Public Support - Support Request #8252

Recognition of DDL encoded Media/StreamTypes

2019-09-04 10:45 - hidden

Status:	Closed		
Priority:	Normal		
Category:			
Customer:	TRACETRONIC	**Product Issue Numbers:**	
Department:		**Affected Products:**	ADTF 2.14.3, ADTF File Library 0.5.0 (BETA)
Requester's Priority:	Normal	**Platform:**	Windows 7 64bit
Support Level:	2nd Level	**Topic:**	ADTF::Common
Resolution:	Solved Issue	**FAQ Links:**	

Description

Supportanfrage

Leider muss ich euch noch einmal mit 2 Verständnisfragen und ein paar technischen Fragen behelligen:

1) Die in den ADTF-2-Examples enthaltenen Beispiel-DATs (example_clock_file.dat und can1.dat) können vom FileAccess-Example nur verarbeitet werden, wenn ich zusätzlich das adtf_devtb_2_deserializer.adtffileplugin aus der Device-Toolbox lade (sonst kommt nur die Fehlermeldung "no sample deserializer factory for adtf.sample.can_message_raw.adtf2_support.serialization.adtf.cid available", wegen des enthaltenen CAN-Daten-Streams). Dass im DAT-Tool ohne zusätzliches Plugin die Typausgabe fehlt, hat wahrscheinlich den gleichen Grund. Darüber hatte mich Florian schon einmal aufgeklärt.

Was ich nun nicht verstehe ist, dass bei Angabe des Plugins der MediaType des CAN-Datenstream als "adtf2/legacy" (major=512, sub=2) ausgegeben wird. Der MediaType "adtf2/legacy" ist doch sonst auch ohne Plugin bekannt!? Noch verwirrender: Wenn ich can1.dat mit dem DAT-Tool in ein neues DAT-File schreibe funktioniert das FileAccess-Example plötzlich ohne Plugin und gibt mir den gleichen MediaType, Major und Sub aus. Wie kommt das?

2) Bisher war meine Annahme, dass das type-Attribute des <stream>-Elements im danebenliegenden Description-File (z.B. type="adtf.core.media_type") den MediaType-Deserializer und die Typrückgabe an der API ("adtf2/legacy") bestimmt. Allerdings funktioniert das FileAccess-Example auch ohne Description-File und gibt mir korrekt "adtf2/legacy" zurück. Steht der MediaType doch im DAT oder funktioniert es nur weil "adtf2/legacy" als Default / Fallback verwendet wird und zufällig passt? Handelt es sich bei der Beziehung zwischen type-Attribut im Description-File und MetaType-Rückgabe an der API um eine eineindeutige Abbildung?

3) Wie kann ich die Streamnamen mit unbekanntem MediaType (der also nur mit nicht vorhandenem Plugin bestimmt werden kann) auslesen? Den Reader bekomme ich bei unbekanntem MediaType (also ohne Plugin) nur mit der Option "ignore_unsupported_streams" instanziert. Dann liefert er mir aber nicht alle Streamnamen. Das DAT-Tool liefert mir hingegen auch die Namen bei unbekanntem MediaType.

4) Wie kann ich sicher DDL-codierte Streams von anderen unterscheiden, die nicht mit der CodecFactory dekordierbar sind? Ist MetaType adtf2/legacy && Major0 && Sub==0 ein eineindeutiges Kriterium? Anders gefragt: Kann die CodecFactory trotz dieser Eigenschaften versagen. Bzw. gibt es andere MediaTypen mit anderen Properties, die sich auch mit der CodecFactory dekodieren lassen?

5) In Zus. mit 3): Welche Einschränkungen gelten für den StreamType-Parameter von adtf::dat::createCodecFactoryFromStreamType() bzgl. Objekttyp und Vorhandensein und Belegung div. Properties etc.?

6) Funktioniert createCodecFactoryFromStreamType vielleicht auch für nicht-DDL-codierte Streams, wenn (z.B.) ein passendes Plugin geladen ist? Bzw. gibt es für diesen Fall eine andere, aber genauso generische Schnittstelle zur Abfrage der Struct-Elementnamen und -werte?

7) Kann die FileLibrary Log-Ausgaben (auf stdout oder in Logdatei) schreiben und wenn ja wie?

Für Antworten auf diese Fragen wäre ich euch wieder sehr dankbar.

Lösung

Zu 1:

> Was ich nun nicht verstehe ist, dass bei Angabe des Plugins der MediaType des CAN-Datenstream als "adtf2/legacy" (major=512, sub=2) ausgegeben wird. Der MediaType "adtf2/legacy" ist doch sonst auch ohne Plugin bekannt!? Noch verwirrender: Wenn ich can1.dat mit dem DAT-Tool in ein neues DAT-File schreibe funktioniert das FileAccess-Example plötzlich ohne Plugin und gibt mir den gleichen MediaType, Major und Sub aus. Wie kommt das?

Nein der Media Typ ist ohne Plugin nicht bekannt, da der nur in serialisierter Form im Index abgelegt ist (auch major und subtype sind serialisierungsabhängig gespeichert). Wenn Du jetzt ein ADTF2 Dat File durchs DAT Tool jagst, kommt per default hinten ein ADTF3 DAT FIle raus, indem die Stream Types unversal nur durch Properties abgebildet sind, daher auch immer lesbar sind.

Zu 2:

> Bisher war meine Annahme, dass das type-Attribute des <stream>-Elements im danebenliegenden Description-File (z.B. type="adtf.core.media_type") den MediaType-Deserializer und die Typrückgabe an der API ("adtf2/legacy") bestimmt. Allerdings funktioniert das FileAccess-Example auch ohne Description-File und gibt mir korrekt "adtf2/legacy" zurück. Steht der MediaType doch im DAT oder funktioniert es nur weil "adtf2/legacy" als Default / Fallback verwendet wird und zufällig passt? Handelt es sich bei der Beziehung zwischen type-Attribut im Description-File und MetaType-Rückgabe an der API um eine eineindeutige Abbildung?

Das Description File ist nur eine zusätzliche Beschreibung. Der Typ ist im Dat File im Index per Namen "adtf.core.media_type" und in serialisierter Form hinterlegt. Das "type" Attribut der Description wird weder in der File Library noch sonst wo (in ADTF) ausgelesen und verwendet.

Zu 3:

> Wie kann ich die Streamnamen mit unbekanntem MediaType (der also nur mit nicht vorhandenem Plugin bestimmt werden kann) auslesen? Den Reader bekomme ich bei unbekanntem MediaType (also ohne Plugin) nur mit der Option "ignore_unsupported_streams" instanziert. Dann liefert er mir aber nicht alle Streamnamen. Das DAT-Tool liefert mir hingegen auch die Namen bei unbekanntem MediaType.

Das geht nur wenn du zusätzlich einen ifhd::v400::IndexedFileReader nimmst und via getStreamName von 0 bis MAX_INDEXED_STREAMS iterierst.

Zu 4:

> Wie kann ich sicher DDL-codierte Streams von anderen unterscheiden, die nicht mit der CodecFactory dekordierbar sind? Ist MetaType adtf2/legacy && Major0 && Sub==0 ein eineindeutiges Kriterium? Anders gefragt: Kann die CodecFactory trotz dieser Eigenschaften versagen. Bzw. gibt es andere MediaTypen mit anderen Properties, die sich auch mit der CodecFactory dekodieren lassen?

Nein. Major/Sub=0 ist nur ein hinreichendes Kriterium. Letztendlich kannst du alles dekodieren, was in der Description Datei steht, nicht mehr und nicht weniger.

Zu 5:

> In Zus. mit 3): Welche Einschränkungen gelten für den StreamType-Parameter von adtf::dat::createCodecFactoryFromStreamType() bzgl. Objekttyp und Vorhandensein und Belegung div. Properties etc.?

Es müssen die zwei Properties "md_struct" und "md_definitions" gesetzt sein. Wichtig der Reader setzt die md_properties bei ADTF2 Files bei den stream_types der streams, die eine Entsprechung im .description file haben automatisch.

Nochmal wichtig: Die Library hast Du auch als Quellcode, da kannst du jederzeit reinsehen.

Zu 6:

> Funktioniert createCodecFactoryFromStreamType vielleicht auch für nicht-DDL-codierte Streams, wenn (z.B.) ein passendes Plugin geladen ist? Bzw. gibt es für diesen Fall eine andere, aber genauso generische Schnittstelle zur Abfrage der Struct-Elementnamen und -werte?

Nein.

Zu 7:

> Kann die FileLibrary Log-Ausgaben (auf stdout oder in Logdatei) schreiben und wenn ja wie?

Nein. Hier hilft nur der Debugger.

History

#1 - 2019-09-04 12:04 - hidden

- Project changed from Public Support to 28

- Topic set to ADTF::Common

- Customer set to TRACETRONIC

- Affected Products ADTF 2.14.3, ADTF File Library 0.5.0 (BETA) added

#2 - 2019-09-04 13:09 - hidden

- Status changed from New to In Progress

#3 - 2019-09-06 08:41 - hidden

Zu 1:

> Was ich nun nicht verstehe ist, dass bei Angabe des Plugins der MediaType des CAN-Datenstream als "adtf2/legacy" (major=512, sub=2) ausgegeben wird. Der MediaType "adtf2/legacy" ist doch sonst auch ohne Plugin bekannt!? Noch verwirrender: Wenn ich can1.dat mit dem DAT-Tool in ein neues DAT-File schreibe funktioniert das FileAccess-Example plötzlich ohne Plugin und gibt mir den gleichen MediaType, Major und Sub aus. Wie kommt das?

Nein der Media Typ ist ohne Plugin nicht bekannt, da der nur in serialisierter Form im Index abgelegt ist (auch major und subtype sind serialisierungsabhängig gespeichert). Wenn Du jetzt ein ADTF2 Dat File durchs DAT Tool jagst, kommt per default hinten ein ADTF3 DAT FIle raus, indem die Stream Types unversal nur durch Properties abgebildet sind, daher auch immer lesbar sind.

Zu 2:

> Bisher war meine Annahme, dass das type-Attribute des <stream>-Elements im danebenliegenden Description-File (z.B. type="adtf.core.media_type") den MediaType-Deserializer und die Typrückgabe an der API ("adtf2/legacy") bestimmt. Allerdings funktioniert das FileAccess-Example auch ohne Description-File und gibt mir korrekt "adtf2/legacy" zurück. Steht der MediaType doch im DAT oder funktioniert es nur weil "adtf2/legacy" als Default / Fallback verwendet wird und zufällig passt? Handelt es sich bei der Beziehung zwischen type-Attribut im Description-File und MetaType-Rückgabe an der API um eine eineindeutige Abbildung?

Das Description File ist nur eine zusätzliche Beschreibung. Der Typ ist im Dat File im Index per Namen "adtf.core.media_type" und in serialisierter Form hinterlegt. Das "type" Attribut der Description wird weder in der File Library noch sonst wo (in ADTF) ausgelesen und verwendet.

Zu 3:

> Wie kann ich die Streamnamen mit unbekanntem MediaType (der also nur mit nicht vorhandenem Plugin bestimmt werden kann) auslesen? Den Reader bekomme ich bei unbekanntem MediaType (also ohne Plugin) nur mit der Option "ignore_unsupported_streams" instanziert. Dann liefert er mir aber nicht alle Streamnamen. Das DAT-Tool liefert mir hingegen auch die Namen bei unbekanntem MediaType.

Das geht nur wenn du zusätzlich einen ifhd::v400::IndexedFileReader nimmst und via getStreamName von 0 bis MAX_INDEXED_STREAMS iterierst.

Zu 4:

> Wie kann ich sicher DDL-codierte Streams von anderen unterscheiden, die nicht mit der CodecFactory dekordierbar sind? Ist MetaType adtf2/legacy && Major0 && Sub==0 ein eineindeutiges Kriterium? Anders gefragt: Kann die CodecFactory trotz dieser Eigenschaften versagen. Bzw. gibt es andere MediaTypen mit anderen Properties, die sich auch mit der CodecFactory dekodieren lassen?

Nein. Major/Sub=0 ist nur ein hinreichendes Kriterium. Letztendlich kannst du alles dekodieren, was in der Description Datei steht, nicht mehr und nicht weniger.

Zu 5:

> In Zus. mit 3): Welche Einschränkungen gelten für den StreamType-Parameter von adtf::dat::createCodecFactoryFromStreamType() bzgl. Objekttyp und Vorhandensein und Belegung div. Properties etc.?

Es müssen die zwei Properties "md_struct" und "md_definitions" gesetzt sein. Wichtig der Reader setzt die md_properties bei ADTF2 Files bei den

stream_types der streams, die eine Entsprechung im .description file haben automatisch.

Nochmal wichtig: Die Library hast Du auch als Quellcode, da kannst du jederzeit reinsehen.

Zu 6:

> Funktioniert createCodecFactoryFromStreamType vielleicht auch für nicht-DDL-codierte Streams, wenn (z.B.) ein passendes Plugin geladen ist? Bzw. gibt es für diesen Fall eine andere, aber genauso generische Schnittstelle zur Abfrage der Struct-Elementnamen und -werte?

Nein.

Zu 7:

> Kann die FileLibrary Log-Ausgaben (auf stdout oder in Logdatei) schreiben und wenn ja wie?

Nein. Hier hilft nur der Debugger.

Grüße,

Martin

#4 - 2019-09-06 10:21 - hidden

- Status changed from In Progress to Customer Feedback Required

#5 - 2019-09-09 10:45 - hidden

Hallo Martin,

vielen Dank für die Erklärungen.

Zu Punkt 4 - Unterscheidung von DDL-codierten, mit codecFactory dekodierbaren Streams von anderen, hätte ich noch ein Rückfrage: Du schreibst Major/Sub=0 ist kein notwendiges Kriterium. Allerdings steht im Quelltext von create_codec_factory_from_stream_type eine Prüfung auf „adtf2/legacy“ und „major/sub == 0“, andernfalls wird mir eine „leere“ CodeFactory zurückgeliefert, mit der ich keine Decoder erzeugen kann. Insofern verstehe ich deine Antwort nicht (es würde mir nichts helfen, wenn ich bspw. zu meinem rawcan-stream mit major/sub=512/2 im description-File zusätzlich die DDL-Infos hinterlege).

Viele Grüße
Thomas

#6 - 2019-09-09 11:20 - hidden

Letztendlich ist auch die Implementierung egal, auch wenn sie nicht dem entspricht, was du sagst:

adtf::dat::ant::createCodecFactoryFromStreamType in ddl_helpers.h:

```
/**
 * creates a ddl codec factory from the given stream type
 * @param [in] stream_type The stream type.
 * @return a codec factory and whether data is serialized or not.
 */
inline std::tuple<ddl::CodecFactory, bool> createCodecFactoryFromStreamType(const std::shared_ptr<const
adtf_file::StreamType>& stream_type)
{
    auto property_type = std::dynamic_pointer_cast<const adtf_file::PropertyStreamType>(stream_type);
    if (!property_type)
    {
        throw std::runtime_error("The stream type is not a property stream type");
    }

    auto media_description = property_type->getProperty("md_definitions");
    auto struct_name = property_type->getProperty("md_struct");
    bool data_is_serialized = false;
    try
    {
        data_is_serialized = property_type->getProperty("md_data_serialized").second == "true";
    }
    catch (...)
    {
    }

    ddl::CodecFactory factory(struct_name.second.c_str(), media_description.second.c_str());
    if (a_util::result::isFailed(factory.isValid()))
```

```
    {
        throw std::runtime_error("unable to parse media description: " + a_util::result::toString(factory.
isValid()));
    }

    return std::tuple<ddl::CodecFactory, bool>(factory, data_is_serialized);
}
```

Du meinst die im Deserializer, das ist aber keine öffentliche Methode sonder die wird zum Deserialisieren der Samples verwendet, also Chunk -> Sample Buffer. Da war es bei ADTF2 in der tat so dass nur dann die DDL zum serialisieren Verwendet wurde, wenn Major/Sub = 0 (Zur Info, die DDL beschreibt immer zwei Repräsentationen in einem, siehe Doku). Andere Sample Implementierungen machten das was auch immer sie meinten (CAN/Flexray).

Zum Zugriff auf die deserialisierten Samples mittels DDL brauchst du nur die "md_structs" und "md_definitions".

#7 - 2019-09-12 09:41 - hidden

- Status changed from Customer Feedback Required to To Be Closed

- Resolution set to Solved Issue

- Platform Windows 7 64bit added

#8 - 2019-09-12 09:49 - hidden

- Project changed from 28 to Public Support

- Description updated

- Private changed from Yes to No

#9 - 2020-07-07 12:49 - hidden

- Status changed from To Be Closed to Closed