cosylab

Beam Position Monitor

FESA classes interface specification

Revision:	1.7
Status:	Released
Repository:	acc/projects/GSI/FAIR-BPM
Project:	ACC-GSI-SD-Libera
Folder:	doc
Document ID:	CSL-DES-08-44068
File:	DES-Beam_Position_Monitor_FESA_interface.doc
Owner:	Igor Verstovšek
Last modification:	November 24, 2008
Created:	September 26, 2008

Document History

Revision	Date	Changed/ reviewed	Section(s)	Modification
1.0	2008-09-26	msekoranja, gjansa, iverstovsek,	All	Created.
1.1	2008-10-09	gjansa	All	Modified Archive action; modified types to conform to FESA implementation
1.2	2008-10-17	gjansa	4.4,4.5,5.10	Modified type of properties.
1.3	2008-10-22	gjansa	4.12,5.12	Merged start/stop action into run property.
1.4	2008-10-30	gjansa	4.2 5.6,5.7 4.10 5.7, 5.8	Renamed property Merged into one property Added Modified
1.5	2008-11-03	gjansa	5.16 5.18	Modified field type. Added.
1.6	2008-11-13	gjansa	6	Added.
1.7	2008-11-21	gjansa	5.11, 5.12	Added.

Confidentiality

This document is classified as a **`confidential document`**. As such, it or parts thereof must not be made accessible to anyone not listed in the Audience section, neither in electronic nor in any other form.

Scope

This document specifies FESA classes interface of the beam position monitoring (BPM) subsystem. Initially, the software will be installed on 13 Libera BPMs on the SIS-18 synchrotron, but a future installation on all FAIR's synchrotrons is expected.

Audience

The audience of this document are:

- All members of the GSI-SD group.
- All members of the GSI-BEL group.
- All members of development team at Cosylab.

The document can be disclosed to all GSI staff and staff of all entities participating in the FAIR project.

Table of Contents

Figures

Tables

1. INTRODUCTION

1.1. PURPOSE

This document specifies the class structure and their required properties. Every property is detailed in terms of structure, supported values and expected behavior.

1.2. SCOPE

This document specifies only the BPM interfaces. The way to develop classes in FESA or any other framework and how to deploy them is not covered here.

1.3. IMPLEMENTATION FRAMEWORK

Even tough the descriptions given here are based on the FESA framework, the interfaces are at the CMW level. Therefore, any front-end framework providing CMW data objects with the right structure could be usable.

Interfaces use property structure as used in OASIS specification defined by CERN [2].

2. INTERFACE STRUCTURE

There are two interfaces to be implemented, BPMMaster and BPM. Their relationship is depicted on figure below.

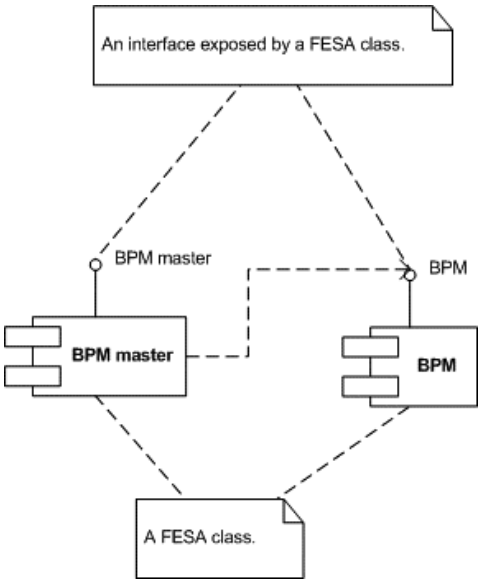

Figure 1: BPMMaster & BPM interfaces

The BPMMaster interface controls a group of BPMs.

3. PROPERTY STRUCTURE

The following two chapters specify the BPMMaster & BPM property interface. They give for each property: a short description, expected behavior, and the fields that must be present in get and set.

3.1. RANGE

As shown in Figure 2, we define two concrete types of settings: The **`ContinuousSetting`** that can take any value from a minimum to a maximum and the **`DiscreteSetting`** that can take only values enumerated in the **`valueRange`** array.

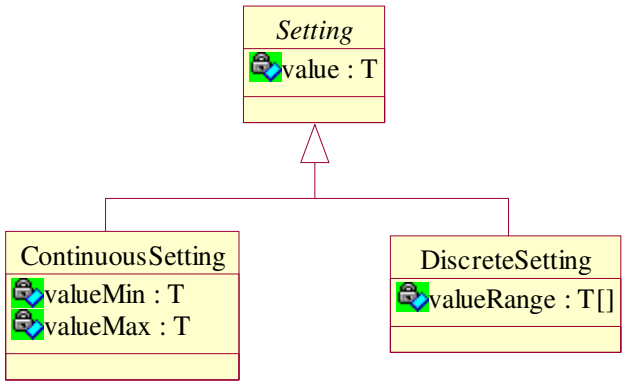

Figure 2: Setting types (reprinted from [2])

Of course, the CMW being based on data maps and not objects, a mapping between the object-oriented view and the data map view is needed (depicted in figure 3). A Boolean is added to let the CMW client know what sub-type of Setting was serialized.

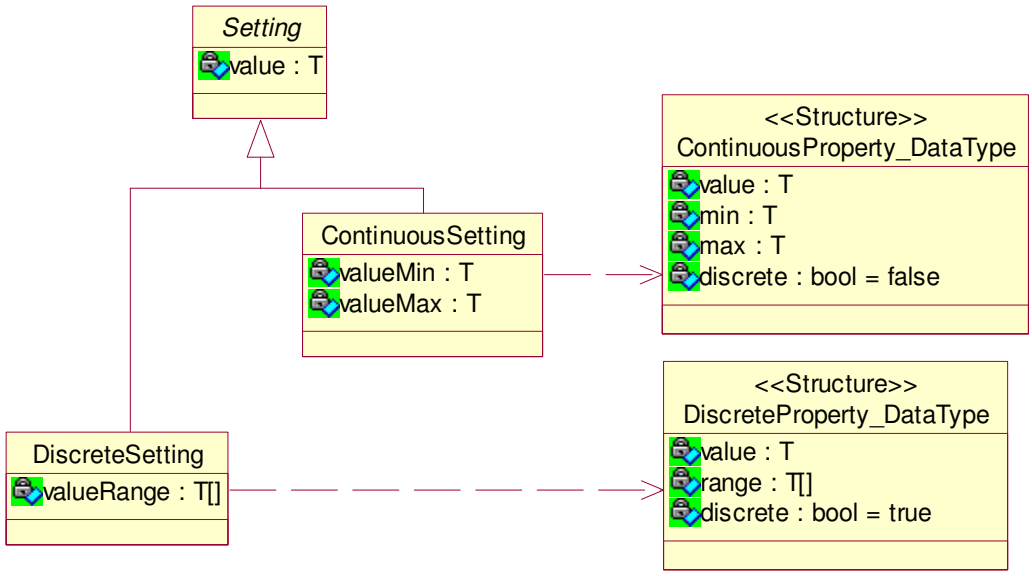

Figure 3: Mapping between Setting type and RDA data (reprinted from [2])

The following fields must be present in all properties representing a setting except for Boolean where the range is quite obvious.

Discrete setting	Continuous settings
value	value
discrete = true	discrete = false
range	min
	max

Table 1: Property fields

Note only the value entry must be sent (and should be expected) for a set of a setting property.

3.2. DEFAULT VALUE

The default value is the value to use when a property is not applicable. In such a case, the value and min/max or range value contain the default value.

3.3. NOTIFICATION

The classes implementing the interfaces must notify the properties on-change only. So, for example, a set of a property with a value equal to the current value must not trigger a notification. Also, if a property value or range changes because of a change in another property, the classes must notify the modified property.

4. BPMMASTER INTERFACE

4.1. BPMS

The BPMs property is a read-only property that returns the list of BPMs names linked to the BPMMaster. Typically, this list is computed run-time from FESA XML instantiation configuration.

4.1.1. Fields

- value: char[][]. Channel device names. Read-Only.

4.2. AUXILIARYBPM

The AuxiliaryBPMs property is a read-only property that returns auxiliary BPM name linked to the BPMMaster. Typically, this name is computed run-time from FESA XML instantiation configuration.

4.2.1. Fields

- value: char[]. Auxiliary BPM device name. Read-Only.

4.3. VIRTUALACCELERATORINFO

The VirtualAcceleratorInfo property is a read-only property that contains data about the current selected virtual accelerator.

4.3.1. Fields

- description: char[]. Virtual accelerator description. Read-Only.

4.4. AMPLIFIERGAIN

The AmplifierGain property controls the gain of all BPM amplifiers.

4.4.1. Fields

Fields of a DiscreteSetting property. Type signed char.

4.4.2. Supported value

Set of valid gain values. Default value: 0db.

4.5. StartEvent

The StartEvent property controls PTIF timing configuration – it specifies start event.

4.5.1. Fields

Fields of a DiscreteSetting property. Type char[].

4.5.2. Supported value

Set of valid start event numbers.

4.6. StopEvent

The StopEvent property controls PTIF timing configuration – it specifies stop event.

4.6.1. Fields

Fields of a DiscreteSetting property. Type char[].

4.6.2. Supported value

Set of valid stop event numbers.

4.7. StartDelay

The StartDelay property controls delay after start event.

4.7.1. Fields

Field of a ContinuousSetting property. Type long.

4.7.2. Supported value

Any positive value in the range. Default value: 0ms.

4.8. VirtualAccelerator

The VirtualAccelerator property selects virtual accelerator.

4.8.1. Fields

Field of a DiscreteSetting property. Type signed char.

4.8.2. Supported value

Set of valid virtual accelerator numbers for current super-cycle.

4.9. STATUS

The Status property is a read-only property that contains status information.

4.9.1. Fields

Fields of a DiscreteSetting property. Type char[].

4.8.2. Supported value

- RUNNING: acquisition is running.
- STOPPED: acquisition stopped (also initial state).
- ERROR: unexpected error state.

4.10. ACQUISITONMODE

The AcquisitonMode property is a property that selects acquisition mode on all BPMs. Property is read-only when acquisition is in progress, i.e. Status property equals RUNNING.

4.10.1. Fields

Fields of a DiscreteSetting property. Type char[].

4.10.2. Supported value

- BUNCH_TO_BUNCH_MODE: bunch to bunch mode.
- RAW_MODE: raw mode.
- ZERO_CALIBRATION_MODE: zero calibration mode.
- POSITION_OFFSET_CALIBRATION_MODE: position offset calibration mode.

4.11. OPERATIONMODE

The OperationMode property is a property that selects operation mode on all BPMs. It is applicable only for Bunch to bunch data mode. Property is read-only when acquisition is in progress, i.e. Status property equals RUNNING.

4.11.1. Fields

Fields of a DiscreteSetting property. Type char[].

4.11.2. Supported value

- CONTINUOUS: continuous mode of running (cycle by cycle).
- ONE_SHOT: one shot and then stop.

4.12. ARCHIVE

The Archive property controls archive options for all BPMs.

4.12.1. Fields

Fields of a DiscreteSetting property. Type char[].

4.12.2. Supported value

- DISABLED: archiving is disabled
- ASCII: archiving in ASCII file
- BINARY: archiving in binary file

4.13. RUN

The Run property is used to arm/unarm all BPMs. When BPM is in arm state is waits for start/stop signals from PTIF.

4.13.1. Fields

Fields of a DiscreteSetting property. Type char[].

4.13.2. Supported value

- ON: BPMs are armed.
- OFF: BPMs are unarmed.

5. BPM INTERFACE

5.1. POSITION

The Position property is a read-only property that returns a position number in a ring (0 is a position of the first BPM after injection).

5.1.1. Fields

- value: signed char. BPM position in a ring. Read-Only.

5.2. STATUS

The Status property is a read-only property that contains status information.

5.2.1. Fields

Fields of a DiscreteSetting property. Type char[].

5.2.2. Supported value

- RUNNING: acquisition is running.
- STOPPED: acquisition stopped (also initial state).
- ERROR: unexpected error state.

5.3. EXPERTINFO

The ExpertInfo property contains expert related information about the BPM. The property is read-only.

5.3.1. Fields

- ip: IP address of the BPM. char[4] type.
- port: UDP port used by BPM. short type.

5.4. AMPLIFIERGAIN

The AmplifierGain property controls the gain of the BPM amplifier.

5.4.1. Fields

Fields of a DiscreteSetting property. Type signed char.

5.4.2. Supported value

Set of valid gain values. Default value: 0db.

5.5. AcquisitonMode

The AcquisitonMode property is a property that selects acquisition mode. Property is read-only when acquisition is progress, i.e. Status property equals RUNNING.

5.5.1. Fields

Fields of a DiscreteSetting property. Type char[].

5.5.2. Supported value

- BUNCH_TO_BUNCH_MODE: bunch to bunch mode.
- RAW_MODE: raw mode.
- ZERO_CALIBRATION_MODE: zero calibration mode.
- POSITION_OFFSET_CALIBRATION_MODE: position offset calibration mode.

5.6. OperationMode

The OperationMode property is a property that selects operation mode. It is applicable only for Bunch to bunch data mode. Property is read-only when acquisition is in progress, i.e. Status property equals RUNNING.

5.6.1. Fields

Fields of a DiscreteSetting property. Type char[].

5.6.2. Supported value

- CONTINUOUS: continuous mode of running (cycle by cycle).
- ONE_SHOT: one shot and then stop.

5.7. ACQUISITION

The Acquisition property is triggered at the end of each cycle. It returns information about the acquisition. Property itself does not provide any acquired data. Specific actions should be used to get the data on demand. The property is read-only.

5.7.1. Fields

- missedCycleCount: increments if cycle is missed (not completely acquired). Type long.
- bunchDataCount: number of bunch data acquired. Type long.
- acquistionLength: time in ms of length of the acquisition (time of the last bunch). Type float.
- cycleStamp: UTC timestamp of the cycle. Type long long.
- cycleTag: Timing system tag for the cycle (to be added later).

5.8. ZEROLINECALIBRATION

The ZeroLineCalibration property takes care of zero line calibration.

5.8.1. Fields

- lastCalibrationData: date of last calibration. Read-only. Type long long.
- status: status of calibration. Read-only. Type bool.
- save: Boolean flag to save the acquired zero line values. Type bool.
- reset: Boolean flag to reset zero line values to zero. Type bool.
- binSize: number of samples to acquire to calculate (average) zero line value. Type long.
- value: (left, right, up, down) value. Read only. Type short[4].

5.9. POSITIONOFFSETCALIBRATION

The PositionOffsetCalibration property takes care of position offset calibration.

5.9.1. Fields

- lastCalibrationData: date of last calibration. Read-only.
- status: status of calibration. Read-only. Type bool.
- save: Boolean flag to save the acquired position offset values. Type bool.
- reset: Boolean flag to reset position offset values to zero. Type bool.

- binSize: number of samples to acquire to calculate (average) position offset value. Type long.
- value: (x, y) value. Read only. Type float[2].

5.10. TuneFFTWindowSize

The TuneFFTWindowSize property sets number of bunch data (samples) to be used to calculate FFT. Read-only during acquisition.

5.10.1. Fields

Fields of a DiscreteSetting property. Type long.

5.10.2. Supported value

Set of valid sizes. Default 1024.

5.11. TuneFFTMode

The TuneFFTMode property controls how FFT is calculated.

5.11.1. Fields

Fields of a DiscreteSetting property. Type char[].

5.11.2. Supported value

- BUNCH_BY_BUNCH: all bunches are used for FFT calculation
- TURN_BY_TURN: only one bunch is used for FFT calculation

5.12. Harmonic

The Harmonic property is used to control harmonic value.

5.12.1. Fields

Field of a ContinuousSetting property. Type short.

5.12.2. Supported value

Any positive value in the range. Default value: 4.

5.13. ARCHIVE

The Archive property controls archive options for BPM.

5.13.1. Fields

Fields of a DiscreteSetting property. Type char[].

5.13.2. Supported value

- DISABLED: archiving is disabled
- ASCII: archiving in ASCII file
- BINARY: archiving in binary file

5.14. RUN

The Run property is used to arm/unarm BPM. When BPM is in arm state it waits for start/stop signals from PTIF.

5.14.1. Fields

Fields of a DiscreteSetting property. Type char[].

5.14.2. Supported value

- ON: BPM is armed.
- OFF: BPM is unarmed.

5.15. ACTION GETLOADSTATUS(BINSIZE, START, END, UNIT, POINTS)

Get BPM underload and overload status.

5.15.1. Arguments

- binSize: average bin size (> 0). Type long.
- start: start offset in the cycle. Type double.
- end: end offset in the cycle. Type double.
- unit: start and end argument unit. Type enum (BUNCH_INDEX, TIME_MS). If TIME_MS returned data will be interpolated over time so that they will fit equidistant time X axis.
- points: number of points to be returned. Type long.

5.15.2. Returns

Underload and overload status bits (u4, u3, u2, u1, o4, o3, o2, o1). Type char[].

5.16. Action getTrendX(binSize, start, end, unit, points)

Get BPM X positions in mm over time.

5.16.1. Arguments

See getLoadStatus() action arguments description.

5.16.2. Returns

X positions. Type float[].

5.17. Action getTrendY(binSize, start, end, unit, points)

Get BPM Y positions in mm over time.

5.17.1. Arguments

See getLoadStatus() action arguments description.

5.17.2. Returns

Y positions. Type float[].

5.18. Action getTrendXY(binSize, start, end, unit, points)

Get BPM (X, Y) positions in mm over time. Also used to get closed orbit data.

5.18.1. Arguments

See getLoadStatus() action arguments description.

5.18.2. Returns

(X, Y) positions. Type float[2][].

5.19. Action getBunchWindow (start, end, unit, points)

Get bunch windows (t2 - t1) in ms within the cycle, used for consistency check of data.

5.19.1. Arguments

- start: start offset in the cycle. Type double.
- end: end offset in the cycle. Type double.
- unit: start and end argument unit. Type enum (BUNCH_INDEX, TIME_MS). If TIME_MS returned data will be interpolated over time so that they will fit equidistant time X axis.
- points: number of points to be returned. Type long.

5.19.2. Returns

Type long[].

5.20. ACTION GETINTENSITY (START, END, UNIT, POINTS)

Get intensity within the cycle.

5.20.1. Arguments

See getBunchWindow() action arguments description.

5.20.2. Returns

Type long[].

5.21. ACTION GETTUNEX (BUCKET, START, UNIT, POINTS)

Get tune of X positions. Property TuneFFTWindowSize defined number of points to be used (and returned) to calculate FFT.

5.21.1. Arguments

- bucket: select bucket (bunch number) for turn by turn mode, if negative (-1) bunch by bunch mode is used.
- start: start offset in the cycle. Type double.
- unit: start and end argument unit. Type enum (BUNCH_INDEX, TIME_MS) .
- points: number of points to be returned. Type long.

5.21.2. Returns

Type float[].

5.22. Action getTuneY (bucket, start, unit, points)

Get tune of Y positions.

5.22.1. Arguments

See getTuneX().

5.22.2. Returns

Type float[].

5.23. Action getTuneQxQy (bucket, start, end, unit)

Get (X, Y) tune.

5.23.1. Arguments

- bucket: select bucket (bunch number) for turn by turn mode, if negative (-1) bunch by bunch mode is used.
- start: start offset in the cycle. Type double.
- end: end offset in the cycle. Type double.
- unit: start and end argument unit. Type enum (BUNCH_INDEX, TIME_MS).

5.23.2. Returns

Type float[][2].

5.24. Action getEnvelopedTrendX (bucket, start, end, unit, points)

Get enveloped trend of X positions.

5.24.1. Arguments

- bucket: select bucket (bunch number) for turn by turn mode, if negative (-1) bunch by bunch mode is used.
- start: start offset in the cycle. Type double.
- end: end offset in the cycle. Type double.
- unit: start and end argument unit. Type enum (BUNCH_INDEX, TIME_MS).
- points: number of points to be returned. Type long.

5.24.2. Returns

(min, max) arrays. Type float[2][].

5.25. Action getEnvelopedTrendY(bucket, start, end, unit, points)

Get enveloped trend of Y positions.

5.25.1. Arguments

See getEnvelopedTrendX() arguments.

5.25.2. Returns

(min, max) arrays. Type float[2][].

5.26. Action getTunesX (bucket, binSize, start, end, unit, points)

Get array of tunes for X position.

5.26.1. Arguments

- bucket: select bucket (bunch number) for turn by turn mode, if negative (-1) bunch by bunch mode is used.
- binSize: sum bin size. Type long.
- start: start offset in the cycle. Type double.
- end: end offset in the cycle. Type double.
- unit: start and end argument unit. Type enum (BUNCH_INDEX, TIME_MS).
- points: number of points to be returned (X axsis). Type long.

5.26.2. Returns

(index/time of bunch/turn, tune) array. Type float[][].

5.27. Action getTunesY (bucket, binSize, start, end unit, points)

Get array of tunes for Y position.

5.27.1. Arguments

See getTunesX() arguments.

5.27.2. Returns

(index/time of bunch/turn, tune) array. Type float[][].

5.28. ACTION GETPHASEADVANCE (START, END, UNIT, POINTS)

Get phase advance.

5.28.1. Arguments

- start: start offset in the cycle. Type double.
- end: end offset in the cycle. Type double.
- unit: start and end argument unit. Type enum (BUNCH_INDEX, TIME_MS).
- points: number of points to be returned. Type long.

5.28.2. Returns

Type float[].

5.29. ACTION GETBUNCHDATA (STARTINDEX, ENDINDEX)

Get bunch data, used for phase advance calculation.

5.29.1. Arguments

- startIndex: start offset index in the cycle. Type double.
- endIndex: end offset index in the cycle. Type double.

5.29.2. Returns

Type byte[][2]. (each bunch consists of 96-bit data)

6. AUXILIARY BPM INTERFACE

6.1. STATUS

The Status property is a read-only property that contains status information.

6.1.1. Fields

Fields of a DiscreteSetting property. Type char[].

6.1.2. Supported value

- RUNNING: acquisition is running.
- STOPPED: acquisition stopped (also initial state).
- ERROR: unexpected error state.

6.2. EXPERTINFO

The ExpertInfo property contains expert related information about the BPM. The property is read-only.

6.2.1. Fields

- ip: IP address of the BPM. char[4] type.
- port: UDP port used by BPM. short type.

6.3. ACQUISITONMODE

The AcquisitonMode property is a property that selects acquisition mode. Property is read-only when acquisition is in progress, i.e. Status property equals RUNNING.

6.3.1. Fields

Fields of a DiscreteSetting property. Type char[].

6.3.2. Supported value

- BUNCH_TO_BUNCH_MODE: bunch to bunch mode.
- RAW_MODE: raw mode.
- ZERO_CALIBRATION_MODE: zero calibration mode.

- POSITION_OFFSET_CALIBRATION_MODE: position offset calibration mode.

6.4. ACQUISITION

The Acquisition property is triggered at the end of each cycle. It returns information about the acquisition. Property itself does not provide any acquired data. Specific actions should be used to get the data on demand. The property is read-only.

6.4.1. Fields

- missedCycleCount: increments if cycle is missed (not completely acquired). Type long.
- bunchDataCount: number of bunch data acquired. Type long.
- acquistionLength: time in ms of length of the acquisition (time of the last bunch). Type float.
- cycleStamp: UTC timestamp of the cycle. Type long long.
- cycleTag: Timing system tag for the cycle (to be added later).

6.5. ARCHIVE

The Archive property controls archive options for BPM.

6.5.1. Fields

Fields of a DiscreteSetting property. Type char[].

6.5.2. Supported value

- DISABLED: archiving is disabled
- ASCII: archiving in ASCII file
- BINARY: archiving in binary file

6.6. RUN

The Run property is used to arm/unarm BPM. When BPM is in arm state it waits for start/stop signals from PTIF.

6.6.1. Fields

Fields of a DiscreteSetting property. Type char[].

6.6.2. Supported value

- ON: BPM is armed.
- OFF: BPM is unarmed.

6.7. ACTION GETSIGNAL(SIGNAL, BINSIZE, START, END, UNIT, POINTS)

Get BPM underload and overload status.

6.7.1. Arguments

- signal: selects which signal to get (1-4). Type long.
- binSize: average bin size (> 0). Type long.
- start: start offset in the cycle. Type double.
- end: end offset in the cycle. Type double.
- unit: start and end argument unit. Type enum (BUNCH_INDEX, TIME_MS). If TIME_MS returned data will be interpolated over time so that they will fit equidistant time X axis.
- points: number of points to be returned. Type long.

6.7.2. Returns

Type float[].

7. DOCUMENT PROPERTIES

7.1. REFERENCES

[1] I. Verstovsek et al.: BPM Requirement Specification, revision 1.3, 24-09-2008

[2] S. Deghaye: OASIS Digitiser Interface, CERN 2007

[3] T. Hoffmann: *SIS 18 BPM-System*, **TH-Notiz-2008-007**.

[4] T. Hoffmann: *BPM DAQ System Study by Cosylab*, **TH-Notiz-008-2008.**

[5] T. Hoffmann, P.Forck, P. Kowina, K. Lang: *Collection of BPM specifications*, **TH-Notiz-012-2008**

[6] I. Verstovšek: SIS 18 BPM Upgrade, Minutes of Kick-off Meeting, revision 1.0, July 23 2008

