.qml" finde ich Funktionen wie:

```
createOutputPin()
listOutputPins()
```

Wo sehe ich, wie diese verwendet werden und welche Funktionen noch zur Verfügung stehen?

Lösung

Kommt zur 3.7

Fehlerbehebung:

```
qmlRegisterType<Example>(uri, 1, 0, "Example");
```

statt

```
qmlRegisterType<Example>("ExamplePlugin", 1, 0, "Example");
```

verwendet. Da die Variable "uri" leer war, konnte das Modul "ExamplePlugin" wohl nicht gefunden werden. Das Modul wird jetzt geladen und ich kann die Klasse in der Qml des Filter verwenden.

History

#1 - 2019-04-25 08:45 - hidden

- Status changed from New to In Progress

- Topic set to ADTF::FilterSDK

#2 - 2019-04-25 12:26 - hidden

- Project changed from 20 to Public Support

- Private changed from Yes to No

- Resolution set to Known Problem

- Product Issue Numbers set to https://www.cip.audi.de/jira/browse/ACORE-8895

- Support Level changed from 2nd Level to 3rd Level

Hallo Matthias,

ja, dazu haben wir ein Ticket (ACORE-8895), hierzu gibt es leider noch keine Doku, wir sind dran.
Wir werden versuchen, dir einen Guide zu schreiben, ich habe das Ticket noch einmal hochpriorisiert.

In der Zwischenzeit zum groben Doing (am Bsp. des Players):

1) In deiner Plugin Description den Filter Editor definieren

```
<editor_descriptions>
    <editor_description>
        <name>Create pins from playback file</name>
        <url>adtf_playback_input_filtereditor.qml</url>
    </editor_description>
</editor_descriptions>
```

2) In deiner Plugin Description dynamische Pins konfigurieren

```
<pin_set_description>
    <pin_descriptions/>
    <binding_object_descriptions/>
    <runner_descriptions/>
    <dynamic_input_pins>false</dynamic_input_pins>
    <dynamic_output_pins>true</dynamic_output_pins>
</pin_set_description>
```

3) In deinen Filter Editor an sich (QML) brauchst du folgende Basics

```
import QtQuick 2.7
import QtQuick.Controls 1.4
import QtQuick.Layouts 1.1
import QtGraphicalEffects 1.0
import QtQuick.Dialogs 1.2

// ...

import EditorPlugin 1.0

EditorPluginBase
{

    // ...

    createOutputPin("mypinname");

    // ...
}
```

4) Funktionen die du nutzen kannst:

```
// Inputpins
function findInputPin(name)
function listInputPins()
function createInputPin(name)
function removeInputPin(pin)

// Outputpins
function findOutputPin(name)
function listOutputPins()
function createOutputPin(name)
function removeOutputPin(pin)

// Properties
function findProperty(component, name)
function createProperty(component, name, value)
function setProperty(component, name, value)
function getPropertyValue(component, name)

// Vielleicht hilfreich
function resolveAndGetAbsoluteUrl(path, basePath)
function launch(command, args, workingDirectory)
```

Vorab-Hinweis: Punkt 1) und 2) wirst du ab ADTF 3.6.0 aus dem Code heraus definieren können

#3 - 2019-04-25 15:25 - hidden

- Status changed from In Progress to Customer Feedback Required

#4 - 2019-04-26 09:14 - hidden

Danke, das hilft weiter. Was ist "component" bei den Property-Funktionen?

Kann ich auch ein CE-Plugin erstellen, damit ich dann über QML auf C++ Klassen zugreifen kann (ähnlich wie hier https://doc.qt.io/qt-5/qtqml-cppintegration-topic.html)?

#5 - 2019-04-26 09:31 - hidden

Hallo Matthias,

> Was ist "component" bei den Property-Funktionen?

Die Plugin-Komponenten, also z.B. ein Filter, Service oder Streaming Source/Sink.
z.B.

```
import QtQuick 2.7
import QtQuick.Controls 1.4
import QtQuick.Layouts 1.1
import QtGraphicalEffects 1.0
import QtQuick.Dialogs 1.2

import EditorPlugin 1.0

EditorPluginBase
{
    Dialog
    {
        id: dialog;
        visible: true
        title: "My Filter Editor"

        standardButtons: StandardButton.Save | StandardButton.Cancel

        Slider
        {
            id: slider
            width: parent.width
            height: 20
            value: 0
            minimumValue: 0
            maximumValue: 100
        }

        Component.onCompleted:
        {
            slider.value = getPropertyValue(targetModel, "test_property");
        }

        onAccepted:
        {
            setProperty(targetModel, "test_property", slider.value)
        }
    }
}
```

> Kann ich auch ein CE-Plugin erstellen, damit ich dann über QML auf C++ Klassen zugreifen kann (ähnlich wie hier https://doc.qt.io/qt-5/qtqml-cppintegration-topic.html)?

Yep, mittels .adtfceplugin.
Schau dir mal den Guide ADTF CE Module an.

#6 - 2019-04-29 12:21 - hidden

- File ExamplePlugin.zip added

- File ExampleFilter.qml added

Ich habe mir nun beispielhaft einen Filter und ein CE-Plugin erstellt (siehe Anhang). Hänge ich den Debugger an den Configuration Editor, bekomme ich folgende Fehler:

- qml: Error to set DialogManager to module :/module/ExamplePlugin/Main.qml: Error: Cannot assign to non-existent property "dialogManager"
  qml: Error to set MainWindow to module :/module/ExamplePlugin/Main.qml: Error: Cannot assign to non-existent property "mainWindow"

- file:///Path/to/ExampleFilter.qml:9:1: module "ExamplePlugin" is not installed

Wie mache ich nun die Klasse "Example" in der Qml des Filters verfügbar?

#7 - 2019-04-29 12:46 - hidden

- Status changed from Customer Feedback Required to In Progress

#8 - 2019-04-30 12:05 - hidden

Hallo Matthias,

wird das CE_Pluigin nach der Installation auch vom CE geladen? Nur wenn das Plugin nachgeladen wird, kann man den Inhalt auch benutzen --- bzw. nur dann wird der Aufruf von

```
qmlRegisterType<>();
```

ausgefuehrt.

In der main.qml wird Example.qml als zusaetzlicher Property-Editor registriert:

```
PropertyViewConfig.addEditor("Example", "qrc:/module/ExamplePlugin/Example.qml");
```

Das wuerde fuer ein dynamisches nachladen von Example.qml sorgen, falls eine Property innerhalb eines Filters, Services, etc. exisitert, deren Template vom Typ Example ist und damit dann das Verhalten innerhalb der Property-View des CE anpassen. Wenn das nicht so sein soll, kann der Aufruf einfach aus dem Konstruktor raus.

Man kann einfach testen ob das Plugin geladen wurde indem man ein kleines Rechteck oder etwas aehnliches in das QML-Item packt.

```
Item {
    Rectangle {
        color: "yellow"
        x: 10
        y: x
        width: 20
        heigth: width
    }
}
```

Zusaetzliche CE-Module kann man in den CE-Optionen unter den entsprechenden Suchpfaden bekannt machen. Genau wie bei den regulaeren Plugins.

#9 - 2019-04-30 12:06 - hidden

- Status changed from In Progress to Customer Feedback Required

#10 - 2019-05-02 11:30 - hidden

Hallo Sebastian,

ich hatte

```
qmlRegisterType<Example>(uri, 1, 0, "Example");
```

statt

```
qmlRegisterType<Example>("ExamplePlugin", 1, 0, "Example");
```

verwendet. Da die Variable "uri" leer war, konnte das Modul "ExamplePlugin" wohl nicht gefunden werden. Das Modul wird jetzt geladen und ich kann die Klasse in der Qml des Filter verwenden.

Besten Dank für die Hinweise!

#11 - 2019-05-02 13:32 - hidden

- Status changed from Customer Feedback Required to To Be Closed

Super das es geklappt hat! Ich mache das Ticket zu, wenn es noch Fragen gibt einfach melden.

#12 - 2020-01-16 15:57 - hidden

- Subject changed from Dokumentation zu qml Funktionen to Documentation for Filter Editor qml

- Description updated

- Status changed from To Be Closed to Closed

Files

ExamplePlugin.zip	2.71 KB	2019-04-29	hidden
ExampleFilter.qml	935 Bytes	2019-04-29	hidden