MIRASYS
DEEP VISION DATA COMPANY

Mirasys Universal Data Driver

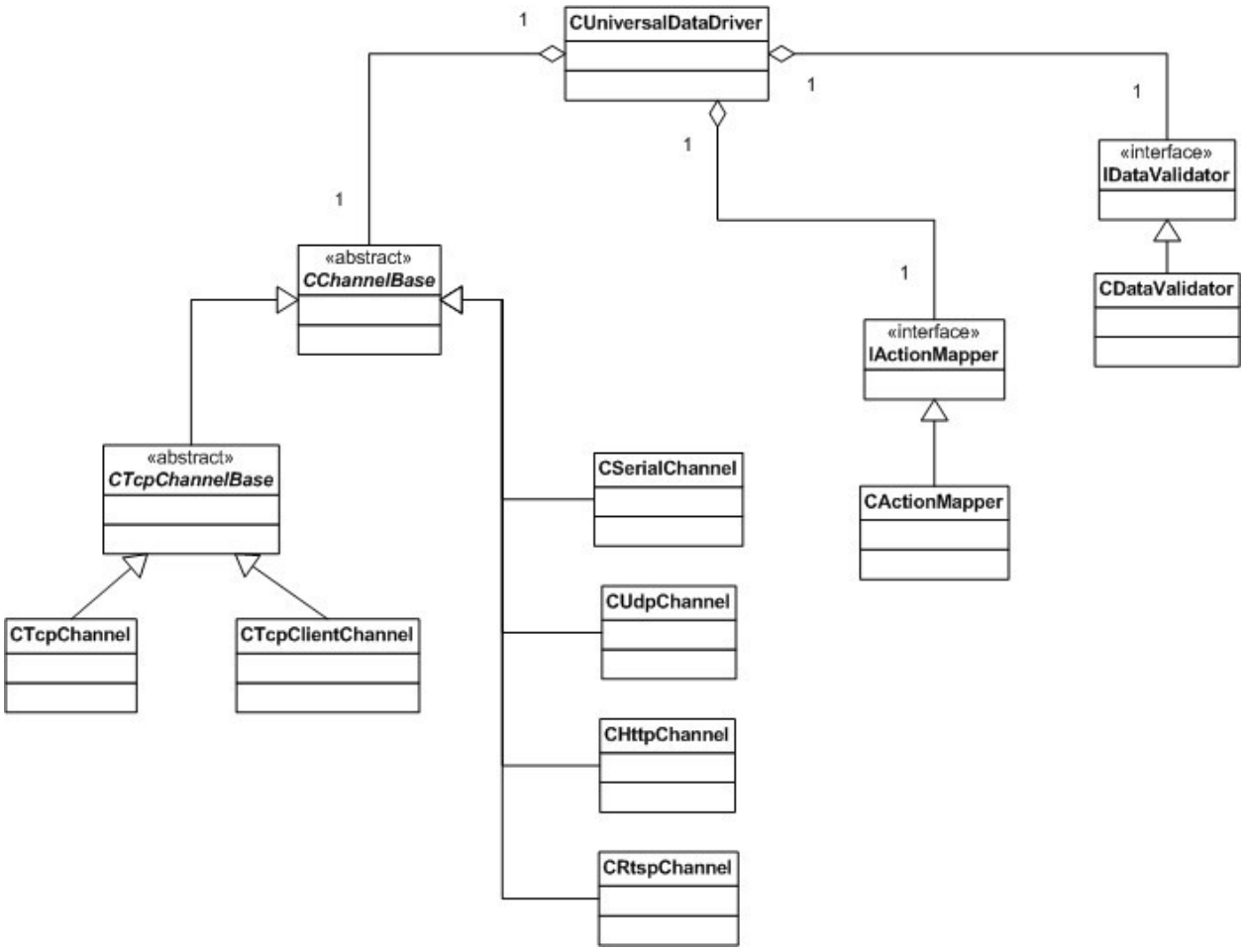
Table of Contents

1. Universal Data Driver guide

Universal Data Driver Idea

Universal data driver (UDD for short) is a configurable text data capture and transaction management interface for Mirasys VMS.
It has several communication interfaces to interact with text data sources, namely TCP/IP and HTTP client and server, and UDP/IP and serial (RS-232) server protocols. There is one special case for RTSP transaction messages. It is used for UDP technologies' IP camera VCA metadata event capturing.
Transaction messages can be either text strings (ASCII messages) or XML formatted messages.
Text format validation and parsing are done using regular expressions (Boost RegEx). XML messages are validated using XSD and parsing using XPath notation.
For binary messages and transaction sources requiring proprietary parsing and special communication protocol (for example SIA or ContactID protocols), there is a possibility to use custom validation (pre-processing) in custom DLL (dynamic link library).
All channels support UTF-8 and UTF-16 text as well as binary data formats (custom validation).
As a result transaction events can trigger alarms. By default, all transaction messages are stored in a text data channel.
Text data content can later be searched using the Spotter text search plugin and be used to synchronize video and audio playback.

COMMUNICATION LAYER CLASS DIAGRAM



DATA VALIDATION AND MAPPING INTERFACE

STRUCTURE OF CONFIGURATION FILE FOR SIMPLE TEXT ALARM DATA

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
```

This example describes validation and message mapping configuration for text data alarms that a Piccolo alarm system can send to the driver.

CHANNEL CONFIG

Channel Config section defines the **linefeed** value of the incoming messages, i.e. the message separator.

Typically Windows applications terminate messages to carriage return and linefeed (CR+LF = 0x0D0A). Unix systems (like IP cameras) typically use only linefeed (LF = 0x0A).
In some cases messages are not separated, i.e. source system is sending data in bursts.
In that case, the linefeed value should be left empty (“”) and use text type parsing to match the incoming character stream.
UDD channel reader can ignore selected characters or byte values to clean the data and help to parse.
Tag **ignored** lists all the characters that should be filtered out before passing them to the UDD parsing phase.
The most common ignored value is a NULL byte, i.e. 0x00.
When the text channel is active and the text data window is open in Spotter, all incoming messages are echoed in it.
If the user would like to control the view, it is possible to clear the window when needed.
This can be done by sending the **clearscreen** string to the text channel.

```
<channelConfig>
<linefeed value="0x0d0a"/>
<ignored value="0x00"/>
<clearscreen value="-----"/>
</channelConfig>
```

Validation

Different validation methods are described under a validation element.
The given attribute for regex contains a regular expression string to be used to validate text data (“*” at the end means 0 or many of the before defined character set).

```
<validation>
<regex value=".*"/>
</validation>
```

For XML formatted messages XSD (XML schema) validation type is used:

```
<validation>
<xsd value="MySchema.xsd"/>
</validation>
```

If the .xsd file is defined, this file will be used for incoming XML message validation. But the "value" attribute can be empty - in this case, the common validation will be used: only checking that the incoming message has the correct XML format.

Log messages in the DVRLog.txt

Detailed logging mode can be enabled by inserting the following option in the XML configuration file.
Logging level parameters are:

- 0: no writing to dvrlog.txt
- 1: only error messages, error and info messages.

To enable additional message writing and input data packet storing, you should add

<additionalDebug> tag to logging element with the attribute value="yes".
Data packets read into the text channel will be written into a DVR folder file named “Debug_UDD_Packets.bin”.
Additional text channel info will be written into the file “Debug_UDD_Log.txt” in the same folder.

```
<logging>
<level value="2"/>
<additionalDebug value="yes"/>
</logging>
```

MESSAGE CONTENT PARSING – UDDXMLMAPPER ELEMENT

Transaction event message content parsing is defined in **uddXmlMapper** element.
Incoming messages can be ASCII strings or well-formed XML.
Binary messages will need pre-processing before UDD can handle them.
Typically pre-processing from binary format to internal XML format is done in a separate proxy service (for example PaxtonProxy or BoschIVAproxy) or in UDD custom validator (for example GalaxyDataValidator.DLL or TexecomDataValidator.DLL).

```
<uddXmlMapper version="2">
```

Message definitions – messageType element

Inside uddXmlMapper element first sub-element is **messageType**.
The message type element defines the incoming message content format (ASCII **text** or **XML).**

```
<messageType value="text" parsing="regex">
```

Inside message type elements are messages with their content references. ASCII text parsing tagging is defined using regular expressions and XML parsing using XPath notation to reference tags (“+” at the end of a group means 1 or many of the before defined character set).

```
<messageType value="text" parsing="regex">
<message number="1" value="alarm">
<param number="1" value="([a-zA-Z]+) (.+)" group="1"/>
<param number="2" value="([a-zA-Z]+) (.+)" group="2"/>
</message>
</messageType>
```

In the above example, ASCII text has three interesting parts; a string of letters, a string of any characters and a string of numbers.
Strings are separated by spaces.
If the incoming message would be (followed by the defined line feed)

then

group=”1” match “start” group=”2” match “low” group=”3” match “4”
rest of the message would be ignored from a matching point of view.
If the incoming XML message having the same kind of information would be (followed by the defined line feed)

```
<?xml version="1.0" encoding="UTF-8"?>
<alarm>
    <rule>start</rule>
    <type level="4">low</type>
    <info>this is normal</info>
</alarm>
```

then
“alarm/rule” match “start” "alarm/type" match “low” "alarm/type/@level" match “4”
Note that XML tag attributes are denoted using @ sign in front of the attribute name.

Message content constants – Constants element

These constant arrays contain values that can be used in the message handling rules section.

```
<constants>
<array name="Rules" params="AT">
<value>start</value>
<value>stop</value>
</array>
<array name="AlarmType" params="AT">
<value>ACCESS CONTROL ALARM 1</value>
<value>ACCESS CONTROL ALARM 2</value>
<value>ACCESS CONTROL ALARM 3</value>
</array>
</constants>
```

In this example, incoming messages contain “Rules” strings that have values “start” and “stop”.
Params “AT” and “TL” mean that these values will be used at least in two different kinds of rule definitions in the UDD rules element.

Action rules - Rules element

Action rules, i.e. how to handle incoming transaction messages, are defined in Rule's element.
There can be more than one type of message and its rules, but typically UDD is configured to handle messages only in one format.
However, one message format usually has several rules and events defined.

```
<rules>
<message number="1" value="alarm" alwaysShowText="yes">
<param number="1" reference="1" value="$Rules" operator="eq" id="AT">
<and number="1" reference="2" value="$AlarmType" operator="eq"/>
<action number="1" type="event" value="$Rules $AlarmType alarm" />
</param>
</message>
</rules>
```

In this example <message number="1" value="alarm" refers to the messageType element and its message definition.

Note, that in text format messages the message tag attribute in messageType has to have a name (value="alarm").
In XML format messages this value is by default name of the root tag, in this example alarm (value="**alarm**/rule").
Message tag attribute alwaysShowText="yes" means that all incoming messages will be shown in the text channel device window.
Message tag attribute handleParameters=" all" means that for the incoming message all rules are run through. By default, UDD stops to the first matching rule.
If there were more than one message format handled in the rules element, then message elements will have an ascending number attribute number="1", number="2", etc.
Rules for the message are defined in the param element. Param elements have an ascending number attribute.
Param element reference attribute is a sequence number of message tags (see messageType/message element above, in the example group="1" match "start" or "alarm/rule" match "start"). So in the example above reference="1" would contain the value "start".
Param tag attribute can be either constant value="start" or a reference to constant array (in constants element) value="$Rules".
Using constant arrays UDD automatically expands rules using the values of constant arrays during runtime.
So, the operation would be the same (in the above example) either using arrays (Rules array containing strings "start" and "stop") or manually configuring two rules params

<param number="1" reference "1" value="start" operator="eq">
<param number="2" reference "1" value="start" operator="eq">

Comparison operators that can be used as Boolean operators are:

eq	== [default]
gt	>
ge	>=
lt	<
le	<=
neq	!= not equal
contains	finds configuration parameter value as substring in the incoming message parameter value
contained	finds incoming message parameter value as substring in the configuration parameter value, this operator is supported in the 2.13.6.0 version of the UDD or higher

Each parameter can have the "type" attribute which is used for correct text values conversion for comparing. Currently, the following types are supported:

- type="integer"
- type="float"

- type="string"

If the "type" attribute is not specified, the "string" type is used by default.
The rule per message can contain one or more parameters, and these parameters can refer to one or more parameters defined in the messageType element as described above.
This example configuration XML for a simple text alarm data has only one parameter per message, so message parameters in message rules refer to parameter 1.

```
    <rules>
      <message number="1" value="Alarm cam001">
        <param number="1" reference="1" type="string" value="Alarm
cam001">
```

Zero or more actions can be associated with message parameters. Currently, defined actions are "event"
to trigger events, "data" to send data, and "metadata" to send metadata to DVR.

```
          <action number="1" type="event"  value="Alarm cam001" />
          <action number="2" type="data"  value="Alarm cam001 data" />
          <action number="3" type="metadata"  value="" />
```

For metadata one additional attribute can be used - "metadatatype". If it is specified it allows to configure metadata type:

- "xml" - XML metadata format
- "udpvca" - special UDP VCA metadata format
- "textevent" - text event metadata format
- "other" - other metadata formats

If the "metadatatype" attribute is not specified the default text channel format is used.
The parameters can also have an associated search tag. If a search tag type attribute has been omitted, it is set by default to be SEARCH_TAG_TYPE_TEXT.

```
          <searchtag value="Alarm cam001" />
        </param>
      </message>
      <message number="2" value="Reset 001">
        <param number="1" reference="1" type="string" value="Reset
001">
          <action number="1" type="event"  value="Reset 001" />
          <searchtag value="Reset 001" />
        </param>
      </message>
      <message number="3" value="Alarm cam002">
        <param number="1" reference="1" type="string" value="Alarm
cam002">
          <action number="1" type="event"  value="Alarm cam002" />
          <searchtag value="Alarm cam002" />
        </param>
      </message>
      <message number="4" value="Reset 002">
        <param number="1" reference="1" type="string" value="Reset
002">
          <action number="1" type="event"  value="Reset 002" />
          <searchtag value="Reset 002" />
        </param>
      </message>
    </rules>
  </uddXmlMapper>
</root>
```

So what this example configuration will do is to tell the driver that:

- we are expecting incoming data to be textual
- text data must match with the defined regex pattern
- there are four possible message types

For example, if the incoming data is

```
Alarm cam001
```

it is validated against regex and matched against defined message types.
The match is found with the defined message number 1, so the driver will send an event and "Alarm cam001" data to DVR.
If the incoming data would be

```
Alarm cam100
```

it would validate against regex but no matching rule is found, so it would be ignored.
And if the incoming data would be

```
Alarm #%&%#!
```

it would not validate against a defined regex pattern and would generate an error in the DVR log file.

WHEN STARTING THE INTEGRATION TEST

It is a best practice that before taking the transaction message validation and parsing configuration into use, first test the communication between the two systems.
This way all captured messages will be written directly into the text channel.
Select validation mode "None" in the channel configuration dialogue, start transaction counterpart and monitor text channel device window in Spotter user interface.

ADDITIONAL PARAMETERS FOR HTTP SERVER

Additional parameters can be configured for the HTTP server channel in the XML configuration file.
These parameters should be placed to the special section <HTTPServer></HTTPServer>. Currently, the following parameters are supported:

- "processingType" attribute - the special attribute to determine what part of incoming HTTP request will be processed:
  - All - whole message
  - Headers - only HTTP headers from the incoming message
  - Uri - only HTTP URI string from the incoming message
  - Content - only HTTP content from the incoming message
- <Response> element - the HTTP response message content.
  - If the attribute "send" value is no, the driver will send an HTTP response message without content
  - If the attribute "send" value is yes, the driver will send an HTTP response message with specified content data

Example of additional parameters configuration for HTTP server channel:

```
---
<Additional>
  <HTTPServer processingType="All">
    <Response send="true">Response string to send</Response>
  </HTTPServer>
</Additional>
---
```

- type="integer"
- type="float"
- type="string"

If the "type" attribute is not specified, the "string" type is used as default.

```
        <param number="1" reference="1" value="$Rules" operator="eq" id="AT">
            <cmd number="1" reference="2" value="$AlarmType" operator="eq"/>
            <action number="1" type="event" value="$Rules $AlarmType alarm" />
         </param>
```

It is possible to combine more than one content reference using Boolean operator AND. In the example
message content reference="1" (“alarm/rule”) and reference="2" ("alarm/type") must both be true (“and” operator) in order to trigger the action event value="$Rules $AlarmType alarm".
If the message content would have been for example “alarm/rule” matching “start” and "alarm/type" matching “low”, then the trigger “start low alarm” would have been triggered.

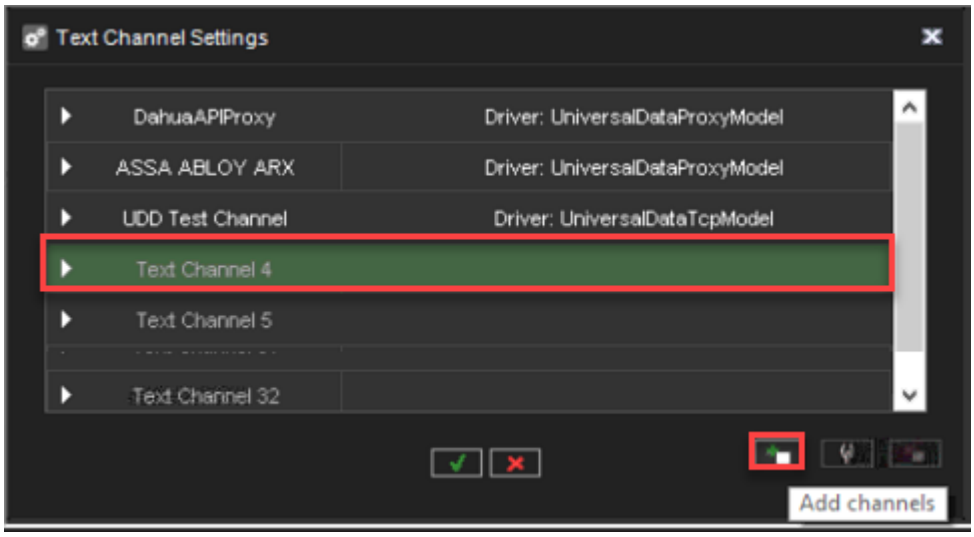
2. How to configure UDD text channel

Create UDD configuration file(s); for text validation UDD configuration XML and for XSD validation both UDD configuration file and XSD validation schema.
Hint: good online XML tools can be found from http://www.freeformatter.com/xml-validator-xsd.htmlhttp://www.freeformatter.com/xsd-generator.html
Copy configuration file(s) into DVR folder (for example C:\Program Files (x86)\DVMS\DVR.

Adding text channel to the Mirasys VMS

1. Open **System Manager**
2. Go to the **VMS servers** tab
3. Open **Text channels**

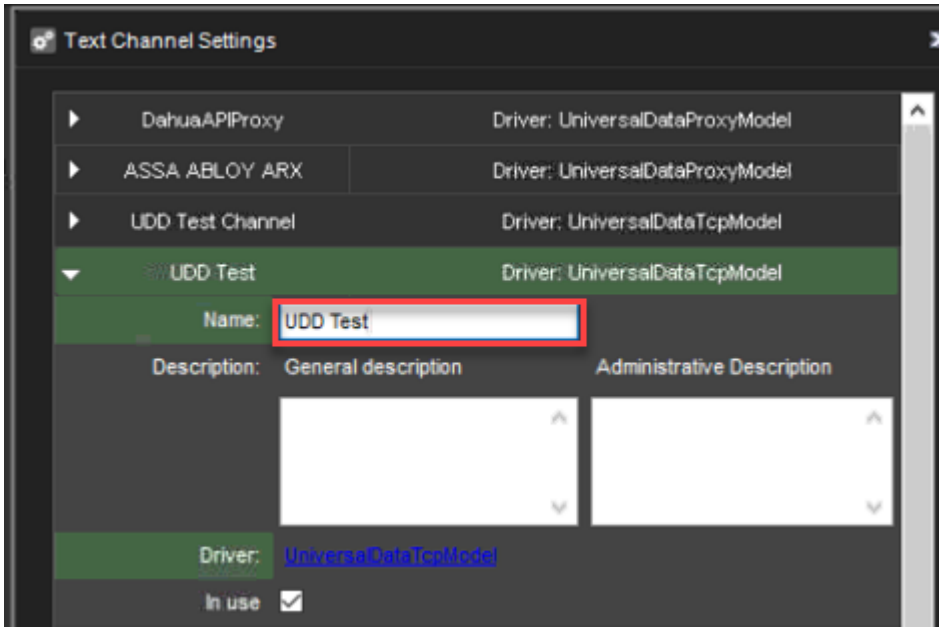
4. Select first free text channel
5. Click **Add channel**



6. Set **Model** to **UniversalDataTcpModel**
7. Enter needed **TCP Port number**(default **40000**)
8. Set **Validation** to **Text**
9. Click **OK**

10. Enter the name of the text channel and press **OK**.

Mirasys Universal Data Driver



Editing UDD4Demo.xml

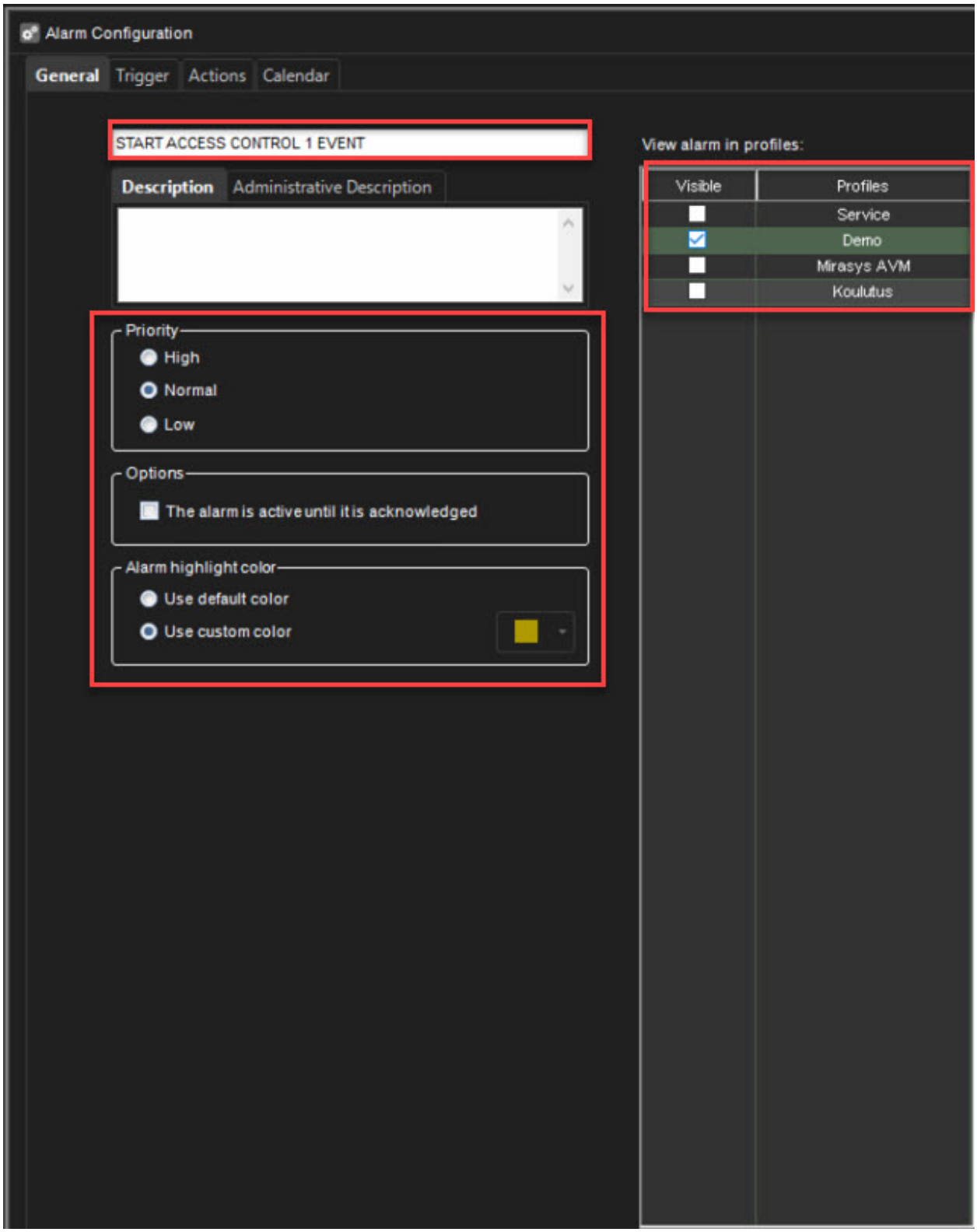
1. Open text editor with admin rights
2. Browse to **C:\Program Files\DVMS\DVR**
3. Select **UDD4Demo.xml** file
4. Edit
   <array name="Rules" params="AT">
   <value>START</value>
   <value>STOP</value>
5. Edit
   <array name="AlarmType" params="AT">
   <value>ACCESS CONTROL 1 EVENT</value>
   <value>ACCESS CONTROL 2 EVENT</value>
   <value>ACCESS CONTROL 3 EVENT</value>
6. Save changes and close text editor
7. Open created text channel and rename text channel and click **OK(**Mirasys VMS reloads changes from UDD4Demo.xml**)**

In this example, we have edited the UDD4Demo.xml file so that when the below text appears, that will be used as a trigger of the alarm.
All other texts are ignored.

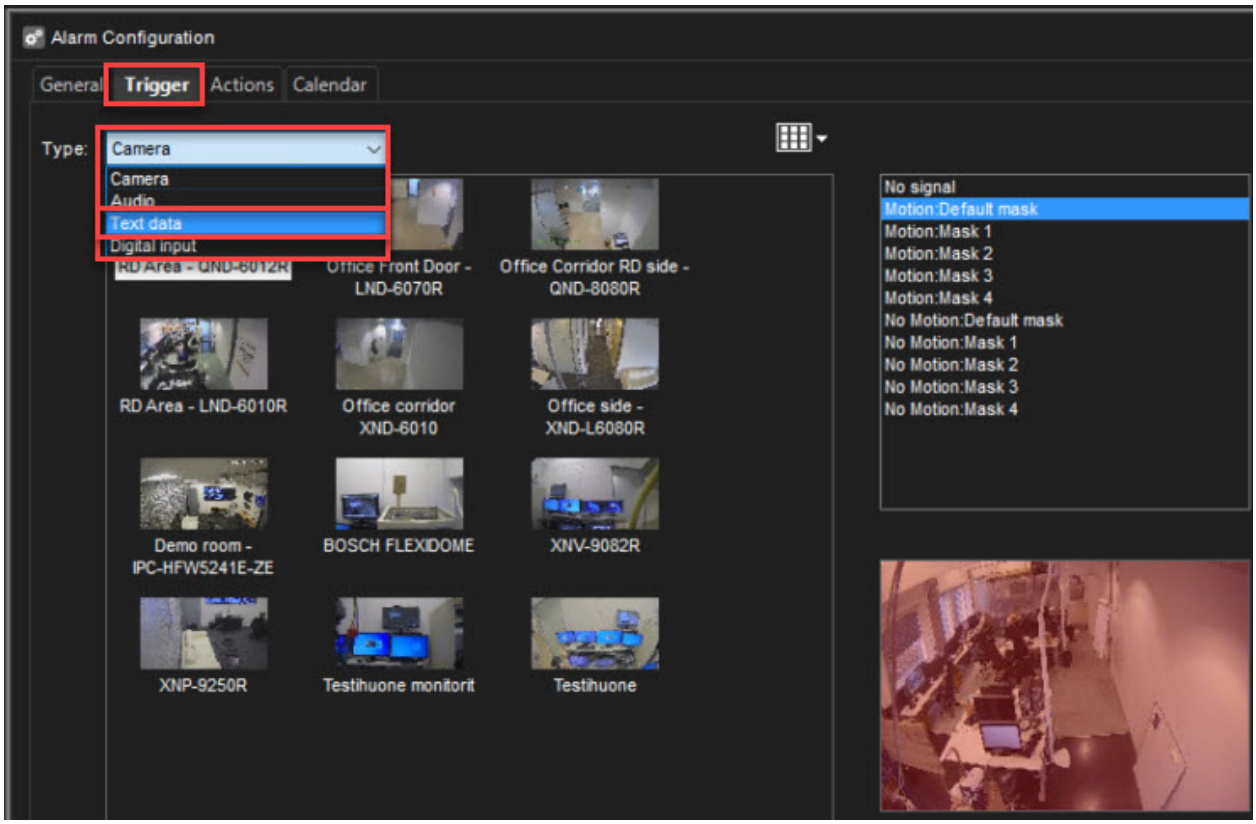
START ACCESS CONTROL 1 EVENT
START ACCESS CONTROL 2 EVENT
START ACCESS CONTROL 3 EVENT

Creating an alarm from the validated text

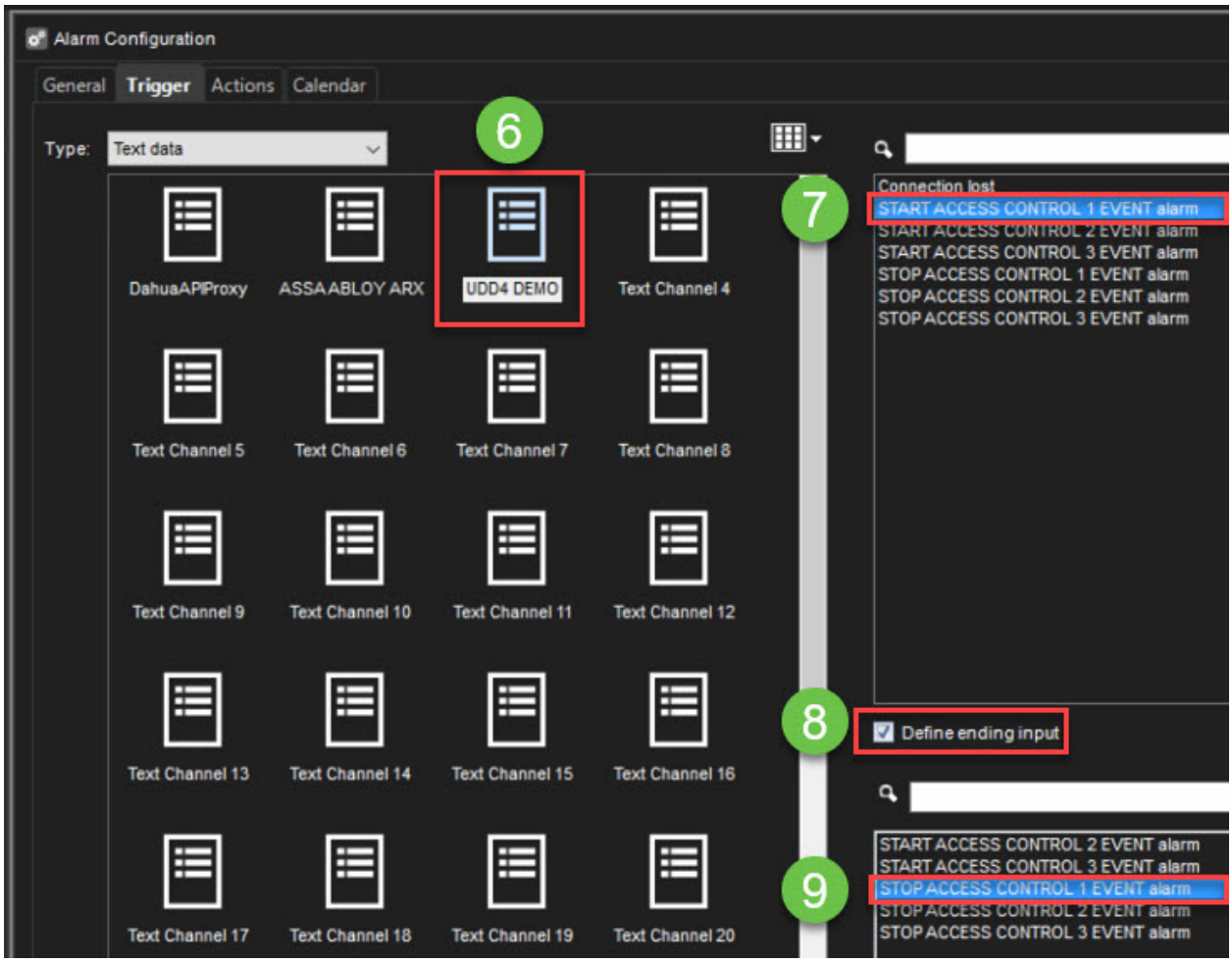
1. Go to the **VMS Servers\Alarms**
2. Click **New Alarm**
3. Enter the name of the alarm(in this example we are using **START ACCESS CONTROL 1 EVENT, START ACCESS CONTROL 2 EVENT and START ACCESS CONTROL 3 EVENT**)
4. Set **Priority** and **View alarm in profiles**



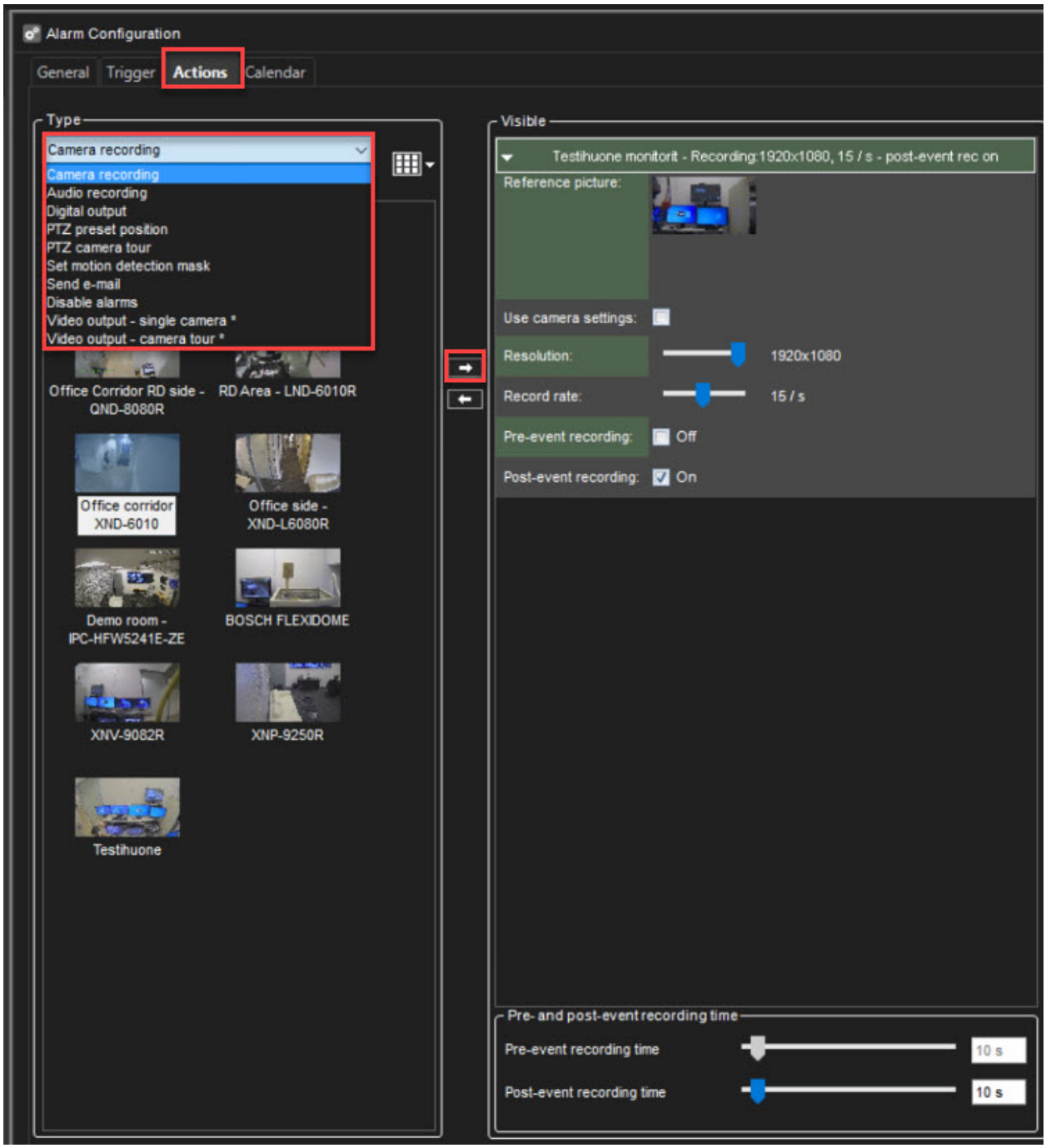
1. Open **Trigger tab**
2. Select **Text data** from the dropdown menu



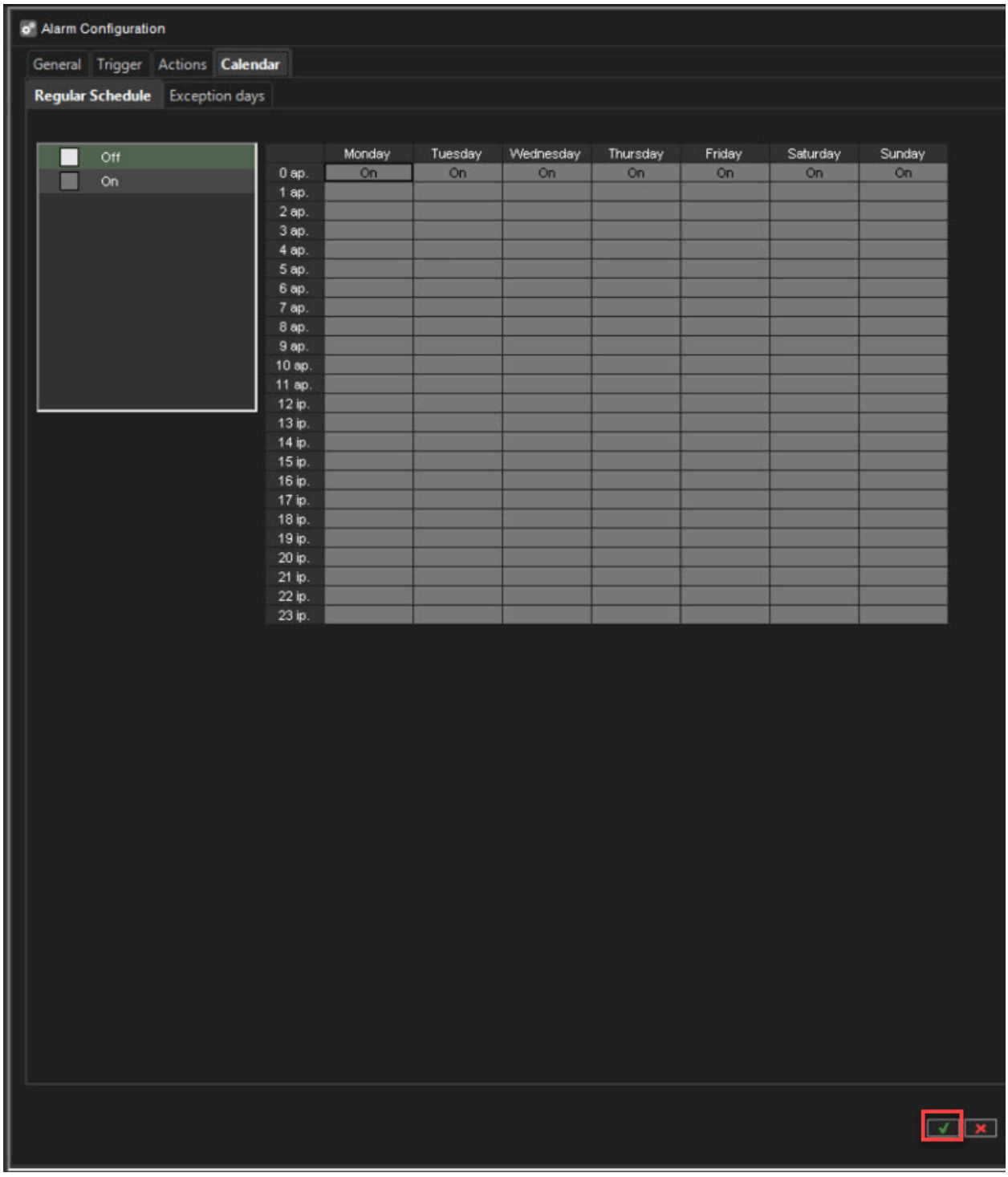
6. Select correct text channel
7. Select alarm trigger from the upper box(this example **START ACCESS CONTROL 1 EVENT)**
8. Enable **Define ending input,** if needed(If Define ending input is set, the alarm will be active until selected input text is received to the text channel)
9. Select correct ending input(this example **STOP ACCESS CONTROL 1 EVENT)**



1. Open **Actions** tab
2. Select actions of the alarm from the dropdown menu and add them to the **Visible** box by clicking **Add**

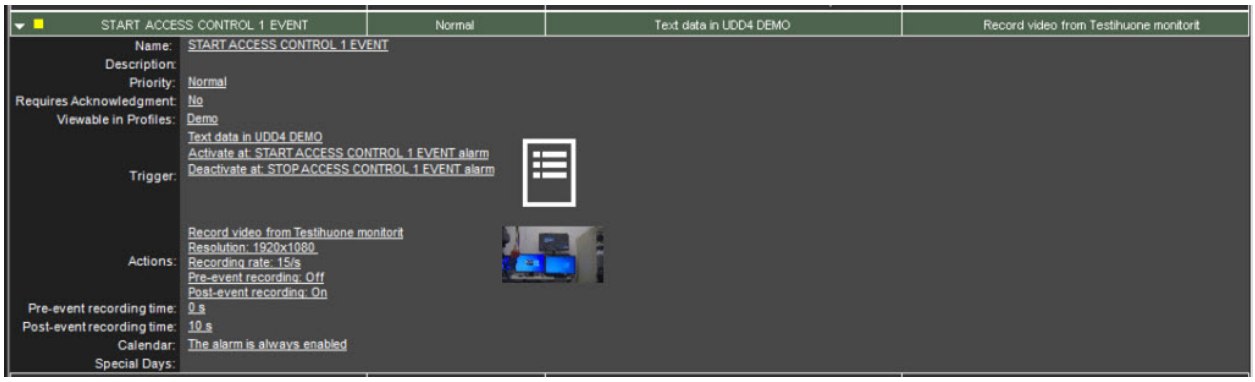


1. Open **Calendar**
2. Define, alarm active hours
3. Click **OK**



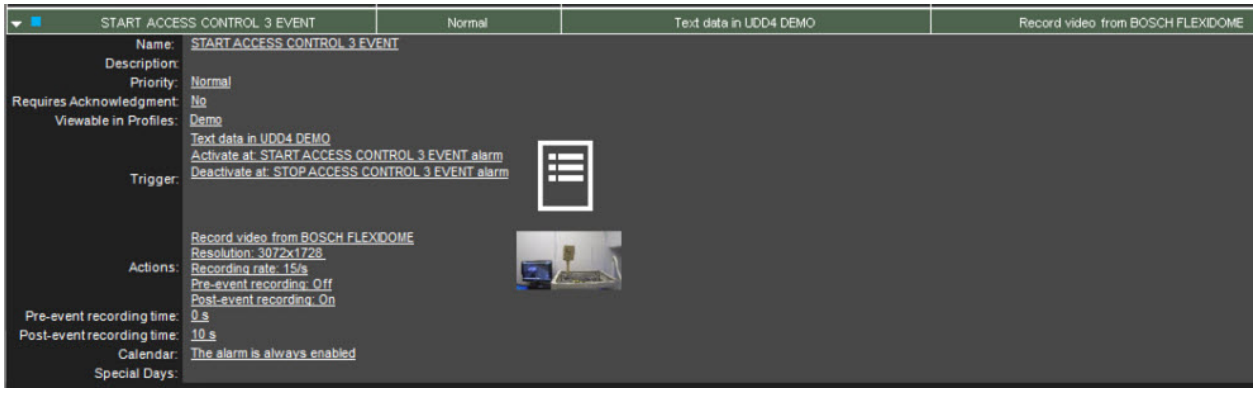
We have now created a separate alarm for each event. Below is an overall view of each alarm.

START ACCESS CONTROL 1 EVENT



START ACCESS CONTROL 2 EVENT

START ACCESS CONTROL 3 EVENT

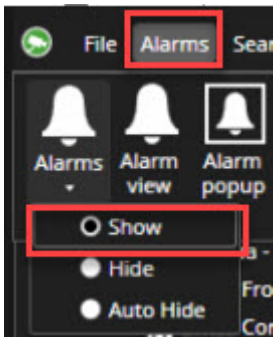


4. Click **OK** to confirm alarm creation

Testing text channel alarm

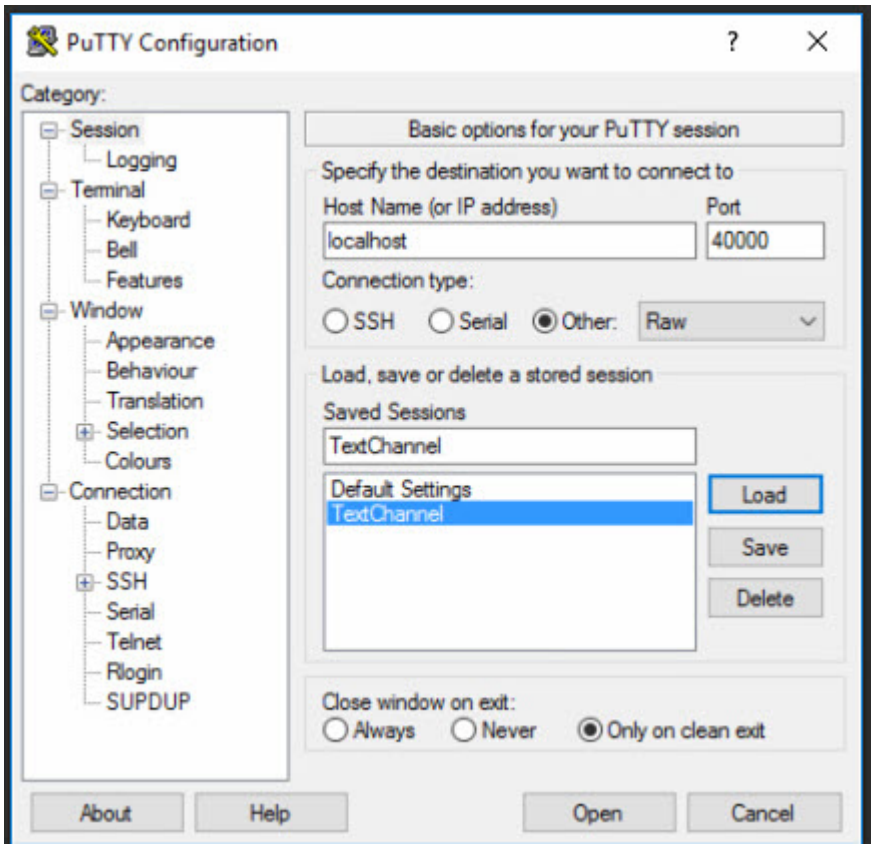
1. Start Spotter
2. Click **Alarms\Alarms** and select **Show**

Mirasys Universal Data Driver



1. Open used text channel to the work area

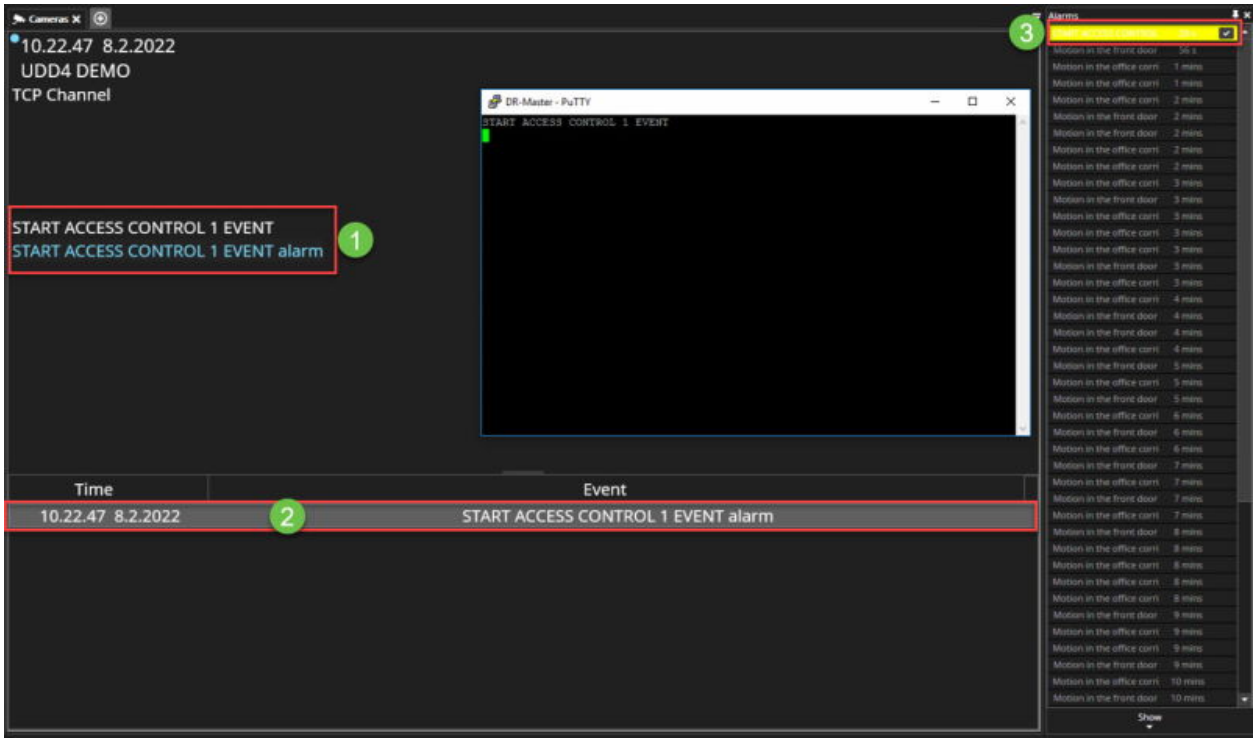
1. Download PuTTY
2. Install Putty and start the application
3. Set correct information
4. Click **Open**



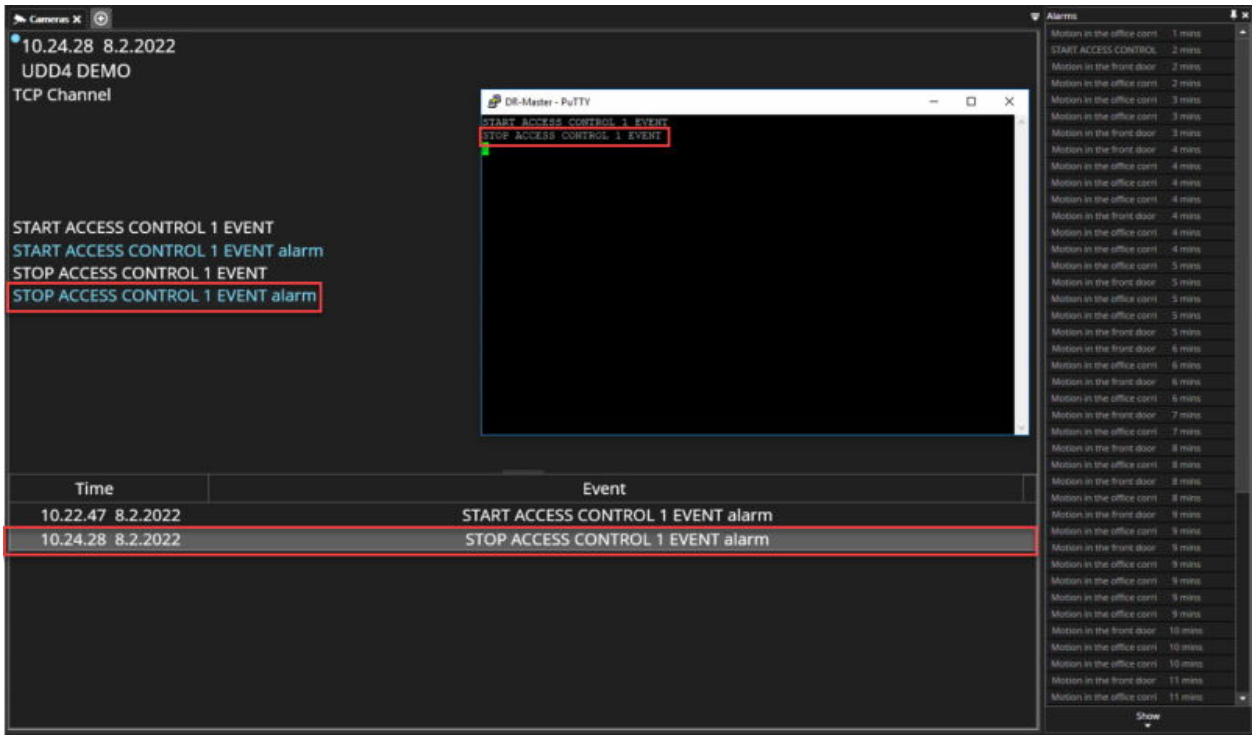
5. Type needed text to the PuTTY
6. Press **Enter**

Mirasys Universal Data Driver

1. Text channel shows received information
2. Event name is shown in the list
3. Alarm list show active alarm

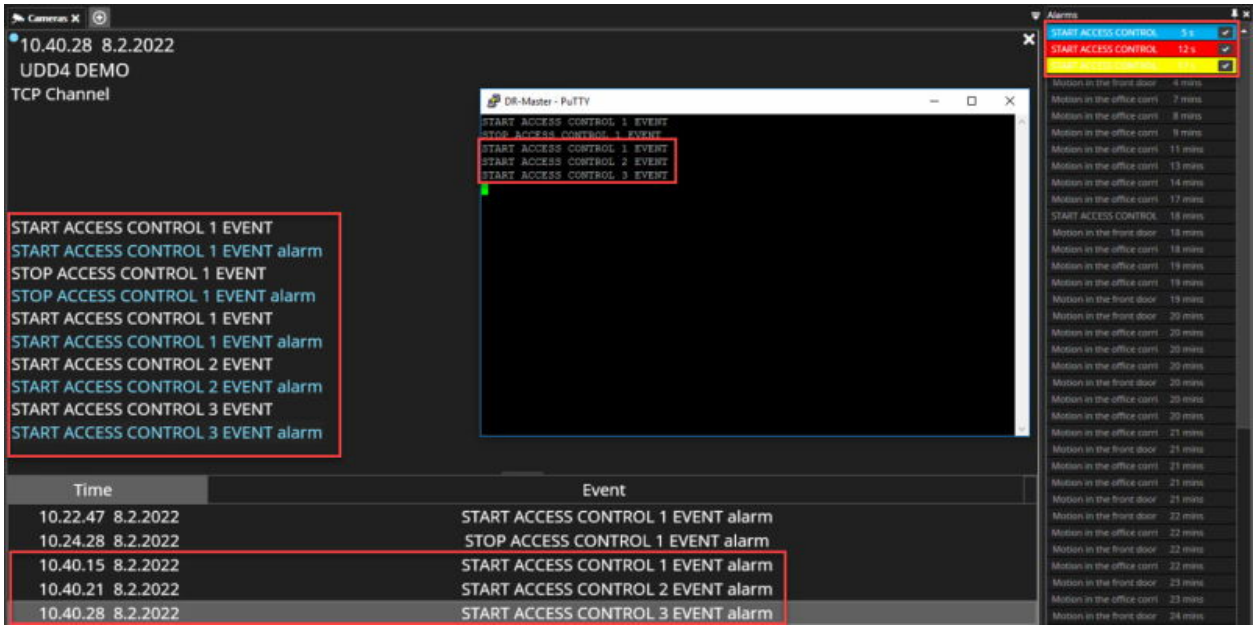


When the text channel receives ending input data, the data is shown in the text channel and alarm is ended



Mirasys Universal Data Driver

The last example shows that the text channel has received validated data



3. UDD4Demo.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
<logging>
<level value="2"/>
<additionalDebug value="no"/>
</logging>
<channelConfig>
<linefeed value="0x0d0a"/>
<ignored value="0x00,0x0B"/>
<clearscreen value="-----"/>
</channelConfig>
<validation>
<regex value=".*"/>
</validation>
<uddXmlMapper version="2">
<messageType value="text" parsing="regex">
<message number="1" value="alarm">
    <param number="1" value="([a-zA-Z]+) (.+)" group="1"/>
    <param number="2" value="([a-zA-Z]+) (.+)" group="2"/>
   </message>
</messageType>
  <constants>
    <array name="Rules" params="AT">
<value>start</value>
<value>stop</value>
</array>
    <array name="AlarmType" params="AT">
<value>temp</value>
      <value>normal</value>
<value>low</value>
</array>
</constants>
<rules>
    <message number="1" value="alarm" alwaysShowText="yes">
<param number="1" reference="1" value="$Rules" operator="eq" id="AT">
        <and number="1" reference="2" value="$AlarmType" operator="eq"/>
         <action number="1" type="event" value="$Rules $AlarmType alarm" />
</param>
    </message>
</rules>
</uddXmlMapper>
</root>
```

MIRASYS
DEEP VISION DATA COMPANY

Mirasys Universal Data Driver

Mirasys Ltd - C1CD, Vaisalantie 2-8, 02130 - Espoo, Finland
Tel +358 (0)9 2533 3300 - info@mirasys.com - www.mirasys.com