Public Support - Support Request #4933

How to use and define Stream Types in ADTF 3.x

2018-11-22 16:28 - hidden

Status:	Closed		
Priority:	Normal		
Category:			
Customer:	ETAS	**Product Issue Numbers:**	
Department:		**Affected Products:**	ADTF 3.3.3
Requester's Priority:	Normal	**Platform:**	Windows 10 64bit
Support Level:	2nd Level	**Topic:**	ADTF::SDK
Resolution:	Solved Issue	**FAQ Links:**	

Description

Supportanfrage

ich bin bei der Entwicklung von Komponenten für die aktuelle 3er Version auf einige Unklarheiten gestoßen, die ich auf diesem Wege versuchen möchte aufzuklären:

1. Mir ist die Verwendung der Funktion create_adtf_default_stream_type (media_description_type.h) nicht klar. Ich hätte erwartet, dass ich einfach eine IStreamType-Instanz zurückgegeben bekomme, allerdings erwartet die Funktion selbst in der Überladung ohne die cSampleCodecFactory-Referenz die Angabe von strStructName und strMediaDescription. Was passiert hier im Hintergrund? Muss bzw. kann ich das Verhalten für meine eigenen Streamtypen reproduzieren?
2. Wie wird die Kompatibilität zwischen zwei Pins festgestellt? Im Configuration Editor war es mir ohne weiteres möglich, einen Out-Pin meiner Streaming Source im Streaming Graph mit einem In-Pin eines Filters im Filter Graph zu verbinden, für den ich einen anderen Streamtypen gesetzt hatte.
3. Welche Rolle spielen Mediadescriptions überhaupt? Werden sie in irgendeiner Form zur Validierung oder sonstiger Logik herangezogen? Oder habe sie nur einen deklarativen Zweck?

Lösung

Zu 1.) Das ist nur eine Hilfsfunktion:

```
tResult create_adtf_default_stream_type(const tChar* strStructName, const tChar* strMediaDescription,
                                        ucom::iobject_ptr<adtf::streaming::IStreamType>& pStreamType,
                                        ddl::tDataRepresentation eRep)
{
    ucom::object_ptr<adtf::streaming::IStreamType> pType =
            ucom::make_object_ptr<adtf::streaming::cStreamType>(stream_meta_type_default());
    RETURN_IF_FAILED(set_stream_type_media_description(*pType, strStructName, strMediaDescription, eRep));
    pStreamType.Reset(pType);
    RETURN_NOERROR;
}
```

Die macht also nichts anderes als einen neuen Stream Type zu erzeugen (mit Metatyp "adtf/default") und dann die 3 Properties zu setzen.

So etwas kann man für seine eigenen Stream Types machen, muss man aber nicht.

2.) Die Kompatibilität wird erst zur Laufzeit überprüft. Das ist auf die Design Entscheidung zurückzuführen, dass sich Stream Types im Lauf der Zeit ändern dürfen. Dadurch lassen sich insbesondere Zyklen im Graph leichter realisieren, bei denen bei Filtern die Datenbeschreibung der Outputs von denen der Inputs abhängen.

Die Überprüfung wird standardmäßig über den Meta Typ realisiert, man kann aber bei einem Reader auch mittels SetAcceptTypeCallback eine eigene Funktion dafür registrieren.

3.) Die Media Description hat in erster Linie den Zweck alle Daten allgemein verarbeiten zu können. Sie richtet sich hier natürlich an

Visualisierungen. Aber da sie auch eine De-/Serialisierung beschreibt, kann sie auch dafür verwendet werden (im Recorder oder den IPC Sourcen und Sinks). Sie ermöglicht es auch allgemeine exporter (über adtf_file/adtfdat_processing) in verschiedene Formate (MDF, HDF, CSV, ...) zu implementieren.

Sie ist daher nicht notwendig um einen Filter Graphen zu bauen, der einfache Regelungstätigkeiten übernimmt. Aber wenn man die Daten mit weiteren Standard ADTF Komponenten verarbeiten oder visualisieren will ist sie von großem Nutzen.

History

#1 - 2018-11-23 08:28 - hidden

- Project changed from Public Support to 21

- Topic set to ADTF::SDK

#2 - 2018-11-26 11:57 - hidden

- Status changed from New to In Progress

Zu 1.) Das ist nur eine Hilfsfunktion:

```
tResult create_adtf_default_stream_type(const tChar* strStructName, const tChar* strMediaDescription,
                                        ucom::iobject_ptr<adtf::streaming::IStreamType>& pStreamType,
                                        ddl::tDataRepresentation eRep)
{
    ucom::object_ptr<adtf::streaming::IStreamType> pType =
            ucom::make_object_ptr<adtf::streaming::cStreamType>(stream_meta_type_default());
    RETURN_IF_FAILED(set_stream_type_media_description(*pType, strStructName, strMediaDescription, eRep));
    pStreamType.Reset(pType);
    RETURN_NOERROR;
}
```

Die macht also nichts anderes als einen neuen Stream Type zu erzeugen (mit Metatyp "adtf/default") und dann die 3 Properties zu setzen.

So etwas kann man für seine eigenen Stream Types machen, muss man aber nicht.

2.) Die Kompatibilität wird erst zur Laufzeit überprüft. Das ist auf die Design Entscheidung zurückzuführen, dass sich Stream Types im Lauf der Zeit ändern dürfen. Dadurch lassen sich insbesondere Zyklen im Graph leichter realisieren, bei denen bei Filtern die Datenbeschreibung der Outputs von denen der Inputs abhängen.

Die Überprüfung wird standardmäßig über den Meta Typ realisiert, man kann aber bei einem Reader auch mittels SetAcceptTypeCallback eine eigene Funktion dafür registrieren.

3.) Die Media Description hat in erster Linie den Zweck alle Daten allgemein verarbeiten zu können. Sie richtet sich hier natürlich an Visualisierungen. Aber da sie auch eine De-/Serialisierung beschreibt, kann sie auch dafür verwendet werden (im Recorder oder den IPC Sourcen und Sinks). Sie ermöglicht es auch allgemeine exporter (über adtf_file/adtfdat_processing) in verschiedene Formate (MDF, HDF, CSV, ...) zu implementieren.

Sie ist daher nicht notwendig um einen Filter Graphen zu bauen, der einfache Regelungstätigkeiten übernimmt. Aber wenn man die Daten mit weiteren Standard ADTF Komponenten verarbeiten oder visualisieren will ist sie von großem Nutzen.

Grüße,

Martin

#3 - 2018-11-26 13:30 - hidden

- Status changed from In Progress to Customer Feedback Required

#4 - 2018-11-26 15:39 - hidden

Danke schonmal,

ich habe noch ein konkretes Problem bezüglich der Verbindungen:

Ich habe eine Datenquelle (cSampleStreamingSource), die dynamische Ausgänge/Pins anbietet. Wenn ein Pin angefordert wird, erstelle ich dazu einen cSampleWriter. Ich möchte dem writer allerdings meinen eigenen cStreamType mitgeben, um falsche Verbindungen zu verhindern. Dazu habe ich meinen StreamMetaType wie folgt definiert:

```
        struct StreamMetaType
        {
            /// The type name of this meta data object
            static constexpr const tChar *const MetaTypeName = "application/gmdf";
            /// The name of the channel name property.
            static constexpr const tChar* const ChannelTypeName = "channel_type";
            /// The name of the media description definitions property.
```

```
            static constexpr const tChar* const MDDefinitionsProperty = "md_definitions";
            /// The name of the media description struct name property.
            static constexpr const tChar* const MDStructProperty = "md_struct";
            /// The name of the media description struct name property.
            static constexpr const tChar* const MDDataSerialized = "md_data_serialized";

            /*!
             * \brief Initializes the properties for of this meta type.
             */
            static tVoid SetProperties( const adtf::ucom::iobject_ptr< adtf::base::IProperties >& pProperties
);
        };
```

Zum Erstellen des Pins erstelle ich dann eine cStreamType-Instanz und erstelle meinen writer:

```
auto streamTypePtr = adtf::ucom::ant::make_object_ptr< adtf::streaming::cStreamType >( gmdf::StreamMetaType( )
 );
adtf::streaming::set_property( *streamTypePtr, gmdf::StreamMetaType::ChannelTypeName, "ABC" );
adtf::streaming::create_pin( *this, writer, ..., streamTypePtr, ...);
```

In meinem FilterGraph habe ich einen Filter, der den StreamTypen "application/gmdf" (und nur diesen) entgegennehmen soll. Dazu erstelle ich meinen reader wie folgt:

```
auto streamTypePtr = adtf::ucom::ant::make_object_ptr< adtf::streaming::cStreamType >( gmdf::StreamMetaType( )
 );
adtf::ucom::object_ptr< const adtf::streaming::ant::IStreamType > readStreamTypePtr = streamTypePtr;
Register( m_sampleReader, "gmdf_data_in", readStreamTypePtr );
```

Die Verbindung klappt (allerdings klappt sie z.B. auch mit dem Demo Reference Generator, was ein anderes Problem ist (zur Laufzeit getestet)), aber ich kann den Wert der "ChannelName" Property ("ABC") in meinem Filter nicht auslesen (gibt immer den in StreamMetaType::SetProperties gesetzten Defaultwert zurück). Ich habe bereits versucht, ein anderen Overload von Register() zu verwenden und SetAcceptCallback auf meinem Reader aufgerufen, ohne einen expliziten StreamType zu setzen - leider ohne Erfolg (der Callback wird nie aufgerufen).

1. Wie muss ich hier vorgehen?
2. Ist es grundsätzlich möglich einen Filter mit einem Pin Trigger und dynamischen Pins zu bauen? Ich hatte hierzu keine geeignete Trigger Funcktion gesehen, daher gehe ich davon aus, dass ich einen von cDynamicFilter abgeleiteten Filter direkt entwickeln muss (falls überhaupt möglich)?

#5 - 2018-11-27 11:43 - hidden

- Status changed from Customer Feedback Required to In Progress

#6 - 2018-12-03 11:33 - hidden

Zu 1.) Hier ist der Weg mit SetAcceptTypeCallback ein callback zu registrieren und dann in diesem die property aus dem übergebenen stream type zu extrahieren. Die ursprüngliche StreamType Instanz die bei Register übergeben wurde wird nie modifiziert.

Wichtig ist auch, dass in einer Funktion die reader.GetNextSample Methode aufgerufen wird, nur innerhalb dieses Aufrufs wird dann das Callback synchron gerufen, ansonsten bleiben Typen und Sample im Puffer liegen.

Zur Info: Die Sample Reader ignorieren solange Samples wie kein akzeptierter StreamType übertragen wurde (der entweder einen passenden Meta Typ hat oder bei dem das Callback keinen Fehler zurückliefert).

Zu 2.) Ja genau, hier muss man cDynamicFilter als Basis verwenden und dann eine Funktion mit RegisterRunner als TriggerFunktion registrieren. Danach kann man mit ConfigureDataInTrigger die TriggerFunktion an einen Pin hängen.

#7 - 2018-12-03 13:53 - hidden

- Status changed from In Progress to Customer Feedback Required

#8 - 2018-12-17 11:08 - hidden

- Subject changed from Klarstellungen zur neuen API to How to use and define Stream Types in ADTF 3.x

- Description updated

- Status changed from Customer Feedback Required to To Be Closed

- Resolution set to Solved Issue

#9 - 2018-12-17 11:08 - hidden

- Project changed from 21 to Public Support

- Status changed from To Be Closed to Closed

- Private changed from Yes to No