ANKA Alarm end EPICS Application Server

a. Contents

b. Glossary of Terms

Term	Description
CSS BEAST	Control system studio implementation of EPICS archiver called BEAST
CA	Channel Access protocol
JCA	Java implementation of CA protocol on client side
CAS	Java implementation of CA protocol on server side

1. Overview

The ANKA business logic server provides easy way to implement server side logic in Java and offer it to various application through EPICS 3 and CA interface and communication protocol. It has similar role as EPICS Database, yet does not replaces it. It supplements it where is weak or convenient, at implementing of algorithm and complex procedures.

1.1. The Alarm Side of Application Server

The ANKA alarm server solution is build around CSS BEAST alarm. The ANKA alarm server is implementation of ANKA EPICS application server with focus on supporting functionality of CSS BEAST alarm engine.

Following are integral parts of alarm installation at ANKA:

- The CSS BEAST alarm server
- The CSS BEAST panels for viewing alarms
- The CSS BEAST alarm utility for managing configuration
- The CSS BEAST prescribed and configured PostgreSQL and JMS
- The CSS BEAST jms2rdb server
- The ANKA alarm server for intercepting and filtering alarms and as well generating own alarms.
- EPICS IOCs with enabled alarm capabilities as source of device-based alarm events.

All alarms are first intercepted by the ANKA alarm server and if some criteria is met, like machine operation is in correct state, then they are sent further to the BEAST alarm engine. The alarm server also monitors some vital signals of the machine and generates appropriate alarm signals to BEAST. The BEAST is configured by configuration provided by ANKA alarm server.

The ANKA alarm server works with existing CA and CSS BEAST communication and and interface standards. There is no new interface and no patching done to CA or CSS BEAST. The alarm server listens to existing EPICS alarms through CA and provides further alarms to CSS BEAST as standard CA channels. JCA library with CAS is used for this purpose. Because of this general nature of alarm server it is used for additional tasks and application as Java application server.

2. The ANKA Alarm Server

This server is implemented on top of Java CA server (CAS). It has following functionality:

- Listens to other PVs and alarms. It relays alarms further if conditions (for example operation state) this allows.
- It hosts alarms, which are not directly connected to devices readout. This is equivalent to Soft records.
  - Checks various device statuses.
  - Provides a watchdog functionality for monitoring important services and processes.
- Filters alarms so particular alarms are generated or forwarded only when machine is in appropriate state.

2.1. Running the ANKA Alarm Server

Since ANKA alarm server is extension of BEAST alarm, later must be in running condition.

BEAST is installed at ANKA at `ankasr-serv:/home/operator/applications/epics/extras/alarm-system` and is started or stopped by calling commands: `css-alarm-server start`, `css-jms2rdb start` or `css-alarm-server stop`, `css-jms2rdb stop` on `ankasr-serv`.

ANKA alarm server is part of distribution bundle called `ANKA-Servers`, which is installed at `ankasr-serv`. Startup scripts are located in folder `ankasr-serv:/home/operator/applications/ANKA-Servers/sh`.

ANKA alarm server is started or stopped with following commands: `anka-alarm-server start` or `anka-alarm-server stop`.

2.2. Updating/restarting the server

The main ANKA alarm server file `alarms.xml` has been changed, then following procedure should be executed to update and restart depending servers and configurations.

1. Update or commit via SVN the ANKA alarm configuration, depending if change occurred locally at `ankasr-serv` or remotely.
   a. To update server call `svn up` in folder `ankasr-serv:/home/operator/applications/ANKA-Servers`.
2. Restart ANKA alarm server by calling in sequence: `anka-alarm-server stop` then `anka-alarm-server start.`
3. Then load configuration changes into BEAST server and restarting BEAST servers. This is done by calling `./sh/load_alarms.sh` in folder `ankasr-serv:/home/operator/applications/ANKA-Servers`.

3. ANKA-Server Distribution Bundle

The ANKA-Servers distribution bundle contains configuration files, executables and libraries and tools needed to start ANKA alarm and application servers.

The bundle can be found in SVN repository: http://ankasvn/svn/machine/distro/ANKA-Servers/trunk and is distributes and updated trough SVN version control mechanism.

Here is structure of the distribution bundle:

- bat - contains MS Windows scripts for running servers
- config - contains configuration files
  - bundle.properties - defines home and configuration folders for Java servers
  - log4j.properties - log4j configuration file
- lib - contains libraries, tools and executables
  - lib/java - Java OS independent libraries, the jar files
  - lib/lin - Linux executables
  - lib/win - MS Windows executables
  - lib/resources - General folder for resources, such as icons.
- log - contains log files
- sh - contains Linux scripts for running servers

3.1. Startup Script

The application servers finds resources located within the distribution bundle trough runtime configuration in the startup scripts.

The alarm server for example is started by `sh/AlarmServer.sh` script. In The script makes three important first or declarations

- `export AS_NAME="AlarmServer"` - this tells what should be the name of log file. It must be declared before main initialization happens.
- `. "$(dirname "$0")/_init.sh"` - call to the main bundle initialization script which sets up environment variables to define Java JRE and classpath.
- `$AS_JAVA -classpath $AS_CLASSPATH  -Dlog4j.debug=false -DAppServer.init=AppServer-alarms.properties com.kriznar.csshell.epics.server.Server` - this line starts the server looks

The startup line contains following elements:

- `$AS_JAVA` - is java executable as defined in the _init.sh script,
- `$AS_CLASSPATH` - is classpath as defined in the _init.sh script,
- `-DAppServer.init=AppServer-alarms.properties` - is name of properties file, which defines initialization properties for the server.

3.2. The Application Server Initialization Configuration

Once the application server `com.kriznar.csshell.epics.server.Server` is started it will try to load a bootstrap file `bundle.properties` (by a way of system parameters or classloader), which is in our case located in `config/bundle.properties`. This file tells server where home and config folders can be found.

The application server will get assigned own configuration folder, which is in case of ANKA bundle in folder `config/AppServer/`.

The server receives instructions what to load and start with the `AppServer.init` System property:

System Property	Description	Required	Default value
AppServer.init	Application server initialization file, for example AppServer-alarms.properties.	No	AppServer.properties

Following properties could be provided or through this initialization file (in case of ANKA alarm server `config/AppServer/AppServer-alarms.properties`) or could be optionally defined as System properties. System properties take precedence.

The basic set of properties (in AppServer.init file):

Property	Description	Required	Default Value	Alarm server example value
AppServer.input	Application server input XML file with definitions of EPICS records to be started. Location is relative to the server configuration folder (provided by bundle: `config/AppServer/`).	Yes		alarms.xml

AppServer.configName	The application server input file could have several configuration. This property tells name of the configuration to be loaded.	No	default	Alarms
AppServer.alarmExport	The name of XML file into which BEAST configuration is exported. This is relative to the `config/AppServer/` folder. If missing, nothing is exported.	No		alarms_export.xml
AppServer.alarmConfigName	The name of BEAST alarm configuration, when exported.	No	ANKA Machine	ANKA Machine
AppServer.persistencyFile	The name of "persistancy" file, where values are stored for PV records, which are restored to last known value, when server is restored.	No	persistency.xml	persistency-alarms.xml

The extended set of additional properties:

Property	Description	Required	Default Value
AppServer.exportOnly	If True than application server just exports BEAST xml file and quits, does not start the server.	No	False
AppServer.passphrase	A secret word, which is used to authorise remote shutdown trough management interface.	No	please

4. The EPICS Records Definitions File

This is XML file defined with AppServer.input property (in server initialization file or System property).

In case of our Alarm server this is `alarms.xml`.

This records definition file can be validated with XML Schema found here: `config/AppServer/server.xsd`. This enables validation of the file and XML writing aides and tag-completion in advanced XML editor, like on used in CSS or Eclipse.

This is example snipped of the alarms.xml with simple alarm records.

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="server.xsd">
 <server name="Alarms">
  <management>
   <name>A:${host}:${server}</name>
  </management>
  <group name="SR PS Main" path="A:SR:PS:">
   <record>
    <name>MB-01:DiffCheck</name>
    <type>DBR_BYTE</type>
    <alarmConf>
     <path>Storage Ring,PS Diff Check</path>
     <description>PS setpoint/readback mismatch</description>
     <latching>true</latching>
    </alarmConf>
    <processor
instance="com.kriznar.csshell.epics.server.processor.SummaryAlarmProcessor">
     <input><links>A:SR:PSStatus:MB-01:Status</links></input>
     <gate><link>A:SR:OperationStatus:01:Mode</link><mask>0b001111</mask></gate>
    </processor>
   </record>
   <!-- ... more records -->
  </group>
  <!-- ... more groups -->
 </server>
 <!-- ... more servers -->
</config>
```

EPICS records are defined in to main ways:

- with XML element record
- or with XML element application.

The `record` element defines single PV with value provided by own ValueProcessor instance.

The `application` element loads an Application instance, which can further load several additional records trough server's API.

4.1. Element: config

This is XML Document root element. It defines the schema file. It can contain one or more `server` elements.

4.2. Element: server

Config element can contain one or more server elements. Server has one required parameter: name. When application server is started it loads server element, which name is provided as startup parameter.

The server element can contain `management` element and number of any of following elements: `record`, `group` or `application`.

4.3. Element: management

It is first subelement of server. It defines how server can be managed from remote trough set of PVs.

At minimum empty management tag must be present to enable management with default parameters. If there is no management tag, then management functionality is not activated.

The management element can contain following sub-elements:

Element (XML tag)	Description	Required	Default value

name	This name is prefix of PVs, which offer management access to this server. If omitted, default value is used: Name can contain two macro substitutions: - ${host} - is valid network host name of current machine, if this can not be resolved, then "localhost" is used - ${server}: is server name, as defined in server element with name attribute.	No	${host}:${server}
shutdown	Shutdown string is suffix part of management PV, which provides shutdown functionality for server. Full shutdown PV for server by default is: ${host}:${server}:Shutdown. To actually shutdown use passphrase as argument in CA set call.	No	:Shutdown
ping	Ping string is suffix part of management PV, which provides ping functionality for the server. Full shutdown PV for server by default is: `${host}:${server}:Ping.`	No	:Ping
list	List string is suffix part of management PV, which provides list functionality for the server. Calling this PV will return array of strings with all PV names hosted by the server. Full list PV for server by default is: `${host}:${server}:List.`	No	:List

4.4. Element: group

Basic function of the group element is to provide container for record elements and help provide hierarchical unique name for the records.

The group element can contain any of following elements: `record`, `group` or `application`.

Following attributes are supported for the group element:

Attribute	Description	Required	Default value
name	The name of the group, it should be human-friendly name, unique within server configuration.	Yes	
path	Path provides part of PV names for records withing the group. Full record PV name is assembled from hierarchy of previous paths and ends with the record name.	Yes	

4.5. Element: record

The record element is in many ways synonym to records as known in EPICS database. It has:

- name, which defined PV, it has
- properties, which are equivalent to record fields, it has
- processing part, which provides value.

For example full EPICS PV name of the record defined in the upper example is: `A:SR:PS:MB-01:DiffCheck`, which is combination of path and record name.

Here is table of sub-elements to the record element:

Element	Description	Required	Default value	Allowed values
name	Name of the record, becomes ending part of the record's PV name.	Yes		
type	DBR type of the record.	Yes		DBR_STRING, DBR_SHORT, DBR_FLOAT, DBR_ENUM, DBR_BYTE, DBR_INT, DBR_DOUBLE
description	Description of the record.	No		
persistent	If true then upon alarm application server restart the record will be initialized to last known value.	No	False	
count	Array size for the record's value.	No	1	
units	Units of the record's value.	No		
upperDispLimit	Value limits, as in CTRL DBR.	No		
lowerDispLimit	Value limits, as in CTRL DBR.	No		
upperWarningLimit	Value limits, as in CTRL DBR.	No		
lowerWarningLimit	Value limits, as in CTRL DBR.	No		
upperAlarmLimit	Value limits, as in CTRL DBR.	No		
lowerAlarmLimit	Value limits, as in CTRL DBR.	No		
upperCtrlLimit	Value limits, as in CTRL DBR.	No		
lowerCtrlLimit	Value limits, as in CTRL DBR.	No		
precision	The record's value precision.	No		
enumLabels	Comma separated list of names fro enum states.	No		
alarm	Initial alarm state of the record. It has no value but two further sub-elements.	No		

alarm.severity	Initial record alarm severity.	No	NO_ALARM	NO_ALARM, MINOR_ALARM, MAJOR_ALARM, INVALID_ALARM
alarm.status	Initial record alarm status.	No	NO_ALARM	NO_ALARM, READ_ALARM, WRITE_ALARM, HIHI_ALARM, HIGH_ALARM, LOLO_ALARM, LOW_ALARM, STATE_ALARM, COS_ALARM, COMM_ALARM, TIMEOUT_ALARM, HW_LIMIT_ALARM, CALC_ALARM, SCAN_ALARM, LINK_ALARM, SOFT_ALARM, BAD_SUB_ALARM, UDF_ALARM, DISABLE_ALARM, SIMM_ALARM, READ_ACCESS_ALARM, WRITE_ACCESS_ALARM
alarmConf	This elements contains other elements as they would appear in BEAST configuration XML. When BEAST XML is exported, then these elements are injected into it. Except for the path sub-element.	No		
alarmConf.path	Path is comma separated list of hierarchy node names in BEAST XML configuration, to which this record/alarm should be injected.	Yes		
alarmConf.description		No	value of records description element	
alarmConfType.enabled		No	True	
alarmConfType.latching	If BEAST alarm should latch.	No	True	

processor	The processor elements defines programming object, a Java class, which processes and provides value of the record. It has one attribute instance, which defines Java class name of that processor object. Sub-elements depend on selected processor object.	No		Following are recognized Java classes for known processors, which can be used for instance attribute without causing XML warning: com.kriznar.csshell.epics.server.processor.HostPingAlarmProcessor, com.kriznar.csshell.epics.server.processor.DefaultAlarmProcessor, com.kriznar.csshell.epics.server.processor.MemoryValueProcessor, com.kriznar.csshell.epics.server.processor.LinkedValueProcessor, com.kriznar.csshell.epics.server.processor.StatusCheckAlarmProcessor, com.kriznar.csshell.epics.server.processor.SummaryAlarmProcessor, com.kriznar.csshell.epics.server.processor.StateWatchdogProcessor, com.kriznar.csshell.epics.server.processor.ValueDiffCheckProcessor, com.kriznar.csshell.epics.server.processor.HeartbeatValueProcessor

4.5.1. Element: processor

Value processor is element inside the record declaration, which defines how the record value is created and processed.

For example a processor can check if a network ping to certain host returns OK, a processor will look something like this:

```
...<record ...>...
 <processor
instance="com.kriznar.csshell.epics.server.processor.HostPingAlarmProcessor">
  <trigger>60000</trigger>
  <host>acc-pc-lx01.anka-acc.kit.edu</host>
 </processor>
</record>...
```

The instance attribute define Java class name, which provides implementation of the processor and does the work.

The contents of <processor> element is used to initialize the actual processor instance and provide parameter for it's operation. In our example parameters are ping repetition trigger time 6000ms and remote host name to ping.

Processors in detail will be discussed in further sections.

4.5.2. A Record without Processor

It is possible to skip processor declaration when declaring record element. Such record will not be loaded into the server and there will be no PV to operate. But still the alarmConf element will be processed and corresponding BEAST configuration will be generated for this record.

This means that a record element without processor but with alarmConf element can be used as placeholder which help generate BEAST alarm configuration element for certain PV without starting a server record for this PV.

4.6. Element: application

The application element defines an application object, which is performs certain task with more than one record. Applications are not used directly in ANKA alarm server.

4.7. Advanced XML Concepts

Typical alarm definition file, such as alarms.xml contain a lot of similar or almost same record definitions, with only slight changes in names of records and PVs. This means a lot of copy&paste of same text. To make tasks of managing such file easier two concepts are introduced: substitution and template insertion. They simplify writing the XML file, but they make more complex cognitive model of the configuration.

4.7.1. Substitution

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="server.xsd">
 <server name="Alarms">
  <management>
  <group name="SR PS Main" path="A:SR:PS:">
   <substitutions>
    <status_alarm_desc>Device status indicates errors or inconsistencies - check
device panel!</status_alarm_desc>
   </substitutions>
   <record>
    <name>MB-01:Status:Alarm</name><type>DBR_BYTE</type>
    <alarmConf>
     <path>Storage Ring,PS Status
Check</path><description>${status_alarm_desc}</description>
    </alarmConf>
    <processor
instance="com.kriznar.csshell.epics.server.processor.StatusCheckAlarmProcessor">
     <!-- ... -->
    </processor>
   </record>
   <record>
    <name>MQ1-01:Status:Alarm</name>
    <type>DBR_BYTE</type>
    <alarmConf>
     <path>Storage Ring,PS Status
Check</path><description>${status_alarm_desc}</description>
    </alarmConf>
    <processor
instance="com.kriznar.csshell.epics.server.processor.StatusCheckAlarmProcessor">
     <!-- ... -->
    </processor>
   </record>
   <record>
    <name>MQ2-01:Status:Alarm</name><type>DBR_BYTE</type>
    <alarmConf>
     <path>Storage Ring,PS Status
Check</path><description>${status_alarm_desc}</description>
    </alarmConf>
    <processor
instance="com.kriznar.csshell.epics.server.processor.StatusCheckAlarmProcessor">
     <!-- ... -->
    </processor>
   </record>
  </group>
 </server>
</config>
```

How to make a substitution:

- Place `<substitutions>` element as first sub-element inside `<server>` or `<group>` element.
- Place <substitute_name1>, <substitute_name2>... as sub-elements to <substitutions> element, their value will be used as substitute for name.
- Where needed reference to a substitution use **`${substitut_name1}`** macro.

Scope of substitution is within the parent element (group or server). Substitution with same name, which is last defined, takes precedence.

Do not use for substitution names, which can be found inside it's scope as element or attribute names. This can cause close loops.

It is possible to use in substitution macro a name of another element, which is defined within same immediate parent.

There are two special build-in substitutions which are valid within each group:

- **`${path}`** - full path until current current point: it is combination of all paths from all groups within which substitution is found.
- **`${path1}`** - single path value of immediate group in which substitution is located.

4.7.2. Template Insertion

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="server.xsd">
 <server name="Alarms">
  <management>
  <group name="HostPing" path="A:GL:HostPing:">
   <substitutions><interval>60000</interval></substitutions>
   <group name="host_ping_template" template="true">
    <record>
     <name>${hname}-01:PingOK</name>
     <type>DBR_BYTE</type>
     <count>1</count>
     <alarmConf>
      <path>Infrastructure,Hosts</path><description>Host name resolution or ping
failed</description>
     </alarmConf>
     <processor
instance="com.kriznar.csshell.epics.server.processor.HostPingAlarmProcessor">
      <trigger>${interval}</trigger>
      <host>${hname}.anka-acc.kit.edu</host>
     </processor>
    </record>
   </group>

<group><substitutions><hname>acc-bpm-lib01</hname></substitutions><insert>host_ping_te
mplate</insert></group>

<group><substitutions><hname>acc-bpm-lib02</hname></substitutions><insert>host_ping_te
mplate</insert></group>

<group><substitutions><hname>acc-bpm-lib03</hname></substitutions><insert>host_ping_te
mplate</insert></group>
  </group>
 </server>
</config>
```

To make a template insertion:

- Make a group of repetitive records or applications at beginning of parent group.
- In records and applications replace declarations or parts of declarations that differ with macros.
- Tag group as a template by adding `template="true"` attribute and by providing unique name to the group.
- Insert templates with formula: `<group><substitutions>macro value</substitutions><insert>template name</insert></group>`.

5. Alarm Processors

In previous chapter we have defined alarm sources with the record configuration elements. Each record element, which want to process and fire alarms, must include processor sub-element.

The processor element defines Java object, which processes values and alarms and also provides configuration for this object.

The attribute named `instance` defines Java class name, which is instantiated by the application (alarm) server. This instance is than initialized with sub-elements of the processor element.

Subsections will make quick presentation of all registered alarm processors and their configuration options.

5.1. DefaultAlarmProcessor

This processor is parent processor for all alarm processors. This means all processors share functionality and configuration options of this processor.

Configuration Element	Description	Required	Default value	Allowed values
gate	A filter which let alarm trough or not based on operation state and mask.	Yes		
gate.mask	Gate mask is a integral value, where bits which matches with operation state value enable alarm propagation.	No	0b11111111111111111111111111111111	Value can be integer in decimal or binary form, where binary form must start with "0b" and then sequence of 0 and 1.
gate.link	A PV to the EPICS channel which provides operation state as enum DBR type.	Yes		

5.2. HostPingAlarmProcessor

This processor make network ping to remote host each time interval defined by trigger element in milliseconds. If ping fails, record value goes from 0 to 1 and alarm is raised.

This processor inherits from DefaultAlarmProcessor the gate functionality.

```
<record>
 <name>acc-bpm-lib01-01:PingOK</name>
 <type>DBR_BYTE</type>
 <alarmConf>
  <path>Infrastructure,Hosts</path>
  <description>Host name resolution or ping failed</description>
 </alarmConf>
 <processor
instance="com.kriznar.csshell.epics.server.processor.HostPingAlarmProcessor">
  <trigger>60000</trigger>
  <host>acc-bpm-lib01.anka-acc.kit.edu</host>
 </processor>
</record>
```

5.3. StateWatchdogProcessor

Value of this processor must be reset regularly within configured time period, if too many updates are missing, then processor raises an alarm. If remote process fails to update the value regulary, it is good indication that it might not function properly.

This processor inherits from DefaultAlarmProcessor the gate functionality.

```
<record>
 <name>SQLArchiver:Running</name>
 <type>DBR_BYTE</type>
 <alarmConf>
  <path>Infrastructure,Operation</path>
  <description>SQL archiver app running</description>
 </alarmConf>
 <processor
instance="com.kriznar.csshell.epics.server.processor.StateWatchdogProcessor">
  <trigger>60000</trigger>
  <valueOn>
   <severity>NO_ALARM</severity>
   <status>NO_ALARM</status>
  </valueOn>
  <valueOff>
   <severity>MAJOR_ALARM</severity>
   <status>STATE_ALARM</status>
  </valueOff>
  <monitor>
   <fails>3</fails>
   <resetValue>0</resetValue>
  </monitor>
 </processor>
</record>
```

5.4. ValueDiffCheckProcessor

Checks difference between setpoint and readback. If different is outside value and update time window, then alarm is raised.

This processor inherits from DefaultAlarmProcessor the gate functionality.

```
<substitutions><val>0.1</val><prec>0.01</prec></substitutions>
<record>
 <name>MB-01:Current:Check</name>
 <type>DBR_BYTE</type>
 <alarmConf>
  <path>Transfer,Extraction,PS Diff Check</path>
  <description>PS setpoint/readback mismatch more then
${val}+${prec}</description><latching>true</latching>
 </alarmConf>
 <processor
instance="com.kriznar.csshell.epics.server.processor.ValueDiffCheckProcessor">
  <device>A:EL:PS:MB-01</device>
  <value_window>${val}</value_window>
  <precision>${prec}</precision>
  <gate><link>A:SR:OperationStatus:01:Mode</link><mask>0b001111</mask></gate>
 </processor>
</record>
```

5.5. SummaryAlarmProcessor

This processor listens to one or more PVs and summarizes and filters alarms from those.

It can also summarize alarms in BEAST alarm tree by providing tree path.

This processor inherits from DefaultAlarmProcessor the gate functionality.

```
<record>
 <name>:SE:VOLT1:Al</name>
 <type>DBR_BYTE</type>
 <alarmConf>
  <path>Storage Ring,BPM Check,A:SR-S1:BPM:01</path>
  <description>Internal voltage levels out of limits!</description>
 </alarmConf>
 <processor
instance="com.kriznar.csshell.epics.server.processor.SummaryAlarmProcessor">
  <input><links>A:SR-S0:BPM:01:SE:VOLT1</links></input>
  <gate><link>A:SR:OperationStatus:01:Mode</link><mask>0b001111</mask></gate>
 </processor>
</record>
```

```
<record>
 <name>Infrastructure:Hosts</name>
 <type>DBR_BYTE</type>
 <description>Alarm tree summary over all host pings</description>
 <processor
instance="com.kriznar.csshell.epics.server.processor.SummaryAlarmProcessor">
  <input>
   <alarmPath>Infrastructure,Hosts</alarmPath>
  </input>
 </processor>
</record>
```

5.6. StatusCheckAlarmProcessor

This processor listens to status, which is stored as 0 or 1 bits in long value. Such as in mbbiDirect record. Status value is matched with on and off mask and alarm is raised if masked bit is in wrong state.

This processor inherits from DefaultAlarmProcessor the gate functionality.

```
<record>
 <name>MB-01:Status:Alarm</name>
 <type>DBR_BYTE</type>
 <alarmConf>
  <path>Storage Ring,PS Status Check</path>
  <description>Device status indicates errors or inconsistencies - check device
panel!</description>
 </alarmConf>
 <processor
instance="com.kriznar.csshell.epics.server.processor.StatusCheckAlarmProcessor">
  <input><links>ACS:PBEND_S.01:status</links></input>
   <maskOn>0b000001101101</maskOn>
  <maskOff>0b000000000010</maskOff>
  <alarmSeverity>MAJOR_ALARM</alarmSeverity>
  <alarmStatus>STATE_ALARM</alarmStatus>
  <gate><link>A:SR:OperationStatus:01:Mode</link><mask>0b001111</mask></gate>
 </processor>
</record>
```

6. Value Processors

The ANKA alarm server is by implementation quite generic. There are several value processors, which can be used in general way, with no special connections to the alarm system.

6.1. MemoryValueProcessor

This is most basic processor, stores in memory value of the record. It is also parent processor to all the rest of processors, also DefaultAlarmProcessor and trough this to all alarm processors.

6.2. HeartbeatValueProcessor

This processors listen to PV channel and fires values updates with configured period regardless the actual update rate. If this record is feed into CSS chart it makes chart look reasonable and helps setting value buffer length with stable time span. This processor also support simple polynomial value conversion.

6.3. LinkedValueProcessor

This processor listens to some PV channel and stores it's value.