DT High Performance File Format Specification

Data Translation, Inc. (DTI) Engineering staff has prepared this guide for use by DTI personnel as an explanation of the proper use and operation of DTI equipment and software. The drawings and specifications contained herein are the property of DTI and shall neither be reproduced in whole or in part without DTI's prior written approval nor be implied to grant any license to make, use, or sell DTI equipment or software, manufactured in accordance herewith.

Information furnished by Data Translation is believed to be accurate and reliable. However, no responsibility is assumed by Data Translation for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under patent rights of Data Translation, Inc. Data Translation reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented, including but not limited to typographical or arithmetic errors, company policy, and price information.

Document Number: 22760, Rev A

Data Translation, Inc.
100 Locke Drive
Marlborough, MA 01752-1192, USA
Telephone (508) 481-3700
Home Page http://www.datatranslation.com/

Abstract

This specification documents the native binary file format to be supported across all Data Translation application software.

Revision History

Date	Author(s)	Version	Comment
02-25-2006	Dirk Schmischke	0.0	Initial Draft
03-03-2006	Sean Sullivan	0.5	Added : Header Chunk – Int32 CreatorId Int64 IndexChunkOffset Channel Info Chunk – Int32 NumChannels Channel Info Xml - PerChannelSampleRate InputChannelNumber Data Chunk - Int64 dataStartIndex Clarified RangeMin, RangeMax, Offset and Gain ChannelInformation field meanings.
04-27-2006	Sean Sullivan	1.0	Updated EventDefinition and EventData chunk
05-09-2006	Sean Sullivan	1.1	Added tags to EventDefintion for event parameters.
03-12-2007	Brian Wessels	1.2	Editorial changes.

Table of Contents

Description

The HPFF is a chunk file format. Each chunk is defined by a chunk ID, a chunk size including everything, and the chunk data.

The following chunk types are currently defined:
Header chunk
Channel information chunk
Data chunk
Event descriptor chunk
Event data chunk
Index chunk
Trigger chunk

More chunk types could be added easily if the need arises.

Usually an HPF file has the following structure:
Header chunk
Channel information chunk
Data chunk
Data chunk
Data chunk
…

The amount of data in a data chunk is not defined but must be a multiple of 64 KB. The implementing classes will let you set the data chunk size.

Note: All fields described in this document are required, but all readers of the format are not required to support all field values. For example, an application may choose to only support the randomDataChannel channel type, but it must be aware that other types exist and choose to ignore them.

Generic chunk

This section describes the generic layout of a chunk. Every chunk starts with the following structure. Every chunk is padded to 64 KB; this is an artefact of using un-buffered file IO.

Int64 chunkID;
Int64 chunkSize;
Int32[] chunkData;
Padded to 64 KB

chunkID

Identifies the chunk type.

chunkSize

This is the size of this chunk in bytes, including header data. Add this number to the start address of this chunk to find the next chunk.

chunkData

This is the payload of this chunk. The structure of the chunkData depends on the chunkID.

Header chunk

Every HPF file has a header chunk in the first position. It identifies the file and file version.

Int64 chunkID = 0x1000;
Int64 chunkSize = 0x10000;
Int32 CreatorId = FourCC (‘datx’)
Int64 fileVersion = 0x10001; // major minor
Int64 IndexChunkOffset;
Char [] XMLdata;
Padded to 64 kB…

Where XMLData has this (extensible) structure:
<RecordingDate>
  “Recording Date text”
</RecordingDate>

fileVersion

This field contains the major and minor version information for the file. The current version is 0x10001.

XML tag <RecordingDate>

This field specifies the start time of the recording. Its format is yyyy/mm/dd hh:nn:ss.xxx.

“yyyy” – year encoded as 4 digits.
“mm” – month encoded as 2 digits.
“dd” – day encoded as 2 digits.
“hh” – hour encoded as 2 digits. Range 0-23.
“nn” – minute encoded as 2 digits.
“ss” – second encoded as 2 digits.
“xxx” – fractional part of a second encoded as zero to 12 digits.

Channel information chunk

Every HPF file should have at least one channel information chunk. This chunk specifies the recorded data channels.

In one channel information chunk, all channels must specify the same time increment. This means all channels must contain the same amount of data per time unit. You can add more channel information chunks if different time increments per channel must be used.

Int64 chunkID = 0x2000;
Int64 chunkSize;
Int32 groupID;
Int32 NumberOfChannels;
Char [] XMLdata;
Padded to 64 kB…

Where XMLData has this (extensible) structure:

```
<ChannelInformationData>
  <ChannelInformation>
    <Name>
      “channel name text”
    </Name>
    <Unit>
      “channel unit text”
    </Unit>
    < PhysicalChannelNumber >
          “Physical Channel Number text“
    </ PhysicalChannelNumber >
    <PerChannelSampleRate>
          “per channel sample rate“
    </PerChannelSampleRate>
    <ChannelType>
      “channelType text”
    </ChannelType>
    <AssignedTimeChannelIndex>
      “AssignedTimeChannelIndex text”
    </AssignedTimeChannelIndex >
    <DataType>
      “DataType text”
    </DataType>
    <DataIndex>
      “Dataindex text”
    </Dataindex>
    <StartTime>
      “StartTime text”
    </StartTime>
    <TimeIncrement>
      “TimeIncrement text”
    </TimeIncrement>
    <RangeMin>
      “RangeMin text”
```

```
    </RangeMin>
    <RangeMax>
      “RangeMax text”
    </RangeMax>
    <DataScale>
      “DataScale text”
    </DataScale>
    <DataOffset>
      “DataOffset text”
    </DataOffset>
    <SensorScale>
      “SensorScale text”
    </SensorScale>
    <SensorOffset>
      “SensorOffset text”
    </SensorOffset>
  </ChannelInformation>
  … (More ChannelInformation items are possible:)
</ChannelInformationData>
```

groupID

It is possible to have more than one channel information chunk in one file. This field identifies the data chunks that belong to this channel information chunk. Additional channel information chunks with different groupID fields will usually be used if different time increments per channel are used.

XML tag <ChannelInformationData>

This XML Tag encapsulates all ChannelInformation items.

XML tag <ChannelInformation>

This XML tag encapsulates a ChannelInformation item.

XML tag <Name>

Specifies the channel name string.

XML tag <Unit>

Specifies the engineering unit string.

XML tag <PhysicalChannelNumber>

The physical input channel number of the device the data was acquired from.

XML tag <PerChannelSampleRate>

The sample rate (per second) at which the data was acquired. For randomDataChannels this is used to calculate the time between samples.

XML tag <ChannelType>

This tag specifies the type of the channel. Currently three types are implemented:

“calculatedTimeChannel”
The channel data is implicitly defined by startTime and timeIncrement. It is calculated on the fly using these two fields. No additional data is stored in this file. The fields assignedTimeChannelIndex, dataType, dataIndex, rangeMin, rangeMax, dataScale and dataOffset are undefined for this channel type.

“randomDataChannel”
The channel data is contained in data chunks. The data could have random values. The startTime and timeIncrement fields are not defined for this channel type.

“monotonicDataChannel”
The channel data is contained in data chunks. The data must have strictly monotonic increasing values. This channel type could be used if – for some reason – a calculated time channel is not sufficient. The advantage of this type over the random data channel type is that binary searching can be used to find specific entries, and it is easier to determine if data is within a clipping rectangle or not.

XML tag <AssignedTimeChannelIndex>

This field specifies which channel should be used as X-axis data for this channel. If the field is set to -1, no X-axis data is assigned to this channel. This could mean that this channel is used as X-axis data for other channels.

If this field contains a value greater than or equal to zero and less than the number of <ChannelInformation> items, a channel containing X-axis data is assigned to this channel. The assigned channel data is always taken from data with the same groupID.
Note: This field is not defined for <ChannelType> == “calculatedTimeChannel”.

XML tag <DataType>

This field specifies the data type for this channel.
Note: This field is not defined for <ChannelType> == “calculatedTimeChannel”.
Currently implemented data types are:

“Int16”
16 bit signed integer.

“Uint16”
16 bit unsigned integer.

“Int32”
32 bit signed integer.

“Float”
32 bit IEEE floating point number.

“Double”
64 bit IEEE floating point number.

XML tag <DataIndex>

This field specifies at which location the channel data could be found in the channel descriptor table in the data chunk.
Note: This field is not defined for <ChannelType> == “calculatedTimeChannel”.

XML tag <StartTime>

This field specifies the start time of a calculated time channel. Its format is yyyy/mm/dd hh:nn:ss.xxx or “0.” If this field is “0,” the channel specifies time relative to the beginning of recording.

“yyyy” – year encoded as 4 digits.
“mm” – month encoded as 2 digits.
“dd” – day encoded as 2 digits.
“hh” – hour encoded as 2 digits. Range 0-23.
“nn” – minute encoded as 2 digits.
“ss” – second encoded as 2 digits.
“xxx” – fractional part of a second encoded as up to 12 digits.
Note: This field is only defined for <ChannelType> == “calculatedTimeChannel”.

XML tag <TimeIncrement>

This field specifies the time increment of a calculated time channel. The time increment is given in seconds. 1.0 means one second.
Note: This field is only defined for <ChannelType> == “calculatedTimeChannel”.

XML tag <RangeMin>

This field contains the minimum possible value of a data sample in the associated data chunk. For example, data whose source is a 16-bit binary encoding A/D converter would have a RangeMin = 0, and RangeMax = 65535.
Note: This field is not defined for <ChannelType> == “calculatedTimeChannel”.

XML tag <RangeMax>

This field contains the maximum possible value of a data sample in the associated data chunk. For example, data whose source is a 16-bit binary encoding A/D converter would have a RangeMin = 0, and RangeMax = 65535.
Note: This field is not defined for <ChannelType> == “calculatedTimeChannel”.

XML tag <DataScale>

This field specifies the scaling of the data used to convert it to Volts. The raw data in the data chunk can be converted to Volts using the equation dataValue = dataScale * rawdata + dataOffset.
Note: This field is not defined for <ChannelType> == “calculatedTimeChannel”.

XML tag <DataOffset>

This field specifies the offset of the data. The raw data in the data chunk can be converted to Volts using the equation dataValue = dataScale * rawdata + dataOffset.
Note: This field is not defined for <ChannelType> == “calculatedTimeChannel”.

XML tag <SensorScale>

This field specifies the sensor scaling of the data, used to convert it from voltage to its natural unit. The voltage data can be converted to its natural unit (psi for example) using the equation dataValue = dataScale * voltageValue + dataOffset.
Note: This field is not defined for <ChannelType> == “calculatedTimeChannel”.

XML tag <SensorOffset>

This field specifies the sensor offset of the data. The voltage data can be converted to its source unit (volts for example) using the equation dataValue = dataScale * voltageData + dataOffset.
Note: This field is not defined for <ChannelType> == “calculatedTimeChannel”.

Data chunk

This chunk type contains the data.

```
Int64 chunkID = 0x3000;
Int64 chunkSize;
Int32 groupID;
Int64 dataStartIndex;
Int32 channelDataCount;
ChannelDescriptor [channelDataCount] channelDescriptor;
Int32 [] data;
Padded to 64 kB…

Where:
ChannelDescriptor =
{
        Int32 offset;
        Int32 length;
};
```

groupID

It is possible to have more than one channel information chunk in one file. This field identifies the channel information chunk that belongs to this data chunk. Additional channel information chunks with different groupID fields will usually be used if different time increments per channel are used.

dataStartIndex

For data chunks, this is the continued index of the first data entry in this chunk. For example, if there were two data chunks in the file, each containing 1000 values per channel, the dataStartIndex for the first data chunk would be 0, and the dataStartIndex for the second would be 1000. This helps locate the correct data chunk without scanning through the whole file.

channelDataCount

The number of channel descriptor entries.

offset

The byte offset from the beginning of this chunk where the data of this channel starts.

length

The byte length of the channel’s data.

Event Definition chunk

The data in this chunk declares the event types that might be seen in the Event Data Chunk.

Int64 chunkID = 0x4000;
Int64 chunkSize;
Int32 definitionCount;
Char [] XMLdata;
Padded to 64 kB…

Where XMLData has this (extensible) structure:

```
<EventDefinitionData>
  <EventDefinition>
    <Name>
      “Event name text”
    </Name>
    <Description>
      “Event description text”
    </Description>
    <Class>
      “Event class text”
    </Class>
    <ID>
      “Event ID text”
    </ID>
    <Type>
      “Event type text”
    </Type>
    <UsesIData1>
      “true or false”
    </ UsesIData1>
    <UsesIData2>
      “true or false”
    </ UsesIData2>
    <UsesDData1>
      “true or false”
    </ UsesDData1>
    <UsesDData2>
      “true or false”
    </ UsesDData2>
    <UsesDData3>
      “true or false”
    </ UsesDData3>
    <UsesDData4>
      “true or false”
    </ UsesDData4>
    <DescriptionIData1>
      “Description text for iData1”
    </ DescriptionIData1>
    <DescriptionIData2>
```

```
    “Description text for iData2”
  </ DescriptionIData2>
  <DescriptionDData1>
    “Description text for dData1”
  </ DescriptionDData1>
  <DescriptionDData2>
    “Description text for dData2”
  </ DescriptionDData2>
  <DescriptionDData3>
    “Description text for dData3”
  </ DescriptionDData3>
  <DescriptionDData4>
    “Description text for dData4”
  </ DescriptionDData4>
  <Parameter1>
    “Parameter1 value”
  </ Parameter1>
  <Parameter2>
    “Parameter2 value”
  </ Parameter2>
  <Tolerance >
    “Tolerance value”
  </ Tolerance>
  <UsesParameter1>
    “true or false”
  </ UsesParameter1>
  <UsesParameter2>
    “true or false”
  </ UseParameter2>
  <UseTolerance>
    “true or false”
  </ UseTolerance>
  <DescriptionParameter1>
    “Description text for Parameter1”
  </ DescriptionParameter1>
  <DescriptionParameter2>
    “Description text for Parameter2”
  </ DescriptionParameter2>
  <DescriptionTolerance>
    “Description text for Tolerance”
  </ DescriptionTolerance>
 </EventDefinition>
 … more EventDefinition items
</EventDefinitionData>
```

XML tag <EventDefinitionData>

This XML Tag encapsulates all EventDefinitionData items.

XML tag <EventDefinition>

This XML Tag encapsulates an EventDefinition item.

XML tag <Name>

Specifies the name of this event type. This item is only informational.

XML tag <Description>

Provides a short description of this event type. This item is only informational.

XML tag <Class>

The Event Class of this event. The event class helps separate events from different vendors or applications. There are some pre defined event classes:

0x0001
Data Translation Event

XML tag <ID>

A number that uniquely identifies this event within the specified event class. For QuickDataAcq for .NET, all ID values are enum's starting at 0x1000.

```
public const int QuickDataAcqEventBase = 0x1000;
```

XML tag <Type>

Determines the basic event type.

"Point"
The definition describes a singular event in time.

"Ranged"
The definition describes an event with a startTime and an endTime.

XML tag <UsesIData1>

Determines whether this event uses the **iData1** field in the event data chunk. This feature could be used by automated event displays to determine whether a parameter needs to be displayed.

"True"
Use the field.

"False"
Do not use the field.

XML tag <UsesIData2>

Determines whether this event uses the **iData2** field in the event data chunk. This feature could be used by automated event displays to determine whether a parameter needs to be displayed.

"True"
Use the field.

"False"
Do not use the field.

XML tag <UsesDData1>

Determines whether this event uses the **dData1** field in the event data chunk. This feature could be used by automated event displays to determine whether a parameter needs to be displayed.

“True”
Use the field.

“False”
Do not use the field.

XML tag <UsesDData2>

Determines whether this event uses the **dData2** field in the event data chunk. This feature could be used by automated event displays to determine whether a parameter needs to be displayed.

“True”
Use the field.

“False”
Do not use the field.

XML tag <UsesDData3>

Determines whether this event uses the **dData3** field in the event data chunk. This feature could be used by automated event displays to determine whether a parameter needs to be displayed.

“True”
Use the field.

“False”
Do not use the field.

XML tag <UsesDData4>

Determines whether this event uses the **dData4** field in the event data chunk. This feature could be used by automated event displays to determine whether a parameter needs to be displayed.

“True”
Use the field.

“False”
Do not use the field.

XML tag <DescriptionIData1>

Specifies description text for **iData1** in the event data chunk. This feature could be used by automated event displays to show a customized parameter description.

XML tag <DescriptionIData2>

Specifies description text for **iData2** in the event data chunk. This feature could be used by automated event displays to show a customized parameter description.

XML tag <DescriptionDData1>

Specifies description text for **dData1** in the event data chunk. This feature could be used by automated event displays to show a customized parameter description.

XML tag <DescriptionDData2>

Specifies description text for **dData2** in the event data chunk. This feature could be used by automated event displays to show a customized parameter description.

XML tag <DescriptionDData3>

Specifies description text for **dData3** in the event data chunk. This feature could be used by automated event displays to show a customized parameter description.

XML tag <DescriptionDData4>

Specifies description text for **dData4** in the event data chunk. This feature could be used by automated event displays to show a customized parameter description.

XML tag <Parameter1>

Specifies a data value associated with the event. For example, the upper limit of a Between event.

XML tag <Parameter2>

Specifies a data value associated with the event. For example, the lower limit of a Between event.

XML tag <Tolerance >

Specifies a tolerance factor for doing comparisons.

XML tag <UsesParameter1>

Determines whether **Parameter1** is used by this event.

XML tag <UsesParameter2>

Determines whether **Parameter2** is used by this event.

XML tag <UsesTolerance >

Determines whether **Tolerance** is used by this event.

XML tag <DescriptionParameter1>

Specifies description text for **Parameter1**. This feature could be used by automated event displays to show a customized parameter description.

XML tag <DescriptionParameter2>

Specifies description text for **Parameter2**. This feature could be used by automated event displays to show a customized parameter description.

XML tag <DescriptionTolerance>

Specifies description text for Tolerance. This feature could be used by automated event displays to show a customized parameter description.

Event data chunk

This chunk contains events in binary form. The declaration of these events can and should be found in the Event Definition Chunk.

Int64 chunkID = 0x5000;
Int64 chunkSize;
Int64 eventCount;
Event [eventCount] event;
Padded to 64 kB…

Where:
Event =
{
	Int32 Class;
	Int32 ID;
	Int32 channelIndex;
	Int64 eventStartIndex;
	Int64 eventEndIndex;
	Int32 iData1;
	Int32 iData2;
	Double dData1;
	Double dData2;
	Double dData3;
	Double dData4;
};

Class

The Event Class of this event. The event class helps separate events from different vendors or applications. There are some pre defined event classes:

0x0001
Data Translation Event

ID

A number that uniquely identifies this event within the specified event class.

eventStartIndex

For ranged events, this is the data sample at which the event started. For singular events, this is the data sample where the event occurred.

eventEndIndex

For ranged events, this is the data sample where the event finished. For singular events, this field has no meaning.

iData1

A 32 bit signed integer parameter. The purpose of this parameter is determined by event class and event ID.

iData2

A 32 bit signed integer parameter. The purpose of this parameter is determined by event class and event ID.

dData1

A 64 bit IEEE floating point parameter. The purpose of this parameter is determined by event class and event ID.

dData2

A 64 bit IEEE floating point parameter. The purpose of this parameter is determined by event class and event ID.

dData3

A 64 bit IEEE floating point parameter. The purpose of this parameter is determined by event class and event ID.

dData4

A 64 bit IEEE floating point parameter. The purpose of this parameter is determined by event class and event ID.

Index Chunk

This chunk – if present – contains a directory of all chunks in this file. It is used to scan through the file rapidly.

Int64 chunkID = 0x6000;
Int64 chunkSize;
Int64 indexCount;
Index [indexCount] event;
Padded to 64 kB…

Where:
Index =
{
	Int64 dataStartIndex;
	Int64 perChannelDataLengthInSamples;
	Int64 chunkID;
	Int64 groupID;
	Int64 fileOffset;
};

dataStartIndex

For data chunks, this is the continued index of the first data entry in this chunk. For example, if there were two data chunks in the file, each containing 1000 values per channel, the dataStartIndex for the first data chunk would be 0, and the dataStartIndex for the second would be 1000. This helps locate the correct data chunk without scanning through the whole file.

chunkID

The chunk ID value.

groupID

The group ID value.

fileOffset

The offset within the file; where this chunk starts.