A WIDAR Correlator for the Expansion Very Large Array (EVLA):

Technical, Functional, and Performance Discussion Document

EVLA Correlator

Output Data Format

NRC-EVLA Memo# 026

Sonja Vrcic, 21 November 2005

ABSTRACT

In addition to observed astronomical data, the WIDAR correlator archives several types of data products and meta-data. This document lists the types of output data that are generated and defines the format of the output data.

Table of Content

1 Introduction

The output of the WIDAR correlator may be classified as follows:

1. Observed astronomical data
2. Configuration
3. Models
4. Information that accompanies astronomical data (such as state counters)
5. Alarms and logs used to report significant events (errors)

Format of the alarms and logs is defined in the EVLA Memo 18. This memo defines the format for the data types 1 to 4.

Chapter 3 provides an overview of the WIDAR output data and the rough estimate of the total amount of output data for the format specified in Chapter 2.

Appendix A contains an example of the Station Board Output File.

Appendix B contains a brief overview of the formats and technologies that were considered for the WIDAR output data format.

2 Format

WIDAR correlator uses XML for transfer of Monitor and Control data. XML is used both for communication with the EVLA Monitor and Control System (Executor) and for communication among WIDAR subsystems.

The WIDAR output data will be XML encoded whenever possible, i.e. when the amount of output data allows for the use of XML.

All output data, except for:

1. Backend output products (observed data), and
2. "Raw" data saved on Station Board (used for radar mode)

is saved in XML format..

2.1 XML

The format of the XML encoded output data is defined using XML Schema.

Sequences of large numbers, like Station Board Wideband Correlator products, state counters, models and power measurements are encoded as hexBinary content. HexBinary format, which encodes each byte as two ASCII characters and thus doubles amount of data, may in the production mode be replaced by the base64Binary encoding which increases amount of data by "only" 33% percent. HexBinary encoding will be used during testing since it allows user to read data content without any tools.

Appendix A contains an example of the Station Board XML Output File. More examples and the XML Schema for the Station Board output can be found on the following web page:

http://192.139.21.206/~vrcics/

Baseline Board, Backend and MCCC Output files will have the similar format.

2.2 Binary Data

Observed data archived by the Backend and Station Board will be saved as binary data. Each binary file has one line of ASCII text as a header. Each data record is preceded by the record header in the form of one line of ASCII text. ASCII headers enable user to easily identify the content and are regarded as extremely useful during testing.

A line of ASCII text is defined here as an ASCII string followed by the Carriage Return <CR> and Line Feed <LF> characters, which mark the end of the header.

The header itself (the string) does not contain <CR><LF>.

The content of the ASCII headers is described in the following paragraphs.

2.2.1 Time

When part of the ASCII header, time is specified in the format defined by the XML Schema.

It is to be defined whether or not time zone should be specified.

One of the following formats will be used:

a) 2002-10-10T17:00:00

b) 2002-10-10T17:00:00-00:00 (time zone)

c) 2002-10-10T17:00:00:000 (milliseconds)

d) 2002-10-10T17:00:00:000-00:00 (msec and time zone)

When specifying wall time (i.e. the time when file or record was transmitted) it is not necessary to specify milliseconds.

2.2.2 Backend Observed Data File

In addition to the file header and record header defined in the file BE_test.h, the Backend observed data file contains ASCII headers that enable user to easily identify the content.

2.2.2.1 ASCII File Header

The file header is a line of ASCII text that contains the following:

1. string “WidarBackendObservedData”
2. string “ CBE_Node=”, followed by the CBE node identifier in ASCII
3. string “ WallTime”, followed by the time in the form defined in Section 2.2.1.
4. string “ Observation=” followed by the observation identifier (ASCII string).
5. Carriage Return and Line Feed ASCII characters, which mark the end of the ASCII header.

Example:

WidarBackendObservedData CBENode=12 WallTime2005=11-14T18:32:12 Observation=MyFirstObservation

2.2.2.2 ASCII Record Header

Each record header in the Backend Observed Data File is preceded by one line of ASCII text.

1. string “WidarBackendObservedData”
2. string “ xStation-Baseband-Subband=”, followed by the station, baseband and subband identifiers as ASCII numbers connected by dashes (e.g. 234-1-12).
3. string “ yStation-Baseband-Subband=”, followed by the station, baseband and subband identifiers as ASCII numbers connected by dashes (e.g. 234-1-12).
4. Carriage Return and Line Feed ASCII characters, which mark the end of the ASCII header.

Example:

WidarBackendObservedData xStation-Baseband-Subband=234-1-12 yStation-Baseband-Subband=32-0-12

2.2.3 Station Board Raw Data Output File

Raw data saved on the Station Board is saved in binary format. In the same fashion as the Backend Observed Data file, the Raw Data file contains file headers and record headers in ASCII format.

2.2.3.1 ASCII File Header

The file header is a line of ASCII text that contains the following:

1. string “WidarStationBoardRawData”
2. string “ WallTime”, followed by the time in the form defined in Section 2.2.1.
3. string “ Observation=” followed by the observation identifier in ASCII form.
4. string “ StationBoard=” followed by the Station Board identifier in the form “rack-crate-slot”. Rack, crate and slot identifier are ASCII numbers.
5. string “ DataPath=” followed by the Station Board Data Path identifier (ASCII number 0 or 1).
6. string “ Filter=” followed by the filter ID (ASCII number in the range 0 to 17)

7. string " Sample=" followed by the number of bits in the saved sample.

8. Carriage Return and Line Feed ASCII characters, which mark the end of the ASCII header.

Example:

WidarStationBoardRwaData WallTime=2005=11-14T18:32:12
Observation=MyFirstObservation StationBoard=23-1-7 DataPath=1 Filter=12 Sample=8

2.2.3.2 ASCII Record Header

Each record header in the Backend Observed Data File is preceded by one line of ASCII text (a string followed by <CR><LF>). The header contains the following:

1. string "WidarStationBoardRawData"

2. string " WallTime=", followed by the time in the form defined in Section 2.2.1.

3. Carriage Return and Line Feed ASCII characters, which mark the end of the ASCII header.

Example:

WidarStationBoardRwaData WallTime2005=11-14T18:32:12:010

3 Output Data

3.1 Station Board

3.1.1 Configuration, counters and models

Configuration will be saved at the beginning of every output file and after the configuration change.

The following counters and measurement will be integrated over a pre-defined period:

1. Input state counts
2. Time interval counts
3. Power measurements
4. Clip counts
5. RFI counts
6. Tone extractor results
7. Output state counts
8. Error counts

Integration time for the counters is a configuration parameter and is specified as a number of interrupts[1]. Integration time must be specified so that the minimum "dump" time is at least 1 second.

The following models will be saved into the Station Board output file:

1. Path delay model
2. Filter delay model
3. Mixer phase model
4. Tone extractor phase model
5. Output phase model

[1] Interrupt occurs every 10 milliseconds.

6. Pulsar gate model
7. Pulsar bin model

Models will be saved whenever a new model is received from the model server, and when model is updated by CMIB (periodically).

3.1.1.1 File size estimate

The size of the file that contains configuration, models and counters for a Station Board with 2 basebands and 16 subbands per baseband is approximately:

- Binary file: 8Kbytes
- XML file, with binary data in hexBinary format: 40Kbytes

When binary data is encoded using base64Binary format, which encodes 3 bytes as 4 ASCII characters, the file is about 10% smaller (36KB).

Table 1 shows amount of data and traffic that generated in the case when configuration, counters and models are archived in XML format. **Table 1** does not include Wideband Correlator output products.

Table 1 Station Board Output: configuration, counters and models (XML, hexBinary)

	Integration time 1 second (100 interrupts)	**Integration time 10 seconds (1000 interrupts)**
1 second	40KB	4KB
1 minute	2.34MB	240KB
1 hour	140,62MB	14MB
1 day	3.29GB	337MB
Traffic	320Kbps	32Kbps

3.1.2 Wideband Correlator Products

Wideband Correlator (WBC) products are integrated over pre-defined period of time. Since integration time is defined as number of interrupts, the actual time between WBC "dumps" is determined by the number of lags and products.

Total amount of lags cannot exceed 4096. When 4 products are required, the maximum number of lags per product is 1024.

Generated amount of data for 4096 lags is:

1024 lags * 4 products * 8[2] bytes per lag = 32KBytes

Wideband Correlator products will be stored in the same XML output file as configuration, models and counters. WBC products are binary data and can be stored either in hexBinary or in base64Binary format.

Note: When XML base64Binary format is used, 4096 lags, i.e. 32768 bytes of binary data are encoded as 43692[3] ASCII characters (42.6KB).

Table 2 WBC Output Data (base64Binary) – single Station Board (integration time 1 second)

	Binary data	XML - base64Binary	XML - hexBinary
1 second	32KB	42.6KB	64KB
1 minute	1.87MB	2.5MB	3.75MB
1 hour	112.50MB	149.765MB	225.00MB
1 day	2.63GB	3.5GB	5.27GB
Traffic	256Kbps	340.8Kbps	512Kbps

Table 3 WBC Output Data (integration time 1 second) – 28 stations, 112 Station Boards

	Binary data	XML - base64Binary	XML -hexBinary
1 second	3.5MB	4.6MB	7MB
Traffic	28Mbps	37Mbps	54Mbps

[2] 2 bytes for the lag id, 2 bytes for the valid count and 4 bytes for the amplitude.

[3] 32768 bytes + 1 padding byte = (32769 / 3) * 4 = 43692

3.1.3 Station Board Raw Data Saving (Radar Mode)

Station Board allows user to save raw data output of a Station Board filter. A single filter can generate up to 32768 Bytes (32KB) of data every 10 milliseconds.

Raw data saving generates the following amount of traffic:

32KB * 100 = 3.2MB * 8 bits = 25.6Mbps

Depending on the bandwidth, the amount of data can be 2, 4, 6, or 8 times smaller.

Raw data saving can be activated on one filter per baseband.

If raw data saving is active on four boards in the same rack, the amount of output data will exceed 100Mbps (25.6Mbps * 4 = 102.4Mbps).

XML ASCII encoding, which would increase amount of data by at least 33%, is not used for the raw data saving.

Station Board saved (raw) data is saved as binary data in a separate file.

3.2 Baseline Board

3.2.1 Configuration

In the same fashion as Station Board, Baseline Board will save configuration at the beginning of each output file and after every configuration change.

Configuration of the input hardware and Correlator Chips may change independently, as different Correlator Chips may belong to different subarrays. Therefore, Baseline Board may choose to save only partial configuration, and not all 16 inputs and 64 Correlator Chips all the time. Details are still to be defined.

3.2.2 Periodically saved data

The data that has to be saved during observation is still to be defined. At least, Baseline Board will periodically save error counters.

Current estimate is that the amount of data generated by Baseline Board will be a fraction of the data generated by the Station Board.

3.3 Backend

3.3.1 Configuration

In the same fashion as other Widar sub-systems Backend will save configuration after every configuration change. The Backend configuration file will list all currently active baselines with source address (Correlator CHIP), CBE node that process baseline and output file where results are stored.

The size of the configuration file (XML) for a single baseline with 16 subbands is about 10KB.

For a subarray that consist of 28 antennas, there are (28*29)/2 = 406 baselines.

Total amount of configuration data will be about 4MB.

3.3.2 Observed Data

The observed data is stored in the binary file that has an initial file header containing the information about the content of the file.

The file header is followed by alternating meta-data (header) and data records. The content of the header and record header of the Backend output file is given in the relevant include file (BE_test.h).

For more information refer to the CBE User's Guide and Maintenance manual.

3.4 Total Amount of Output Data on the Monitor & Control Network

Total amount of output data generated by WIDAR correlator depends on the configuration.

Table 4 shows the total amount of output data for the subarray that has 28 antennas. There are 8 basebands per antenna and 16 subbands per each baseband. Four Wideband Correlator products are required for each pair of basebands; 1024 lags per product. Raw data saving is active on a single Station Board filter.

The meta-data generated by the Baseline Board and Backend is still to be defined. An assumption is that amount of meta-data generated by the Baseline Board and Backend will be just a fraction of the data generated by the Station Board. The Baseline Board and Backend configuration can be rather lengthy (more than 150KB for a Baseline Board), however, on average, the configuration will remain the same for at least an hour.

Table 4 does not take in consideration the amount of data archived by the MCCC and CPCC. MCCC will archive configuration, however, configuration messages received by MCCC contain less details and are significantly smaller than those forwarded to Baseline

Board and Backend. Numbers in **Table 4** are rough estimate of the amount of meta-data generated by the Baseline Board and Backend.

Table 4 Total amount of data and traffic on M&C network

		Data per second	Traffic
Configuration, counters, models (integration time 1 second)	**One Station Board**	40KB	320Kbps
	28 Stations 112 Station Boards	4.37MB	35Mbps
Configuration, counters, models (integration time 10 seconds).	**One Station Board**	4KB	28.8Kbps
	28 Stations 112 Station Boards	448KB	3.15Mbps
Wideband Correlator 4 products, 1024 lags per product, base64Binary	**One Station Board**	42.6KB	340Kbps
	28 Stations 112 Station Boards	4.6MB	37.2Mbps
Raw Data Saving for one Station Board Filter (one subband)		3.2MB	25.6Mbps
Baseline Board	**One Baseline Board**	1KB	8Kbps
	160 Baseline Boards	160KB	1.25Mbps
Backend	**1 baseline**	0.5KB	4Kbps
	406 baselines	203KB	1.5Kbps
Total amount of data	**1 second integration**	12.58MB	100Mbps
	10 seconds integration	8.65MB	69.2Mbps

4 Appendix A – XML File

This XML file shown below is an example of the Station Board Output file. The file includes example for all the record types (tables) that are generated by a Station Board.

4.1 Example: Station Board Output File

```
 <!—-Station Board configuration: 3-bit input, 4-bit correlation, 2 basebands, 1 subband per baseband -->

StbOutputFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="L:\MyDocuments\OutputDataFormat\WidarOutput.xsd">

<StbMappingTable observationId="MyFirstObservation" rack="150" crate="1" slot="7" wallTime"2001-12-
17T09:30:47.500-05:00" startTime="2001-12-17T09:30:47.000-05:00" integTime="1000">

      <pathMap path="0" inpStream="0">

            <filterMap filter="0" band="0" output="0" station="255" bb="6" sb="0"/>

      </pathMap>

      <pathMap path="1" inpStream="1">

            <filterMap filter="0" band="0" output="0" station="255" bb="7" sb="0"/>

      </pathMap>

</StbMappingTable>
```

```
<StbInpStateCountsTable observationId="MyFirstObservation" rack="150" crate="1" slot="7" wallTime"2001-12-
17T09:30:47.500-05:00" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00"
integTime="1000">

      <StbInpStCntsInteg id="0" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-
05:00">

      <StbInpStCntsPath id="0">

      <!-- 3-bit input: 8 states are encoded as 64 ASCII characters (4 byte numbers, (4*8)*2=64) -->

      <StbInpStCntsBand band="0" states="8">

            1234567890ABCDEF1234567890ABCEDF1234567890ABCDEF1234567890ABCDEF

      </StbInpStCntsBand>

</StbInpStCntsPath>

<StbInpStCntsPath id="1">

      <StbInpStCntsBand band="0" states="8">

            1234567890ABCDEF1234567890ABCEDF1234567890ABCDEF1234567890ABCDEF

      </StbInpStCntsBand>

</StbInpStCntsPath>

</StbInpStCntsInteg>

</StbInpStateCountsTable>
```

```
<StbTickIntTable observationId="MyFirstObservation" rack="150" crate="1" slot="7" wallTime"2001-12-
17T09:30:47.500-05:00" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00"
integTime="1000">

      <StbTickInteg id="0" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-
05:00" value="2147483684"/>

</StbTickIntTable>

<!--  -->

<StbWbcTable observationId="MyFirstObservation" rack="150" crate="1" slot="7" wallTime="2001-12-
17T09:30:47.500-05:00" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00"
integTime="1000" lags="1024" bw="2048000000" >

      <StbWbcProdDef pPath="0" pBand="0" lPath="0" lBand="0">abcNotFullContent12345</StbWbcProdDef>

      <StbWbcProdDef pPath="0" pBand="0" lPath="1" lBand="0">abcNotFullContent12345</StbWbcProdDef>

      <StbWbcProdDef pPath="1" pBand="0" lPath="0" lBand="0">abcNotFullContent12345</StbWbcProdDef>

      <StbWbcProdDef pPath="1" pBand="0" lPath="1" lBand="0"/>abcNotFullContent12345</StbWbcProdDef>

</StbWbcTable>

<!--  -->

<StbModelTable observationId="MyFirstObservation" rack="150" crate="1" slot="7" wallTime="2001-12-
17T09:30:47.500-05:00" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00"
integTime="1000">

<StbModelPath id="0" numPoints="3">

      <!-- 3 eight byte numbers are encoded as 48 ASCII characters. -->

      <StbPathDelPoints>1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</StbPathDelPoints>
```

```
      <StbModelFilter id="0" numTones="2">
            <!-- 3 eight byte numbers are encoded as 48 ASCII characters. -->
            <DelPoints>1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</DelPoints>
            <MixPoints>1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</MixPoints>
            <TexPoints tone="1">1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</TexPoints>
            <TexPoints tone="2">1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</TexPoints>
      </StbModelFilter>
      <StbModelOut id="0" filter="0" fDelay="4294967295">
            <MixPoints>1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</MixPoints>
            <PbnPoints bins="2000">1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</PbnPoints>
      </StbModelOut>
</StbModelPath>
<StbModelPath id="1" numPoints="3">
      <StbPathDelPoints>1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</StbPathDelPoints>
      <StbModelFilter id="0">
            <DelPoints>1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</DelPoints>
            <MixPoints>1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</MixPoints>
      </StbModelFilter>
      <StbModelOut id="0" filter="0" fDelay="4294967295">
```

```
            <MixPoints>1234567890ABCDEFABCD1234567890ABCDEFABCD12345678</MixPoints>
      </StbModelOut>
</StbModelPath>
</StbModelTable>
<!--  -->
<StbErrorCntTable>
<StbErrInteg id="1" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00">
      <StbError id="65535" count="4294967295"/>
      <StbError id="31" count="5"/>
</StbErrInteg>
</StbErrorCntTable>
<!--  -->
<StbRecDataTable>
      <StbRecDataCfg path="1" filter="0" sampleBits="8" file="http://146.88.2.43/MyFirst/RadarData.rad"/>
</StbRecDataTable>
```

```
<StbOutStateCountsTable>

<StbOutStCntsInteg id="1" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-
05:00">

<StbOutStCntsPath id="0">

      <!-- Filter output 4-bits: 16 states are encoded as 128 ASCII chars -->

      <StbOutStCntsFilter filter="0"
states="16">1234567890ABCDEF1234567890ABCEDF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCED
F1234567890ABCDEF1234567890ABCDEF</StbOutStCntsFilter>

</StbOutStCntsPath>

<StbOutStCntsPath id="1">

      <StbOutStCntsFilter filter="0"
states="16">1234567890ABCDEF1234567890ABCEDF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCED
F1234567890ABCDEF1234567890ABCDEF</StbOutStCntsFilter>

</StbOutStCntsPath>

</StbOutStCntsInteg>

</StbOutStateCountsTable>
```

```
<StbTexResTable>

<StbTexResInteg id="1" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00">

<StbTexResPath id="1">

      <StbTexResFilter id="0">

            <StbTone tone="1" cos="18446744073709551615" sin="12345678901234567890" valid="4294967295"/>

            <StbTone tone="2" cos="12345678901234567890" sin="18446744073709551615" valid="1234567890"/>

      </StbTexResFilter>

</StbTexResPath>

</StbTexResInteg>

</StbTexResTable>

<!--  -->

<StbRfiTable observationId="MyFirstObservation" rack="150" crate="1" slot="7" wallTime="2001-12-
17T09:30:47.500-05:00" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00"
integTime="1000">

<StbRfiInteg id="1" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00">

<StbRfiPath id="0">

      <StbRfiFilter id="0" threshold="65535" stretch="65535" detCnt="4294967295"/>

</StbRfiPath>

<StbRfiPath id="1">

      <StbRfiFilter id="0" threshold="65535" stretch="65535" detCnt="4294967295"/>
```

```
</StbRfiPath>

</StbRfiInteg>

</StbRfiTable>

<!--  -->

<StbClipCntTable observationId="MyFirstObservation" rack="150" crate="1" slot="7" wallTime="2001-12-
17T09:30:47.500-05:00" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00"
integTime="1000">

<StbClipCntInteg id="65535" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-
05:00">

<StbClipCntPath id="0">

      <StbClipCntFilter id="0">

            <StbClipCntStage stage="1" scale="65535" counter="4294967295"/>

            <StbClipCntStage stage="2" scale="655" counter="42949295"/>

            <StbClipCntStage stage="3" scale="655" counter="42967295"/>

            <StbClipCntStage stage="4" scale="655" counter="4967295"/>

      </StbClipCntFilter>

</StbClipCntPath>

<StbClipCntPath id="1">

      <StbClipCntFilter id="0">

            <StbClipCntStage stage="1" scale="65535" counter="0"/>
```

```
      </StbClipCntFilter>

</StbClipCntPath>

</StbClipCntInteg>

</StbClipCntTable>

<!--  -->

<StbPowerTable observationId="MyFirstObservation" rack="150" crate="1" slot="7" wallTime="2001-12-
17T09:30:47.500-05:00" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00"
integTime="1000">

<StbPowerInteg id="65" startTime="2001-12-17T09:30:47.000-05:00" endTime="2001-12-17T09:30:57.000-05:00">

<StbPowerPath id="0">

      <StbPwrFilter id="0" qScale="65535" st1Scale="4294967295" st2Scale="4294967295" st3Scale="4294967295"
st4Scale="4294967295" qBits="255" >

      1234567890ABCDEFABCD1234567890ABCDEFABCD12345678

      </StbPwrFilter>

</StbPowerPath>

<StbPowerPath id="1">

      <StbPwrFilter id="0" qScale="65535" st1Scale="4294967295" qBits="255" >

      1234567890ABCDEFABCD1234567890ABCDEFABCD12345678

      </StbPwrFilter>

</StbPowerPath>
```

```
</StbPowerInteg>

</StbPowerTable>

<!--  -->

<WidarEndOfFile observationId="MyFirstObservation" rack="150" crate="1" slot="7" wallTime="2001-12-
17T09:31:01.010-05:00" endTime="2001-12-17T09:30:57.000-05:00"/>
```

</StbOutputFile>

5 Appendix B - Considered Formats / Technologies

The following data formats have been considered:

1. Binary files. Archiving all data in binary format is the most efficient when it comes to use of CMIB CPU, network resources and storage space. However, it is perceived that for testing and maintenance it is beneficial to have at least the file header in a readable format, so that user or operator may easily identify the content of the file.

2. XDR is a binary encoding, data is not human readable, but the format is portable. However, our estimate is that encoding observed data in XDR format would affect performance to such extent that it is not feasible.

3. HDF5: in addition to a portable data format, HDF5 provides a way to organize file system across multiple devices. HDF5 provides an infrastructure for an archiving system; however, at this time we are not sure if EVLA archive will use HDF5, it does not make sense to use it for the Widar output. Data types defined by HDF5 are more or less similar to XDR; in addition HDF5 supports both big-endian and little-endian encoding. In the same fashion as for XDR, encoding observed data in HDF5 would significantly affect performance.

4. XML is used for transfer of Monitor & Control data among all components of the EVLA system. Configuration, models and meta-data can be archived in XML format. The amount of the Backend output data and raw data saved on the Station Board does not allow for the use of ASCII encoding (which increases amount of data by at least 33%).

5. An XOP (XML-binary Optimized Packaging) encoded file can contain both XML encoded meta-data and binary data, or alternatively, can specify names of the files that contain binary data. XOP would provide consistency with the Monitor & Control data format, which is XML encoded. The advantage of the XOP is that meta-data (file header and other general information) is human readable (ASCII). For example, backend output XOP file could contain XML encoded meta-data and a pointer to the binary file (and possibly the description of the file).

   On the other side, XOP would potentially introduce more complexity into file organization and handling, especially if the overall file organization is not coordinated. Another drawback is that XOP is emerging technology. It will be supported in the Java 6.0, which will be available in summer of 2006. For now, we will have to create our own library for writing and reading XOP files.

6. Fast Infosets (the so-called “binary XML”) replaces XML tags with numbers; both tags and data are binary encoded. Fast Infosets is an emerging technology;

ITU-T standard for Fast Infosets has been accepted in September 2005. Sun has started an open source project to develop software libraries and other tools for Fast Infosets.

5.1 XDR

External Data Representation Standard (XDR), defined by RFC 1832, is a standard for description and encoding of data.

http://www.ietf.org/rfc/rfc1832.txt

XDR uses implicit typing, i.e. types are not specified in the file.

XDR uses big-endian encoding. All items require multiple of 4 bytes.

XDR defines the following data types: integer (4 bytes), unsigned integer (4 bytes), enumeration, Boolean, hyper integer (8 bytes), unsigned hyper integer, floating point, double-precision floating point, quadruple-precision floating point, fixed length opaque data, variable length opaque data (first 4 bytes define length), string, fixed length array, variable length array, structure, discriminated union (where the first element defines the type), void, constant, typedef, optional-data (type of union used to define optional data, the second choice is void),

5.2 Hierarchical Data Format – HDF5

http://hdf.ncsa.uiuc.edu/

HDF implements a model for managing and storing data.

HDF5 has three components:

1. a general purpose data model,
2. an I/O library and
3. a file format.

HDF5 library provides C, C++, Fortran 90 and Java interface.

The HDF software is developed and supported by the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, USA, and is freely available (open source).

The HDF project involves the development and support of software and file formats for scientific data management. The HDF software includes I/O libraries and tools for analyzing, visualizing, and converting scientific data.

There are two HDF formats, HDF (4.x and previous releases) and HDF5. These formats are completely different and *not compatible*.

The structure of a HDF5 file is self-describing, an application can navigate HDF5 file to discover and understand all the objects it contains.

Note: It seems that we should decide to use HDF5 only if higher levels of EVLA software (e.g. Data Capture, End-to-End) use the same format.

5.3 XML

5.3.1 XML Data Types

http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#base64Binary

XML has two built-in data types for binary data:

- **base64Binary encoding**, specified in RFC 2045 (MIME - Multipurpose Internet Mail Extensions), is designed to represent arbitrary sequences of octets in a form that need not be humanly readable. It uses a 64-character subset (A-Za-z0-9+/) to represent binary data and = for padding. Base64 processes data as 24-bit groups, mapping this data to four encoded characters. It is sometimes referred to as 3-to-4 encoding. Each 6 bits of the 24-bit group is used as an index into a mapping table (the base64 alphabet) to obtain a character for the encoded data. The encoded data is consistently about 33 percent larger than the unencoded data.

  For compatibility with older mail gateways, RFC 2045 suggests that base64 data should have lines limited to at most 76 characters in length. This line-length limitation is *not* enforced by XML Schema processors.

- **hexBinary** has a lexical representation where each binary octet is encoded as a character tuple, consisting of two hexadecimal digits ([0-9a-fA-F]) representing the octet code. For example, "0FB7" is a hex encoding for the 16-bit integer 4023 (whose binary representation is 111110110111).

5.4 XML-binary Optimized Packaging (XOP)

W3C Recommendation, 25 January 2005

http://www.w3.org/TR/2005/REC-xop10-20050125/#xop_packages

A XOP package is created by placing a serialization of the XML encoded data inside of an extensible packaging format (such as MIME Multipart/Related, RFC 2387). Then, selected portions of its content that are base64-encoded binary data are extracted and re-encoded (i.e., the data is decoded from base64) and placed into the package. The locations of those selected portions are marked in the XML with a special element that links to the packaged data using URIs.

In fact, binary data need never be encoded in base64 form. If the data to be included is already available as a binary octet stream, then either an application or other software acting on its behalf can directly copy that data into an XOP package. When parsing an XOP package, applications may access the binary data directly; the base64 binary character representation can be computed from the binary data only if required by the application.

However, at the conceptual level, this binary data can be thought of as being base64-encoded in the XML Document.

Only element content can be optimized; attributes, non-base64-compatible character data, and data not in the canonical representation of the base64Binary datatype cannot be successfully optimized by XOP.

Example (from the XOP definition):

XML element prior to XOP processing

```
<m:data xmlns:m='http://example.org/stuff'>
        <m:photo>/aWKKapGGyQ=</m:photo>
        <m:sig>Faa7vROi2VQ=</m:sig>
</m:data>
```

The same XML element as XOP package:

```
MIME-Version: 1.0
Content-Type: Multipart/Related;boundary=MIME_boundary;
        type="application/xop+xml";
        start="<mymessage.xml@example.org>";
        start-info="text/xml"
Content-Description: An XML document with my pic and sig in it

--MIME_boundary
Content-Type: application/xop+xml;
        charset=UTF-8;
        type="text/xml"
Content-Transfer-Encoding: 8bit
Content-ID: <mymessage.xml@example.org>

<m:data xmlns:m='http://example.org/stuff'>
        <m:photo><xop:Include
                xmlns:xop='http://www.w3.org/2004/08/xop/include'
                href='cid:http://example.org/me.png'/>
        </m:photo>
        <m:sig><xop:Include
                xmlns:xop='http://www.w3.org/2004/08/xop/include'
                href='cid:http://example.org/my.hsh'/>
        </m:sig>
</m:data>
--MIME_boundary
Content-Type: image/png
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/me.png>
// binary octets for png
--MIME_boundary
Content-Type: application/pkcs7-signature
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/my.hsh>
// binary octets for signature
--MIME_boundary--
```

5.4.1 IETF RFCs related to the XOP definition

RFC1341: MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies

http://www.ietf.org/rfc/rfc1341.txt

A Content-Type header field is used to specify the type and subtype of data in the body of a message and to fully specify the native representation (encoding) of such data.

The following content types are defined:

a) "text", which can be used to represent textual information in a number of character sets and formatted text description languages in a standardized manner.

b) "multipart", which can be used to combine several body parts, possibly of differing types of data, into a single message.

c) "application", which can be used to transmit some other kind of data, typically either uninterpreted binary data or information to be processed by a mail-based application. The primary subtype, "octet-stream", is to be used in the case of uninterpreted binary data, in which case the simplest recommended action is to offer to write the information into a file for the user.

d) "message", for encapsulating a mail message.

e) "image", for transmitting still image (picture) data.

f) "audio", for transmitting audio or voice data.

g) "video", for transmitting video or moving image data, possibly with audio as part of the composite video data format.

Private subtype values are starting with “X-“. Subtype values are not case sensitive.

For example, we could define subtype: “application/x-evlaStbStateCounts”

RFC2387 defines the MIME Multipart/Related Content Type, which is used by XOP.

http://www.ietf.org/rfc/rfc2387.txt

In the example below, the record length list, type Application/X-FixedRecord, consists of a set of INTEGERs in ASCII format, one per line. Each INTEGER specifies the number of octets in the record. The content is specified in an “octet-stream” body part.

Example: MIME Multipart related

```
Content-Type: Multipart/Related; boundary=example-1
        start="<950120.aaCC@XIson.com>";
        type="Application/X-FixedRecord"
        start-info="-o ps"
    --example-1
    Content-Type: Application/X-FixedRecord
    Content-ID: <950120.aaCC@XIson.com>
    25
    10
    34
    10
    25
    21
    26
    10
    --example-1
    Content-Type: Application/octet-stream
    Content-Description: The fixed length records
    Content-Transfer-Encoding: base64
    Content-ID: <950120.aaCB@XIson.com>

    T2xkIE1hY0RvbmFsZCBoYWQgYSBmYXJtCkUgSS
    BFIEkgTwpBbmQgb24gaGlzIGZhcm0gaGUgaGFk
    IHNvbWUgZHVja3MKRSBJIEUgSSBPCldpdGggYS
    BxdWFjayBxdWFjayBoZXJlLAphIHF1YWNrIHF1
    YWNrIHRoZXJlLApldmVyeSB3aGVyZSBhIHF1YW
    NrIHF1YWNrCkUgSSBFIEkgTwo=

    --example-1—
```

5.5 Fast Infoset

https://fi.dev.java.net/

The Fast Infoset specification defines an alternate serialization for XML that uses a binary encoding. Fast Infoset was built to optimize parsing and serialization. Fast Infoset is sometimes referred to as a "Binary XML".

"Binary XML" refers to binary formats that can be integrated into the "XML stack" and converted to and from XML. In fact, the term "Binary XML" is an oxymoron.

The Fast Infoset specification (ITU-T Rec. X.891 | ISO/IEC 24824-1) is standardized jointly by ISO and ITU-T (2005).

Fast Infoset encodes an XML document data as a binary stream and substitutes XML tags with number codes. The table that defines mapping from tags to numbers (and vice versa) is included in the file (stream). The document is generally smaller than comparable XML file, and receiver system can parse it more quickly. Sun claims that software applications perform two or three times faster when using FastInfoset (instead of XML).

Sun has started an open source project FastInfoset @ Java.Net .

Two main use cases of Fast Infoset are Web services and 3D graphics.

Comment: This is emerging technology; although Sun provides open source software it is not clear (to me) if there are any tools available. On the other side, the fact that it has become ITU-T standard probably is a signal that there is more than one company supporting this.