Public Support - Support Request #10715

Stream Type error - how to extract elements from complex data type to map a plain type ?

2020-03-06 14:30 - hidden

Status:	Closed		
Priority:	Normal		
Category:			
Customer:	AUDI	**Product Issue Numbers:**	https://www.cip.audi.de/jira/browse/ACORE-10455
Department:		**Affected Products:**	
Requester's Priority:	Normal	**Platform:**	
Support Level:	3rd Level	**Topic:**	ADTF::MessageBus
Resolution:	Solved Issue	**FAQ Links:**	

Description

Supportanfrage

We have tested a co-simulation between dSpace VEOS and ADTF 3. After adaption of your example session (https://www.cip.audi.de/bitbucket/projects/OPENDEV/repos/adtf_community/browse), the communication between VEOS and ADTF 3 works.
Currently we tried to process the data received from VEOS, using a tutorial filter written in your GUIDES, however, We encountered an type error:
error occured in 'unnamed_sample_stream' (category: 'stream_error'): Result code '-38 '(ERR_FAILED) - The destination plain type '' differs from the source plain type 'FLOAT32' [File: c:\workbench\adtf3\adtf-3.6.2-win10_x64_vc141\pkg\adtfstreaming3\include\adtfstreaming3\streamtype.h] [Line: 150] [Func: adtf::streaming::ant::stream_meta_type_plain::IsCompatible]

Backgroud:

1. Processing filter: https://support.digitalwerk.net/adtf/v3/guides/tutorial_filter_processor.html

2. Our input is generated from Simulink using dSPACE ADTF Tool, we converted the type to single, but it seems that has no effect on the error, no matter which data type we change our input to, the error stays the same.

[cid:image001.jpg@01D5F3C1.838B4D80]

Our questions:
What is the cause of this error? Is the inconsistency of the data or ?
How can we adapt our input data to this filter or how can we adapt this filter to our input data?

Lösung

Issue for Exracting Filter created -> ACORE-10455
Code snippets are also attached.

History

#2 - 2020-03-09 08:51 - hidden

- Project changed from Public Support to 11

- Status changed from New to In Progress

- Topic set to ADTF::MessageBus

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

- Customer set to AUDI

#3 - 2020-03-09 09:12 - hidden

Hi Xinjie Huang,

There is no type information transmitted from Simulink to ADTF. Using the community message bus implementation only allows you to specify a DDL struct that is used to classify the data at the message bus source.

So define a DDL Struct that contains only a single float member and store it in a .description file that is then loaded by the Media Description Service. Then specify this struct in the 'struct_name' property of the message bus source and use the same definition for the Input Pin in your Filter.

Or for a quick hack just use 'stream_meta_type_anonymous' for the input pin, which will essentially accepts any type and disables any compatibility checks. This should of course be only used with caution and you are advised to only use it as a temporary solution until you have the DDL setup ready.

Regards,

Martin

#4 - 2020-03-09 19:18 - hidden

- Status changed from In Progress to Customer Feedback Required

#5 - 2020-03-10 10:30 - hidden

Hi Martin,

Thank you for your quick answer. That quick hack worked. Now we have another question:

Suppose the processing filter is from a third party thus we don’t have the access to change it; and for sure there is no type information transmitted from Simulink to ADTF, which means we can’t change our input either. Then comes the question, is there any CONVERTOR in-between, that can convert our input according to the receiving filter? For the input data, we can generate a .description file from Simulink ADTF Tool.

Thank you and,
Best regards.

Xinjie Huang,

#6 - 2020-03-11 10:15 - hidden

Hi Martin,

Thank you for your quick answer. That quick hack worked. Now we have another question:

Suppose the processing filter is from a third party thus we don’t have the access to change it; and for sure there is no type information transmitted from Simulink to ADTF, which means we can’t change our input either. Then comes the question, is there any CONVERTOR in-between, that can convert our input according to the receiving filter? For the input data, we can generate a .description file from Simulink ADTF Tool.

Thank you and,
Best regards.

Xinjie Huang

#7 - 2020-03-11 11:48 - hidden

- Status changed from Customer Feedback Required to In Progress

#8 - 2020-03-12 09:09 - hidden

Hi Xinjie Huang,

no there is no such converter filter, for good reasons :-). As said before, use the 'struct_name' property of the message bus source to define the output stream type. This needs to be compatible to the one the connected filter expects.

Regards,

Martin

#9 - 2020-03-12 09:23 - hidden

- Status changed from In Progress to Customer Feedback Required

#10 - 2020-03-12 15:13 - hidden

[Großmann-Neuhäusler telefonisch]
Es wird eine Fehlermeldung angezeigt
Ist es möglich, telefonisch mit Martin Heimlich in Kontakt zu treten?

@Franz Großmann-Neuhäusler
bitte diese Fehlermeldung hier im Ticket dokumentieren

#11 - 2020-03-13 08:45 - hidden

- File adtf1.description added

- File Playground.plugindescription added

Hi Martin,

Please find our description file generated from Simulink (adtf1.description) and plugindescription file for our filter (Playground.plugindescription) in the attachment.

We have already used the ‚struct_name‘ property of the message bus source. Our boundary conditions are as follows: We get the plugin provided and are not able to change the Pins. The description file for the message bus source is automatically generated by the dSpace ADTF Blockset. We don’t want to change this one as well (if possible). Our assumption is, that these two don’t mach.
I guess it’s quite hard to write everything. Is it possible to talk about this Problem by phone?

Thanks in advance.

Regards

Xinjie & Franz

#12 - 2020-03-16 08:52 - hidden

- Status changed from Customer Feedback Required to In Progress

#15 - 2020-03-25 08:20 - hidden

Hi Xinjie & Franz,

yes, the stream types won't match. The Streaming source will provide a stream type of stream meta type "adtf/default" with a Media Description of "tmy_adtf_message_id_test" (or what you configured in the "struct_name" property). Judging from the plugindescription the Destination Filter expects a stream type of stream meta type "adtf/plain".

These two types are only compatible the other way around. i.e one filter can provide data of "adtf/plain" and another one can receive it with a stream type of "adtf/default" stream meta type set with the exact same media description:

```
<struct name="plain" alignment="1" version="1">
  <element name="value" arraysize="1" type="tInt32">
    <deserialized alignment="1"/>
    <serialized bytepos="0" byteorder="LE"/>
  </element>
</struct>
```

So if you cannot adjust the destination filter implementation, you unfortunately have to implement a conversion filter:

The simplest (but not very type safe method) would be:

```
class cPlainTypeOutputFilter: public cFilter
{
public:
    cPlainTypeOutputFilter()
    {
        CreateInputPin("input");
        m_pWriter = CreateOutputPin("output", stream_type_plain<tInt32>());
    }

    tResult ProcessInput(ISampleReader*, const iobject_ptr<const ISample>& pSample) override
    {
        m_pWriter->Write(pSample);
        RETURN_NOERROR;
    }
private:
    ISampleWriter* m_pWriter = nullptr;
};
```

Regards,

Martin

#16 - 2020-03-25 08:22 - hidden

Keep in mind, that this only works if the data really is only one integer. If it is a more complex structure you have to access the individual elements in the conversion filter.

#17 - 2020-03-25 08:32 - hidden

I quickly hacked this filter. Its not applicable for all use cases as it only supports outputing tInt32 values but should work for you.

```
class cElementExtractorFilter: public cFilter
{
public:
    cElementExtractorFilter()
    {
        RegisterPropertyVariable("element_name", m_strElementName);

        m_pReader = CreateInputPin<decoding_sample_reader<>>("input");
        m_pWriter = CreateOutputPin("output", stream_type_plain<tInt32>());
    }

    tResult ProcessInput(tNanoSeconds, ISampleReader*) override
    {
        cSampleDecoder oDecoder;
        tNanoSeconds tmSampleTime;

        while (m_pReader->GetNextDecoder(oDecoder, tmSampleTime))
        {
            output_sample_data<tInt32> oValue(tmSampleTime, oDecoder.GetElementValue(*m_strElementName).
GetInt32());
            m_pWriter->Write(oValue.Release());
        }

        RETURN_NOERROR;
    }
private:
    property_variable<cString> m_strElementName;
    decoding_sample_reader<>* m_pReader = nullptr;
    ISampleWriter* m_pWriter = nullptr;
};
```

Grüße,

Martin

#18 - 2020-03-25 09:05 - hidden

I have now implemented a generic extractor filter, which should handle every case:

```
class cElementExtractorFilter: public cFilter
{
public:
    ADTF_CLASS_ID_NAME(cElementExtractorFilter,
                       "element_extractor_trigger.filter.adtf.cid",
                       "Element Extractor");
public:
    cElementExtractorFilter()
    {
        RegisterPropertyVariable("element_name", m_strElementName);

        m_pReader = CreateInputPin<decoding_sample_reader<>>("input");
        m_pWriter = CreateOutputPin("output");
    }

    tResult AcceptType(ISampleReader* pReader,
                       const adtf::ucom::iobject_ptr<const IStreamType>& pType) override
    {
        RETURN_IF_FAILED(cFilter::AcceptType(pReader, pType));

        RETURN_IF_FAILED_DESC(ddl::access_element::find_index(*m_pReader, *m_strElementName, m_nElementIndex),
                              "Unable to find element '%s' in source stream type", m_strElementName->GetPtr
());

        object_ptr<IStreamType> pNewType;
        const auto& oElement = m_pReader->GetStaticElement(m_nElementIndex);
        switch (oElement.eType)
        {
        case VT_Int8: pNewType = make_object_ptr<stream_type_plain<tInt8>>(); m_nTypeSize = 1; break;
        case VT_UInt8: pNewType = make_object_ptr<stream_type_plain<tUInt8>>(); m_nTypeSize = 1; break;
        case VT_Int16: pNewType = make_object_ptr<stream_type_plain<tInt16>>(); m_nTypeSize = 2; break;
```

```
        case VT_UInt16: pNewType = make_object_ptr<stream_type_plain<tUInt16>>(); m_nTypeSize = 2; break;
        case VT_Int32: pNewType = make_object_ptr<stream_type_plain<tInt32>>(); m_nTypeSize = 4; break;
        case VT_UInt32: pNewType = make_object_ptr<stream_type_plain<tUInt32>>(); m_nTypeSize = 4; break;
        case VT_Int64: pNewType = make_object_ptr<stream_type_plain<tInt64>>(); m_nTypeSize = 8; break;
        case VT_UInt64: pNewType = make_object_ptr<stream_type_plain<tUInt64>>(); m_nTypeSize = 8; break;
        case VT_Float32: pNewType = make_object_ptr<stream_type_plain<tFloat32>>(); m_nTypeSize = 4; break;
        case VT_Float64: pNewType = make_object_ptr<stream_type_plain<tFloat64>>(); m_nTypeSize = 8; break;
        default:
            RETURN_ERROR_DESC(ERR_NOT_SUPPORTED, "Unsupported element typo '%d'", oElement.eType);
        }

        m_pWriter->ChangeType(pNewType);

        RETURN_NOERROR;
    }

    tResult ProcessInput(tNanoSeconds, ISampleReader*) override
    {
        cSampleDecoder oDecoder;
        tNanoSeconds tmSampleTime;

        while (m_pReader->GetNextDecoder(oDecoder, tmSampleTime))
        {
            object_ptr<ISample> pSample;
            RETURN_IF_FAILED(alloc_sample(pSample, tmSampleTime));

            {
                object_ptr_locked<ISampleBuffer> pBuffer;
                RETURN_IF_FAILED(pSample->WriteLock(pBuffer, m_nTypeSize));
                oDecoder.GetElementValue(m_nElementIndex, pBuffer->GetPtr());
            }

            m_pWriter->Write(pSample);
        }

        RETURN_NOERROR;
    }
private:
    property_variable<cString> m_strElementName;
    decoding_sample_reader<>* m_pReader = nullptr;
    ISampleWriter* m_pWriter = nullptr;
    tSize m_nElementIndex = 0;
    tSize m_nTypeSize = 0;
};
```

#21 - 2020-03-25 10:50 - hidden

- Status changed from In Progress to Customer Feedback Required

#22 - 2020-03-25 16:00 - hidden

- File element_extractor_filter.cpp added

- Resolution set to Solved Issue

- Product Issue Numbers set to https://www.cip.audi.de/jira/browse/ACORE-10455

I've made the filter even more reusable by adding an option the specify the output type.

The attached example filter will be delivered with ADTF 3.8, but you can build it on your own right now.

#23 - 2020-04-01 10:44 - hidden

Hello Xinjie & Franz,

we did not receive more feedback for the ticket.
Can this support ticket be closed?

#24 - 2020-04-01 11:15 - hidden

Hi Digitalwerk Support,

I thought that this ticket has been set to resolved that it doesn’t need feedback from us, sorry for that.
The filter now works perfect! This ticket can be closed.

We would like to thank you again for your support!

Best regards,
Xinjie

#25 - 2020-04-06 09:01 - hidden

- Status changed from Customer Feedback Required to To Be Closed

- Project changed from 11 to Public Support

- Subject changed from ADTF3 stream error 38 Problem to Stream Type error - how to extract elements from complex data type to map a plain type ?

- Description updated

- Private changed from Yes to No

#28 - 2020-07-07 12:49 - hidden

- Status changed from To Be Closed to Closed

Files

image001.jpg	15.9 KB	2020-03-06	hidden
adtf1.description	4.08 KB	2020-03-13	hidden
Playground.plugindescription	5.93 KB	2020-03-13	hidden
element_extractor_filter.cpp	5.56 KB	2020-03-25	hidden