A workshop for SysMon and Zabbix 2.2.8 for seamless auxiliary data

Possible system monitoring for VLBI

May 2015

IVS Task Force

for

Seamless Auxiliary Data

by

Alexander Neidhardt

Jim Lovell

IVS products are related to additional, auxiliary data from the different participating sites and telescopes. Mainly meteorology and time offsets are currently in use but these data are not continuously available. Furthermore, other data might be of value for future VGOS observations. The IVS Task Force on Seamless Auxiliary Data should show possible realizations, make suggestions on what data should be provided and how observatories can contribute to the real-time data stream.

Purpose

Currently most of the additionally acquired, auxiliary data are only available in the form of log file entries during a dedicated observation. In some cases it would be much better for analysis purposes to have continuous data available outside these periods instead of estimations. Such data are also interesting for a general overview of the current network situation or for the real-time correlation purposes.

Furthermore, many of the additional, local data sets, which are available at some observatories and which would be of interest for research purposes, are not directly available (e.g. Invar strain meter measurements in the radio telescope Wettzell). Therefore in some cases it would be helpful to have the possibility to include such additional, partly optional, local data into research scenarios.

Justification

The main products of the IVS rely on additional, auxiliary data such as meteorology and time corrections between local clocks and UTC. These data are part of the log files, which are produced during the observation. All times without observations must be interpolated or extrapolated from the existing data sets. Therefore it would have some positive effects on the accuracy of IVS data products if data were to be continuously available.

Real-time ancillary data can also contribute in a dynamic observing scenario where scheduling decisions are automatically made based on changing situations at the observatories. If for example a telescope cannot observe due to high wind or an unexpected equipment failure, the schedule could be adapted in real-time to optimize the observations by excluding or down-weighting the affected station. This can only happen if a centralized real-time repository of auxiliary data is established.

Terms of Reference

The main objective can be split into two parts:

a) An initial proof-of-concept realization through data collection from a limited number of observatories with a view to expanding data collection procedures to as many observatories as possible (focus of the proposed group members)

b) the collection (and definition) of possible future auxiliary data
(focus of the proposed group members together with the proposed correspondents)

To enable the proof-of-concept realization using a centralized data repository, a server system with a suitable RAID-set of hard drives should be bought by the Forschungseinrichtung Satellitengeodäsie, Technische Universität München. It will be installed at the Geodetic Observatory Wettzell. The server should enable the receiving of incoming data sets in different formats and on different access points (e.g. as FTP-files or as real-time data streams on IP-sockets). It should also have a suitable database to administer the time-tagged data and a Web server to access current values, but also historic data files.

In a first development process, e-RemoteCtrl should be extended to send data from time-to-time to the central repository, so that continuous data streams are available, as long as the NASA Field System with e-RemoteCtrl is active. This step simplifies the installation of suitable software at the different sites at the proof-of-concept stage.

In close cooperation with the developers at the AuScope network, Australia, access points and interfaces should be realized to enable the different monitoring systems, like MONICA in Australia and SyMon in Wettzell, Germany, to send real-time streams to the central repository at Wettzell. The advantage of this approach is that it is independent of the NASA Field System and therefore from the online times of the Field System PC.

Further specifications and potential data requirements should be discussed and defined during telecon meetings. The focus will be on enabling dynamic observations in real-time and in creating the possibility of automated processing of observation data scheduling through to analysis. Another goal is the creation of a transition and realization plan for other IVS telescopes after the proof-of-concept test phase has been completed.

Desired Outcomes

1) A first standardization of an interface for a central data repository to receive incoming data and to fetch selected data sets from a data acquisition server.
2) The hardware and realization of a central repository at the Geodetic Observatory Wettzell
3) An initial proof-of-concept implementation with software and hardware components at selected observatories, which are able to send auxiliary data (focused on meteorological data) to a central repository.
4) A list of data, which may be relevant for future VGOS observations.
5) A transition and realization plan for IVS telescopes

Timeline

Currently 2 years are proposed for the realization of the desired outcomes. The first year focuses on the realization of the central data repository at the Geodetic Observatory Wettzell and the realization of the e-RemoteCtrl extension. The second year focuses on the adaption of real-time streams to existing system monitoring suites, like the Australian MONICA and the SysMon of the Wettzell observatory. During the whole period of the task force, the new specifications should be discussed, so that a report with the suggestions can be offered at the end of the 2 years.

Proposed Group Members

Chair: Alexander Neidhardt (IVS, Germany)
Vice-Chair: Jim Lovell (IVS, Australia)
Ed Himwich (IVS, USA)
Jamie McCallum (IVS, Australia)
Christian Plötz (IVS, Germany)
Jonathan Quick (IVS, South Africa)
Matthias Schönberger (BKG, Germany)

Proposed Group Correspondents

Johannes Böhm (IVS, Austria)
John Gipson (IVS, USA)
Rüdiger Haas (IVS, Sweden)
Arthur Niell (IVS, USA)
Axel Nothnagel (IVS, Germany)
Bill Petrachenko (IVS, Canada)
Lucia Plank (IVS, Australia/Austria)

Planned Operations

Telecoms should be arranged, at least each second month to discuss the required auxiliary data sets. Selected test sites should be equipped with the e-RemoteCtrl extensions to enable the data sending in the first phase of the test-bed realization. Stations, which are already equipped with MONICA or SysMon, should be extended with the real-time stream and should participate to the central repository.

Wettzell SysMon

(a solution for

the MCI-suggestion)

by

Chris Beaudoin (MIT Haystack)
Martin Ettl (FESG/TUM, MPIfR)
Ed Himwich (NASA, GSFC, NVI)
Alexander Neidhardt (FESG/TUM)
Matthias Schönberger (BKG)

VLBI2010 Monitor and Control Infrastructure (MCI) Working Document from the MCI Collaboration Group

represented by
Chris Beaudoin (MIT Haystack)
Martin Ettl (FESG/TUM, MPIfR)
Ed Himwich (NASA, GSFC, NVI)
Alexander Neidhardt (FESG/TUM)
Matthias Mühlbauer (BKG)

Summary: This document is intended to give direction to the efforts put towards realizing a generalized VLBI2010 Monitoring and Control Infrastructure (MCI) system based on the notes outlined in the MCI collaboration group meeting held in Bonn Germany on March 28, 2010. This working document is currently open to discussion among the members of the committee but after discussion and revisions it should be frozen into a final specifications document.

1. Introduction

The VLBI2010 Monitor and Control Infrastructure (MCI) is being developed with the intent to promote uniformity of MCI throughout the next-generation IVS network stations. The key ideas form the basis upon which the VLBI2010 MCI architecture should be developed are:

(1) The architecture and software should be open-source and open-ended, so that stations are free to use modify and adapt it for local needs
(2) The architecture should be hierarchically extendable, so that one MCI Node collects MCI data from different sensors but acts itself as a sensor for further MCI Nodes in the higher hierarchy
(3) The architecture should be self-identifying in order to facilitate straightforward expansion of the station's MCI and promote IVS network uniformity
(4) Data logging should be cast into the form of a data management with safety, completeness, integrity and different logging and sampling rates

With these main concepts in mind throughout the development process, the committee has adopted a nodal structure which will incorporate a standardized setup and a MCI Interface, which each software for a sensor and each node must realize.

2. Suggested VLBI2010 MCI Node structure of the hardware

The MCI node follows a layered structure shown in figure 2-1. On layer 1 are all sensors of different types as temperature sensors, sensors for front-end monitoring, meteorological sensors, emergency switches etc., which offer different kind of data in different hardware forms. Layer 2 is an optional data collimation and safety layer. It helps to combine several sensors with hardware, to sample data with an own hardware or to realize a safety system which must re-act on critical situation in real-time using hardware to protect human beings and the monitored system itself with interlock connections. Layer 3 is a computer (in best case a fanless, low-energy PC) which realizes the MCI Node for the data acquisition, preparation and presentation. All software parts for the sensors, recorded by the node, reside there. Finally layer 4 realizes the usage of the data for presentation, further processing, control, automation and distribution to the analyzing centers using HTML or a standardized MCI interface.

Layer 4 can also be another MCI Node which collects data from different MCI Nodes over Ethernet and from other direct connected hardware sensors. This allows to build-up hierarchical systems which propagate their data throughout a monitored system to controlling and interpretation instances. High level system monitoring tools as Nagios, Pandora FMS or Zabbix can be adapted, acting as MCI node.

The sensors itself are connected using analog or digital connections in combination with PC-cards, serial connection of the different types as RS232 or RS485, Ethernet connections over a Local Area Network or any other connection which can be read with software in the MCI Node (e.g. USB).

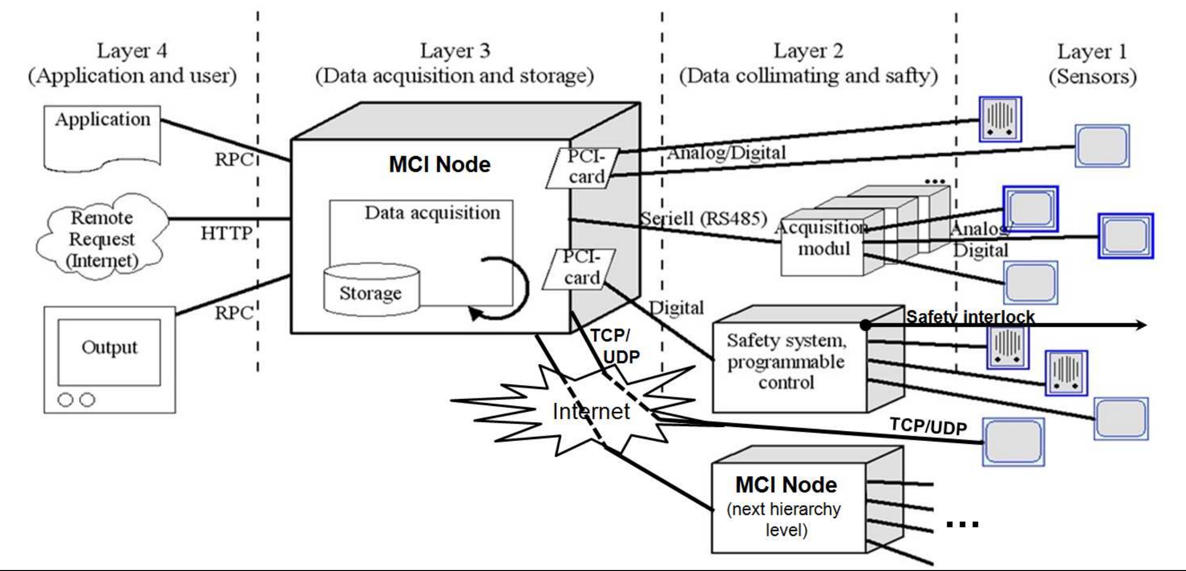


Fig. 2-1: Hardware structure of a MCI Node

3. Suggested VLBI2010 MCI Node structure of the software

For the internal architecture of a node the MCI group suggests the following very open and integrative way. The main part of the node is a data storage system. As modern Data Base Management Systems (DBMS) allow many concurrent accesses and are in several cases scalable to different hardware platforms the data storage might be such a DBMS, e.g. PostgreSQL. It can be extended by a (proprietary) file system server, to build-up hybrid storage possibilities, e.g. to save configuration files to the files system while the data are kept in the database for a dedicated time period. Also for a service, which autonomously archives data from the data base to the file system, such a parallel file server can be used. The database is completely local and can only be accessed from there.

Another part is given by servers providing access of data stored in the database. They offer such information based on a standardized MCI interface. Two basic versions of interfaces are planned: one for the real-time data access (only for the current values), including a timestamp, the warning level and the monitoring value itself. The second version offers a possibility to get historic data in the same format as the real-time interface. As long as data are existent in the database they can be individually requested for different time intervals, e.g. to support plotting of graphs. These two interfaces offer very fast access to current as well as to historic data. Retrieving huge blocks of historic data from the database is probably more time consuming.

While all of the so far described parts are offered by the MCI group as a software package, the “data feeder”, which sends data into the system, must be written by each station individually. Such data injectors are MCI Sensor Control Points (SCP). An SCP is an individual program dealing with the hardware driver. It is responsible for the sampling, requesting, controlling, warning and time tagging. By using the standardized MCI interface they can inject data into the data storage system. In general they can operate and manage one single or several external hardware sensors as subsystems. This can be scaled to different requirements as one SCP can operate several hardware sensors. But also several SCPs, each with one different sensor, can run parallel on the node. Because of the standardized interface one node is again nothing more than a hardware SCP for another following node, requesting all sensor values at once, collected by the lower layered node.

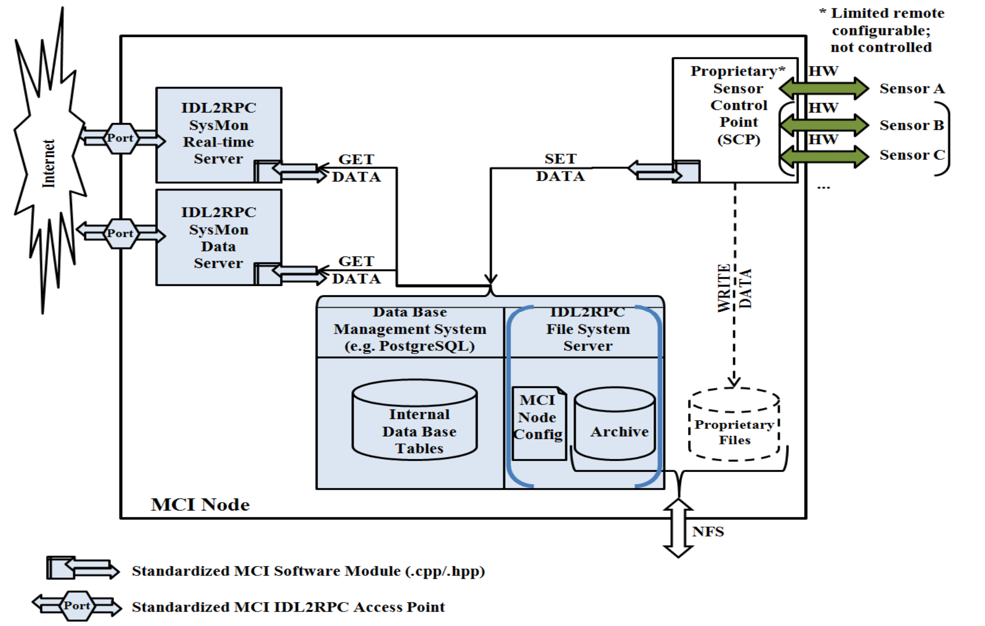


Fig. 3-1: The node internal software architecture for a simple proprietary sensor control point

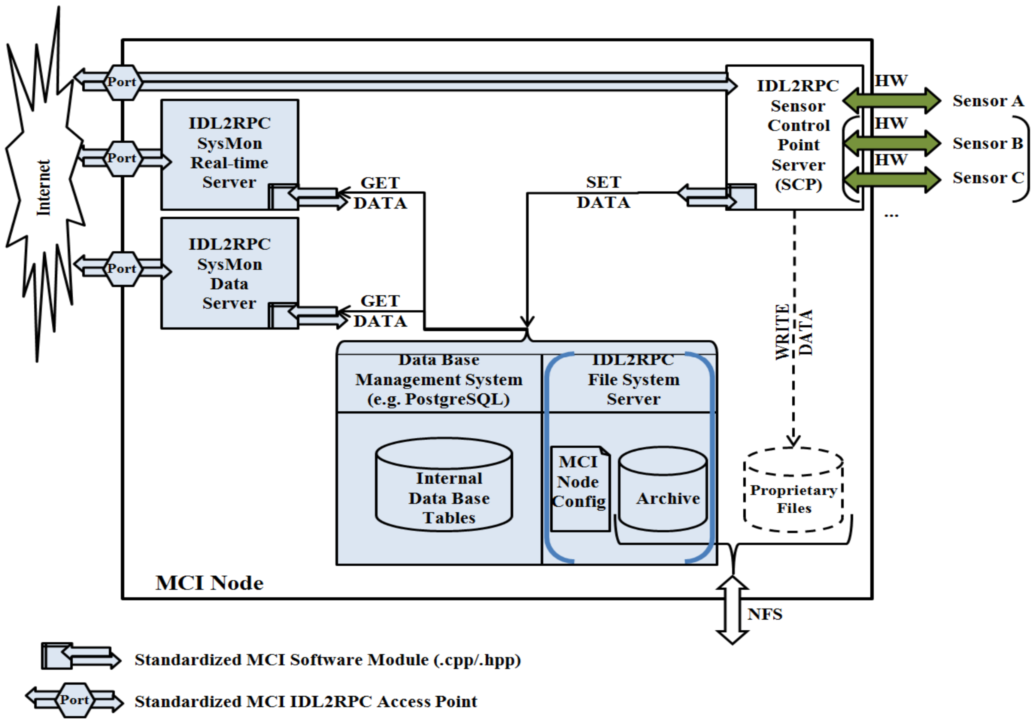


Fig. 3-2: The node internal software architecture for a MCI-suggested sensor control point

The easiest way to implement such a SCP is shown in figure 3-1. It is a simple program, which runs independently from the rest and sends in data. It is not controlled and only limited controllable from remote. In this case the complete quality management must be done individually, offered by the SCP developer. This attempt allows an easy startup and the connection to existing monitoring systems is possible, using simple scripts etc., as given with the Australian MONICA system.

The MCI suggested way to implement a MCP is illustrated in figure 3-2. It uses the IDL2RPC development. Remote Procedure Call communications are described with a simple Interface Definition Language (IDL), which are then converted automatically into code. It includes additional safety features, as e.g. a watchdog, on the server side. Therefore a MCI suggested SCP is an IDL2RPC server, which is controlled by the internal control structures of IDL2RPC offering its service on a dedicated port to the external environment. Therefore IDL2RPC requests can be sent directly to the SCP, which also offers the possibility for direct commanding. To process the requests parallel to the hardware control, each server can be configured to use threads (including all of the necessary semaphore methods to deal with critical sections, accessing memory or devices parallel).

Each SCP can also write its own data files to the file system, which allows higher data rates. All files are stored in a dedicated directory location, which is accessible either with NFS or Samba. The directory structure is individual for the proprietary files, written directly from the proprietary, direct writing SCPs. The parallel standardized file archiving system separates data into daily, monthly and yearly sections in another folder of this directory tree.

The SCP processes are started automatically on startup time of the node computer. During the very first startup they must register themselves at the data storage system. After then the servers get their startup configuration from the data storage system each startup, which they must request during the start. The registration is done by sending in the current configuration file content which must be of the style shown in code 3-1 to 3-3. The dedicated identifier tags must be defined. Beside these individual others are allowed. The configuration is then saved within the database, acting as a centralized data stock for the configuration. Local changes in the servers must be registered again. All other changes can be propagated from remote via the data storage system.

```
<MCISensorControlPoint>
  ControlPointID = Wz_Invar_1 # Unique identifier (name or number) for a sensor
                              # in a system => this will be converted to the MCI
                              # identifier of <SCPID>(<IPADR>,<PORT>)
  ControlPointType = IDL2RPC  # Type of this control point, e.g. IDL2RPC or PROPRIETARY
  ControlPointPort = 50500    # Access port of the SCP for direct requests
  # Individual control point configuration values e.g. device settings for all sensors
  <MCISensor>
      # Connected hardware sensor
      SensorID = Wz_TempCenter_1   # Unique identifier (name or number) for a sensor
                                   # in a system
      SensorName = TempCenter      # Identifying name of a sensor controlled by SCP
      SensorType = Temperature sensor # Type of sensor e.g. temperature sensor
      SensorUnit = °C              # Unit of the measured value
      SensorManufacturer = Company # Manufacturer of the sensor
      SensorModel = AX510Temp      # Model number etc. of the sensor
      SensorPosition = Midway in azimuth axis # Descriptive position explanation or
                                              # geometric location
      SensorUpdateInterval = 180s  # Time steps between each value update (~ rate)
                                   # in seconds [s] or microseconds [us]
      SensorResolution = 0.05      # Resolution of sensor in the above unit
                                   # (also defines the valid decimal places)
      SensorDataAvailabilityTime = 1d # Direct availability of data from database
                                      # in days [d] or seconds [s]
      SensorMinLimit = -20         # Lower limit of representable values
      SensorMaxLimit = 50          # Upper limit of representable values
      SensorMinWarningLimit = -15  # Lower than this value throws warning state (or ‘off’)
      SensorMaxWarningLimit = 25   # Greater than this value throws warning state
                                   # (or ‘off’)
      SensorMinAlertLimit = -18    # Lower than this value throws alert state (or ‘off’)
      SensorMaxAlertLimit = 30     # Greater than this value throws alert state (or ‘off’)
      SensorFlagProvider = yes     # Flag that server collects HW data and offers them
      SensorFlagConsumer = no      # Flag that data can be sent to the server
      SensorFlagCommandable = no   # Flag that server offers a command line funct.
      SensorFlagManageable = no    # Flag that server offers additional RPC funct.
      SensorDataArchiveDirectory = /archive/MCI/ # Directory for standard archive service
                                                 # or empty
      SensorPropArchiveDirectory = # Directory for proprietary SCP data archiving
                                   # or empty
      # Individual sensor configuration values e.g. sensor specific device settings
  </MCISensor>
  <MCISensor>
      ...
  </MCISensor>
  <MCISensorSubnode>
      # Connected MCI subnode
  </MCISensorSubnode>
  ...
</MCISensorControlPoint>
```

Code 3-1: Configuration example for a MCI Sensor Control Point with connected hardware sensors

```
<MCISensorControlPoint>
```

```
  ControlPointID = Wz_RequSubnodeCabine_1 # Unique identifier (name or number)
                                          # for a sensor in a system
                                          # => this will be converted to the MCI
                                          # identifier of <SCPID>(<IPADR>,<PORT>)
  ControlPointType = IDL2RPC  # Type of this control point, e.g. IDL2RPC or PROPRIETARY
  ControlPointPort = 50520    # Access port of the SCP for direct requests
  # Individual control point configuration values e.g. device settings for all sensors
  <MCISensorSubnode>
      # Connected MCI subnode
      SubnodeIP = 192.168.178.200  # IP-address of subnode
      SubnodeRealtimePort = 50550  # Real-time port of subnode
      SubnodeUpdateInterval = 180s # Update interval for data from the subnode
                                   # (also propagated as concatenation to the higher
                                   # hierarchy levels e.g. 180s<<180s)
  </MCISensorSubnode>
</MCISensorControlPoint>
```

Code 3-2: Configuration example for a MCI Sensor Control Point with a connected MCI subnode

```
<MCINode>
  NodeID = Wz_SubnodeCabine_1 # Unique identifier (name or number) for a sensor
                              # in a system => this will be converted to the MCI
                              # identifier of <SCPID>(<IPADR>,<PORT>)
  NodeType = IDL2RPC  # Type of this control point, e.g. IDL2RPC or PROPRIETARY
  NodeRealtimePort = 50600    # Access port of the SCP for direct requests
  NodeSelectPort = 50601      # Access port of the SCP for history requests
  # Individual control point configuration values e.g. device settings for all sensors
</MCINode>
```

Code 3-3: Configuration example for a MCI Node Data Server (used by real-time and selectable data server)

The net sensor identification, used in the system to identify the sensor over a complete hierarchical architecture, is generated automatically using a concatenation of the different identifiers in combination with the IP-addresses and the ports (see figure 3-3). If a proprietary sensor only propagates data to the data storage system it will be registered with the combined ControlPointID and SensorID (e.g. ControlPointID<<SensorID as net sensor identification) on the local data system. If it uses the IDL2RPC style and offers the data also on a separate port, it uses the combination of the ControlPointID with IP-address and Port in brackets and the SensorID (e.g. with the example above "Wz_Invar_1(193.174.168.100,50500)<< Wz_TempCenter_1" as net sensor identification). The concatenation of these names are done automatically within the data storage system interface. As the data are also offered over a real-time and selectable data server the concatenation is extended with the information about the access server (e.g. "Wz_SubnodeCabine_1(193.174.168.100,50600)<<Wz_Invar_1(193.174.168.100,50500)<<Wz_TempCenter_1" as net sensor identification). The next hierarchy level follows the same style and so on. A more illustrating example is shown in figure 3-3.

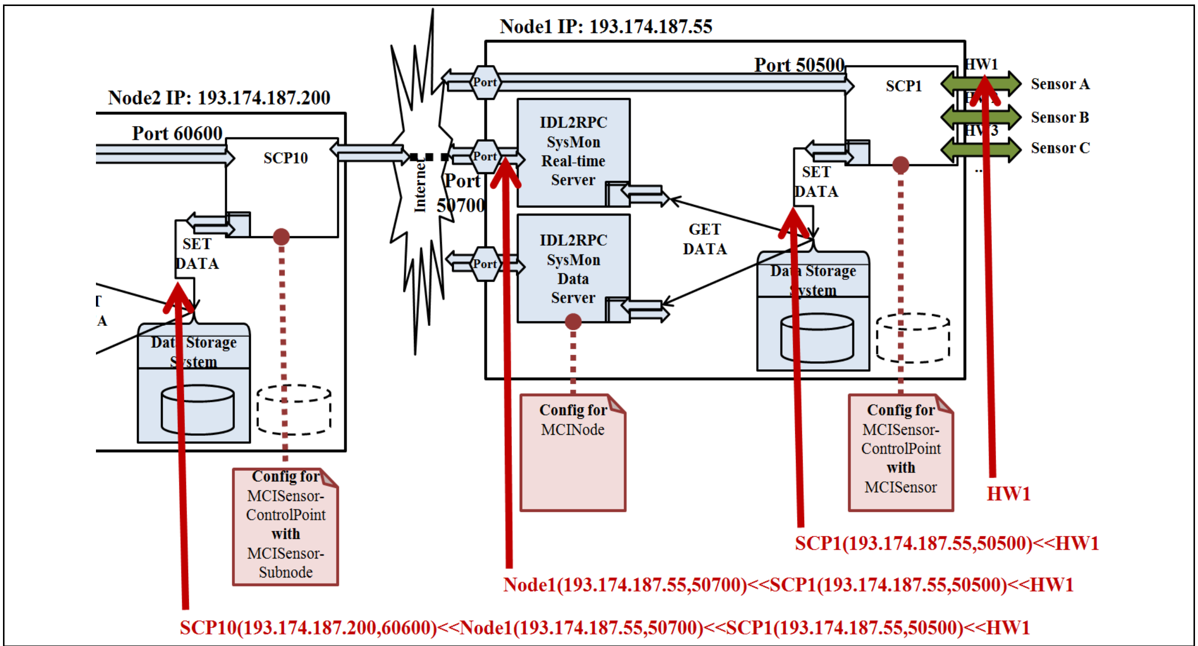


Fig. 3-3: *Illustration of the compilation of net sensor identifiers in hierarchical systems*

All of these structures are organized using a standardized interface in a library to the data storage system (shared object .so in combination with header files). All functionalities are hidden behind this interface module which is similar to the IDL2RPC remote procedure call interface. Main part of the data storage system is the architecture of the node-internal database, which is kept very simple. All administrative activities are operated in the interface library, so that the user does not have to know anything about databases at all. A possible database structure is shown in figure 3-4 and 3-5.

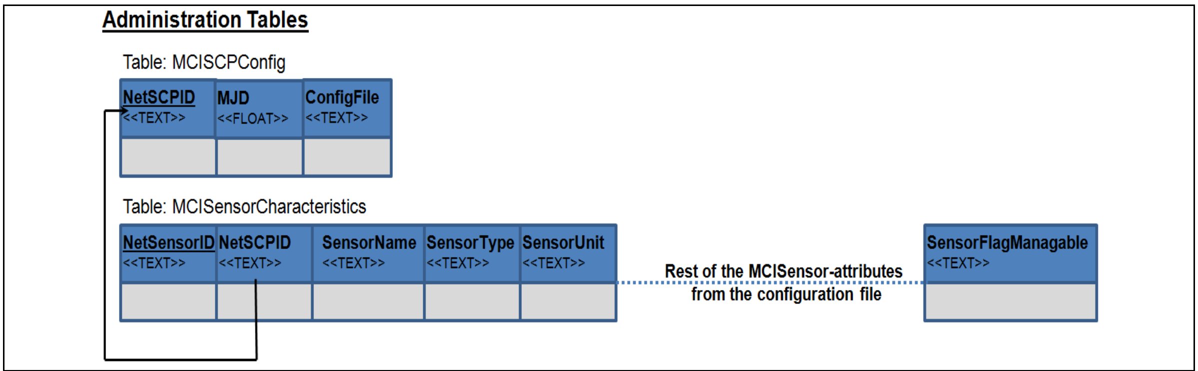


Fig. 3-4: Database tables to administrate sensors and sensor control points

The administrative tables contain redundant information, ~~once~~ in configuration file text form and ~~once~~ in separated simply accessible information. Changes can only be propagated in form of configuration files. The information is then split up into the second table which might also contain local adaption e.g. as the locally used sensor update interval in hierarchical systems.

The data tables contain the time-tagged current values, the time-tagged historic values in sensor specific separate tables per sensor and the time-tagged log information in sensor specific separate tables. Each table contains the time in Modified Julian Date (MJD) including time to a precision of msec. This offers a possibility for simple ordering of values. The values themselves are saved in IEEE double-precision format. If ASCII conversions are processed, the complete double-precision number (decimal places) are used. Each value record contains a flag for alarm levels, where 0 = no alarm, 1 = sensor control point alarm, 2 = warning, 3 = alert. If additional specifications are necessary, they can be coded with alarm values greater than 3.

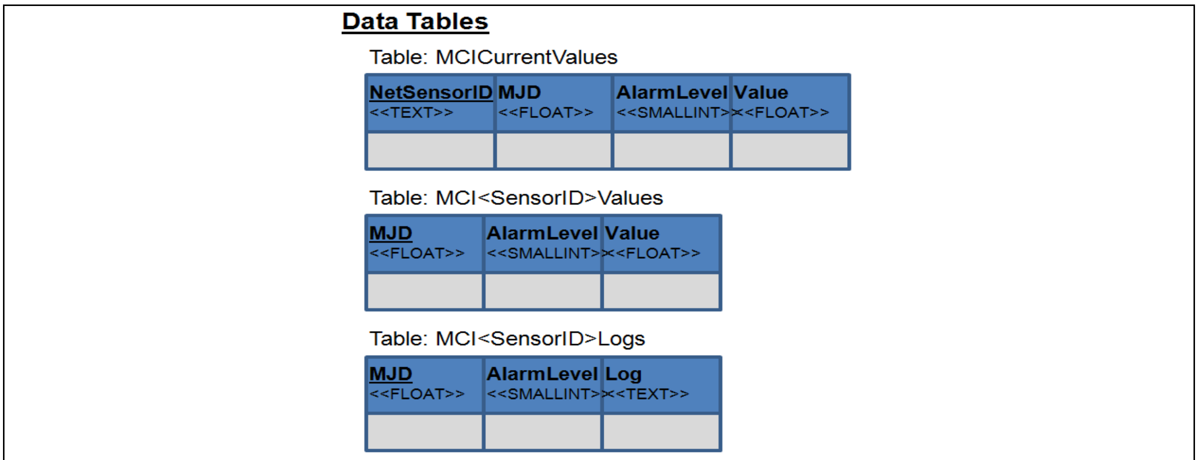


Fig. 3-4: Database tables to save data of sensors

The archiving system of the MCI node uses tables and configuration information to create separated daily files located in day and year folders under the directory /archive/MCI/<NetSensorID>. Proprietary file writer in the sensor control points itself can write to anywhere.

Another possibility is the installation of a local web server (as e.g. Apache). It can be used to offer configuration and data access via Web pages maybe in combination with some graphical or higher level monitoring tools (as Zabbix)

4. VLBI2010 MCI Standardized Interface functionality

Main goal of this document is to define standardized interface functionality for the above described hardware and software structure. The C-function collection in code 4-1 is a first realization of such an interface. Explanations are included to the function heads.

```
// Function return values
const MCISCP_RET_OK  = 0;  // Function returns ok
const MCISCP_RET_NOK = 1;  // Function returns not ok

// Warning levels
const const MCISENSOR_WARNINGLEVEL_OK = 0;  // No warning
const const MCISENSOR_WARNINGLEVEL_SCPALARM = 1;  // Sensor control point server alarm
const const MCISENSOR_WARNINGLEVEL_WARNING = 2;   // Value is in warning interval
const const MCISENSOR_WARNINGLEVEL_ALERT = 3;     // Value reached alert interval

// Interface function definition
interface <SensorControlPointName> #MCIVer1.0_20120417001
{
    // ########################################################################################
    // GET MONITORING DATA
    // ########################################################################################
    /**********************************************************************
        function  usGetCurrentDataText
     **********************************************************************/
    /*!           Return the content of the database table for current values
                  in form of the original database table
                  with the possibility to select dedicated net sensors by ID
                  Returns: NetSensorID | MJD | AlarmLevel | Value
        \param    in pstrNetSensorIDSelect<> -> Array to select net sensors or empty
        \param    out string pstrSensorValueColumnNames<> <- The table column head lines as array
        \param    out string ppstrSensorValueTable<><> <- 2-dim. Array with the table content of the
                                                         request
        \return   unsigned short <- Error code (0 = ok, >0 = error)
     **********************************************************************/
    /*  author    MCI colaboration group
        date      17.04.2012
        revision  -
        info      -
     **********************************************************************/
    unsigned short usGetCurrentDataText (in pstrNetSensorIDSelect<>,
                                         out string pstrSensorValueColumnNames<>,
                                         out string ppstrSensorValueTable<><>);
    /**********************************************************************
        function  usGetCurrentDataMJD
     **********************************************************************/
    /*!           Return the content of the database table for current values
                  in form of separated column vectors and in Modified Julian Date (MJD)
                  with the possibility to select dedicated net sensors by ID
                  (each row over the column vectors is one sensor entry)
        \param    in pstrNetSensorIDSelect<> <-> Array to select net sensors or empty,
                                              returns the resulting, found net sensors
        \param    out double pdTimeMJD<> <- Resulting array with the record times in MJD
        \param    out unsigned int puiAlarmLevel<> <- Resulting array with alarm levels
        \param    out double pdValue<> <- Resulting array with values
        \return   unsigned short <- Error code (0 = ok, >0 = error)
     **********************************************************************/
    /*  author    MCI colaboration group
        date      17.04.2012
        revision  -
        info      -
     **********************************************************************/
    unsigned short usGetCurrentDataMJD (inout pstrNetSensorID<>,
                                        out double pdTimeMJD<>,
                                        out unsigned int puiAlarmLevel<>,
                                        out double pdValue<>);

    /**********************************************************************
        function  usGetCurrentDataTextSinceMJD
     **********************************************************************/
    /*!           Same as usGetCurrentDataText including a selection for dates since a
                  dedicated time, so that only values are transferred which have been updated
                  during this time period
        \param    in pstrNetSensorIDSelect<> -> Array to select net sensors or empty
        \param    in double dSinceTimeMJD -> Time since when the data should be returned
        \param    out string pstrSensorValueColumnNames<> <- The table column head lines as array
        \param    out string ppstrSensorValueTable<><> <- 2-dim. Array with the table content of the
                                                         request
        \return   unsigned short <- Error code (0 = ok, >0 = error)
```

```
 **********************************************************************/
/*  author    MCI colaboration group
    date      17.04.2012
    revision  -
    info      -
 **********************************************************************/
unsigned short usGetCurrentDataTextSinceMJD (in pstrNetSensorIDSelect<>,
                                             in double dSinceTimeMJD,
                                             out string pstrSensorValueColumnNames<>,
                                             out string ppstrSensorValueTable<><>);
/**********************************************************************
    function  usGetDataFromToMJDText
 **********************************************************************/
/*!          Transfer complete tables of data of one sensor with selection of start
             time and an end time
    \param    in strNetSensorIDSelect -> Select net sensors
    \param    in double dStarttimeMJD -> Time since when the data should be returned
    \param    in double dEndtimeMJD -> Time to which data should be returned
    \param    in unsigned short usTableSelector -> Select table (0=Value&Log, 1=Value, 2=Log)
    \param    out string pstrSensorValueColumnNames<> <- The table column head lines as array
    \param    out string ppstrSensorValueTable<><> <- 2-dim. Array with the table content of the
                                                    request for data with the structure
                                                    NetSensorID | MJD | AlarmLevel | Value
    \param    out string pstrSensorLogColumnNames <> <- The table column head lines as array
    \param    out string ppstrSensorLogTable <><> <- 2-dim. Array with the table content of the
                                                    request for logs with the structure
                                                    NetSensorID | MJD | AlarmLevel | LogText
    \return   unsigned short <- Error code (0 = ok, >0 = error)
 **********************************************************************/
/*  author    MCI colaboration group
    date      17.04.2012
    revision  -
    info      -
 **********************************************************************/
unsigned short usGetDataFromToMJDText (in strSelectNetSensorID,
                                       in double dStarttimeMJD,
                                       in double dEndtimeMJD,
                                       in unsigned short usTableSelector,
                                       out string pstrSensorValueColumnNames<>,
                                       out string ppstrSensorValueTable<><>,
                                       out string pstrSensorLogColumnNames<>,
                                       out string ppstrSensorLogTable<><>);

// ##########################################################################################
// SET MONITORING DATA
// ##########################################################################################
/**********************************************************************
    function  usSetDataText
 **********************************************************************/
/*!          Insert a data table into system with the structure
             NetSensorID | MJD | AlarmLevel | Value
    \param    in string ppstrSensorValueTable<><> -> 2-dim. Array with the table content of the
                                                    insert
    \return   unsigned short <- Error code (0 = ok, >0 = error)
 **********************************************************************/
/*  author    MCI colaboration group
    date      17.04.2012
    revision  -
    info      -
 **********************************************************************/
unsigned short usSetDataText (in string ppstrSensorValueTable <><>);
/**********************************************************************
    function  usSetSingleDataMJD
 **********************************************************************/
/*!          Insert a single data set into system
    \param    in string strNetSensorID -> Net sensor identifier
    \param    in double dTimeMJD -> Time in MJD when the value was recorded
    \param    in unsigned int uiAlarmLevel -> Alarm level of value
    \param    in double dValue -> The value itself
    \return   unsigned short <- Error code (0 = ok, >0 = error)
 **********************************************************************/
/*  author    MCI colaboration group
    date      17.04.2012
    revision  -
    info      -
 **********************************************************************/
unsigned short usSetSingleDataMJD (in string strNetSensorID,
                                   in double dTimeMJD,
                                   in unsigned int uiAlarmLevel,
                                   in double dValue);
/**********************************************************************
    function  usSetDataMJD
 **********************************************************************/
/*!          Insert a data sets into system
    \param    in string pstrNetSensorID <> -> Net sensor identifiers
    \param    in double pdTimeMJD <> -> Times in MJD when the values were recorded
    \param    in unsigned int puiAlarmLevel <> -> Alarm levels of value
    \param    in double pdValue <> -> The values itself
```

```
    \return   unsigned short <- Error code (0 = ok, >0 = error)
 ***********************************************************************/
/*  author    MCI colaboration group
    date      17.04.2012
    revision  -
    info      -
 ***********************************************************************/
unsigned short usSetDataMJD (in string pstrNetSensorID <>,
                             in double pdTimeMJD <>,
                             in unsigned int puiAlarmLevel <>,
                             in double pdValue <>);

// ##########################################################################################
// SET LOG DATA
// ##########################################################################################
/***********************************************************************
    function  usSetLog
 ***********************************************************************/
/*!           Insert a log table into system with the structure
              NetSensorID | MJD | AlarmLevel | LogText
    \param    in string ppstrSensorLogTable <><> -> 2-dim. Array with the table content of the
                                                    insert
    \return   unsigned short <- Error code (0 = ok, >0 = error)
 ***********************************************************************/
/*  author    MCI colaboration group
    date      17.04.2012
    revision  -
    info      -
 ***********************************************************************/
unsigned short usSetLog (in string ppstrSensorLogTable <><>);
/***********************************************************************
    function  usSetSingleLogMJD
 ***********************************************************************/
/*!           Insert a single log set into system
    \param    in string strNetSensorID -> Net sensor identifier
    \param    in double dTimeMJD -> Time in MJD when the value was recorded
    \param    in unsigned int uiAlarmLevel -> Alarm level of value
    \param    in string strLogText -> The log text itself
    \return   unsigned short <- Error code (0 = ok, >0 = error)
 ***********************************************************************/
/*  author    MCI colaboration group
    date      17.04.2012
    revision  -
    info      -
 ***********************************************************************/
unsigned short usSetSingleLogMJD (in string strNetSensorID,
                                  in double dTimeMJD,
                                  in unsigned int uiAlarmLevel,
                                  in string strLogText);
/***********************************************************************
    function  usSetLogMJD
 ***********************************************************************/
/*!           Insert a log sets into system
    \param    in string pstrNetSensorID <> -> Net sensor identifiers
    \param    in double pdTimeMJD <> -> Times in MJD when the values were recorded
    \param    in unsigned int puiAlarmLevel <> -> Alarm levels of value
    \param    in string pstrLogText <> -> The log texts itself
    \return   unsigned short <- Error code (0 = ok, >0 = error)
 ***********************************************************************/
/*  author    MCI colaboration group
    date      17.04.2012
    revision  -
    info      -
 ***********************************************************************/
unsigned short usSetLogMJD (in string pstrNetSensorID <>,
                            in double pdTimeMJD <>,
                            in unsigned int puiAlarmLevel <>,
                            in string pstrLogText <>);

// ##########################################################################################
// SET CONFIGURATION TO ADMINISTRATE SENSOR CONTROL POINT
// ##########################################################################################
/***********************************************************************
    function  usSetSingleConfigurationMJD
 ***********************************************************************/
/*!           Insert a single configuration into system
    \param    in string strNetSensorControlPointID -> Net sensor control point identifier
    \param    in double dTimeMJD -> Valid-from time of configuration
    \param    in string strConfigFileContent -> Configuration content
    \return   unsigned short <- Error code (0 = ok, >0 = error)
 ***********************************************************************/
/*  author    MCI colaboration group
    date      17.04.2012
    revision  -
    info      -
 ***********************************************************************/
unsigned short usSetSingleConfigurationMJD (in string strNetSensorControlPointID,
                                            in double dTimeMJD,
```

```
                                                 in string strConfigFileContent);
    /***********************************************************************
       function  usSetConfigurationMJD
     ***********************************************************************/
    /*!          Insert configurations into system
        \param    in string pstrNetSensorControlPointID <> -> Net sensor control point identifiers
        \param    in double pdTimeMJD <> -> Valid-from times of configuration
        \param    in string pstrConfigFileContent <> -> Configuration contents
        \return   unsigned short <- Error code (0 = ok, >0 = error)
     ***********************************************************************/
    /*  author    MCI colaboration group
        date      17.04.2012
        revision  -
        info      -
     ***********************************************************************/
    unsigned short usSetConfigurationMJD (in string pstrNetSensorControlPointID <>,
                                          in double pdTimeMJD <>
                                          in string pstrConfigFileContent <>);

    // ##########################################################################################
    // GET CONFIGURATION TO ADMINISTRATE SENSOR CONTROL POINT
    // ##########################################################################################
    /***********************************************************************
       function  usGetConfigurationMJD
     ***********************************************************************/
    /*!          Return configurations from system with selection of sensors
        \param    inout string pstrNetSensorControlPointID <> <-> Net sensor control point identifiers
                                                               and selector
        \param    out double pdTimeMJD <> <- Valid-from times of configuration
        \param    out string pstrConfigFileContent <> <- Configuration contents
        \return   unsigned short <- Error code (0 = ok, >0 = error)
     ***********************************************************************/
    /*  author    MCI colaboration group
        date      17.04.2012
        revision  -
        info      -
     ***********************************************************************/
    unsigned short usGetConfigurationMJD (inout string pstrNetSensorControlPointID <>,
                                          out double pdTimeMJD <>,
                                          out string pstrConfigFileContent <>);
    /***********************************************************************
       function  usGetConfigurationSinceMJD
     ***********************************************************************/
    /*!          Return configurations from system with selection of sensors (similar to
                 usGetConfigurationMJD) with an additional selector, to just get recently changed
                 configurations
        \param    inout string pstrNetSensorControlPointID <> <-> Net sensor control point identifiers
                                                               and selector
        \param    in double dSinceTimeMJD -> Select from this time on
        \param    out double pdTimeMJD <> <- Valid-from times of configuration
        \param    out string pstrConfigFileContent <> <- Configuration contents
        \return   unsigned short <- Error code (0 = ok, >0 = error)
     ***********************************************************************/
    /*  author    MCI colaboration group
        date      17.04.2012
        revision  -
        info      -
     ***********************************************************************/
    unsigned short usGetConfigurationSinceMJD (inout string pstrNetSensorControlPointID <>,
                                               in double dSinceTimeMJD,
                                               out double pdTimeMJD <>,
                                               out string pstrConfigFileContent <>);

    // ##########################################################################################
    // OTHER ADMINISTRATION FUNCTIONS
    // ##########################################################################################
    /***********************************************************************
       function  usGetRegisteredNetSensorIDs
     ***********************************************************************/
    /*!          Return a list of registered net sensors
        \param    out string pstrNetSensorID <> <- List of sensor identifiers
        \return   unsigned short <- Error code (0 = ok, >0 = error)
     ***********************************************************************/
    /*  author    MCI colaboration group
        date      17.04.2012
        revision  -
        info      -
     ***********************************************************************/
    unsigned short usGetRegisteredNetSensorIDs (out string pstrNetSensorID <>);

    // ##########################################################################################
    // Individual, proprietary functions
    // ##########################################################################################
    // <your personal functions>
};
```

Code 4-1: The suggested interface functionality to communicate with a MCI system monitoring node in IDL2RPC notation

5. Monitor control points in radio telescopes

A suggestion for monitoring of IVS telescopes separates the different types of monitoring data into three different types:

- Data for science and analysis (mostly lower sampling rates of several seconds; scheduled or predefined; e.g. meteo, WVR, clock offsets, ...)
- Data for system operations (medium sampling rates of seconds and sub-seconds similar to the human re-action times; permanently available for system control; e.g. power supply, wind, emergency stops, rack temperatures, ...)
- Data for diagnostics (higher sampling rates of milli- and micro-seconds; on demand according to sensor control point possibilities; e.g. servo currents, contouring errors, ...)

Recorded data can be used either for one single usage type or can be for different types.

SysMon

Hardware

and

Installation

by

Alexander Neidhardt

Hardware

• Standard equipment on standard, robust architectures

• Modular, multi-layer system

• Open for several devices and sensors

• Passive system for monitoring without actuators

• Linux-operating system (maybe minimal installation)

• Open Source

• C/C++

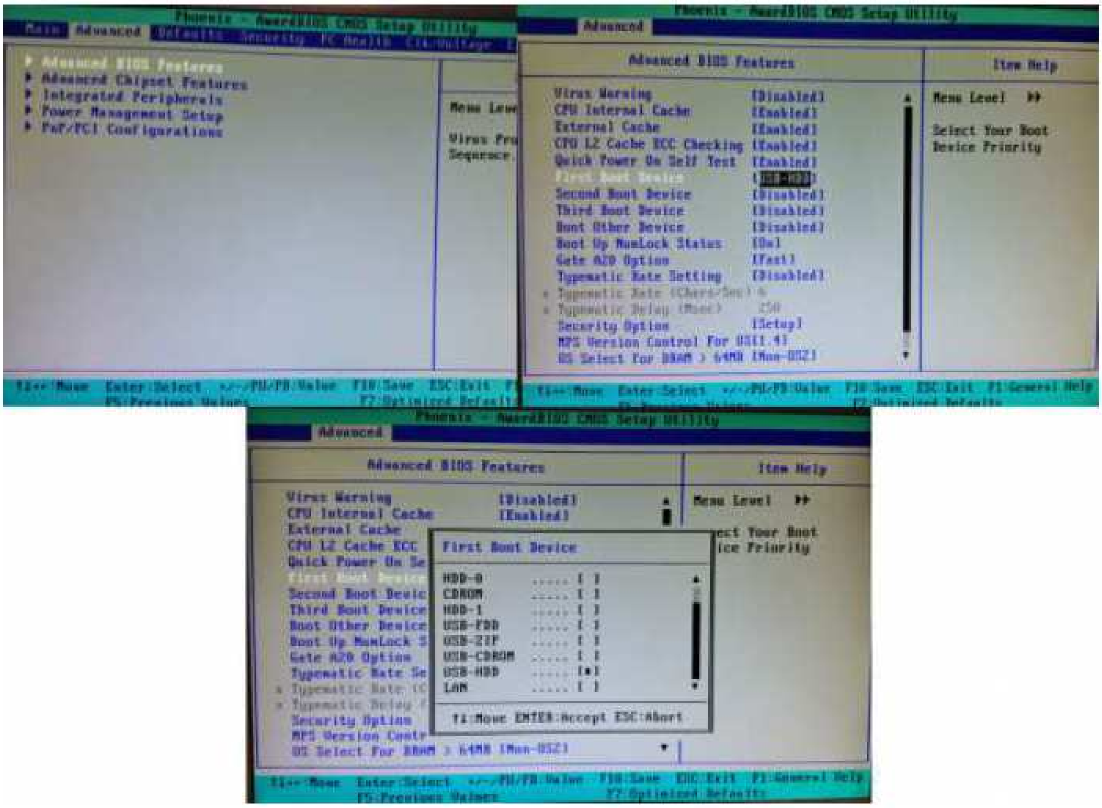
• Communication internal with idl2rpc-generator

• Vendor independence

VLBI System Monitoring (VLBI-SysMon)

Installation of Linux (e.g. Ubuntu)

- Create a bootable data stick (e.g. with the software "UNetbootin" under Windows") with the suitable Linux version (e.g. Ubuntu 14.04)
- Change BIOS setup (enter with DEL) to boot only from USB-ZIP (USB Zip-Drive) as first boot device and disable "Boot Other Device"
    - Older Phoenix AwardBIOS



- Plug-in data stick and boot
- Install Linux according to the description of the individual Linux distribution (try to run a standard installation). To simplify the installation, the complete system should be on one mount point, e.g. /dev/sda1 (check it with *"df -h"*). "oper" should be used as default user, which can also automatically login"
- After the installation, change the BIOS setup back to the first, installed harddisk (hard drive, HDD)
- On older AEC-6840 PCs ("blue boxes") with the VIA Technologies, Inc. VT8623 [Apollo CLE266] integrated CastleRock graphics [1106:3122] it is necessary to (the vendor of the graphic card can be found with *"lspci -vnn | grep VGA -A 12"*:
    - blacklist the VIA graphics card with *"echo "blacklist vt8623fb" > /etc/modprobe.d/blacklist-vt8623fb.conf"*
    - change the /etc/X11/xorg.conf to xorg.conf for openchrome
    - change the access rights *"chmod 644 /etc/X11/xorg.conf"*
- If necessary, set a APT-proxy with *"cat > /etc/apt/apt.conf"*, where the following must be entered (finish with "Ctrl+C")

```
Acquire::http::Proxy "http://gate-w.wettzell.ifag.de:8000";
```

- Downgrade the desktop environment from "Unity" to a lightweight one, e.g. "LXDE (Lightweight X11 Desktop Environment)", (it is still possible to change the environment after log-out and clicking onto the Ubuntu logo over the user login)
    - with " *sudo apt-get install lubuntu-desktop* "
    - and set it as default environment:
        - check which environments are available with " *ls /usr/share/xsessions/* " and if Lubuntu.desktop exists and
        - edit the default settings file with " *vi /usr/share/lightdm/lightdm.conf.d/50-ubuntu.conf* " as root and change it to

```
[SeatDefaults]
user-session=Lubuntu
```

- Reboot
- (Maybe it is necessary to change "update-apt-xapi"-settings, which updates the sofwtare database regularly and takes a lot of CPU time)

Incomplete Language Support

- If the Window "Information available - Incomplete Language Support" pops up, update the language support by clicking on "Run this action now" in the following window
- and follow the instructions

Disable screen locker

- Set all parameters in the screensaver in the menu **Start menu → Preferences → Light Locker Settings** to "Never" and switch "Enable light-locker" to "OFF"

Define network

- Set hostname: "*vi /etc/hostname*" and set it to "rtwsysmon"
- Set the network with the graphical user interface (see Ubuntu description),
  - under the LXDE (Lightweight X11 Desktop Environment) it is in the **Start menu → Preferences → Network Connections**
  - e.g. for Wettzell see the IP-addresses of the "vlbi" network
- Set the "Automatic proxy configuration URL" in the Internet browser to "http://gate-w.wettzell.ifag.de/gg.pac [http://gate-w.wettzell.ifag.de/gg.pac]"
- (**Hint:** Call "*sudo nm-connection-editor*" to start the connection editor in a remote SSH-terminal with X11-forwarded)

Enable serial devices

- Add user "oper" to the group "dialout" with "*usermod -a -G dialout oper*"
- Reboot the PC

- **If this does not work do the following:**
  - Create a file /etc/udev/rules.d/40-permissions.rules and
  - enter the following command into this file

```
KERNEL=="ttyS[0-9]",OWNER="root",GROUP="dialout",MODE="0777"
```

- Then enter the following command in a new xterm as "root"

```
/etc/init.d/udev stop
rm /dev/ttyS*
/etc/init.d/udev start
```

- Check the user rights of the ttyS* devices with "*ls -al /dev/ttyS**"
- Reboot the PC

Set clock to UTC (GMT+0)

- First set the time and timezone in the desktop with **Start menu → System tools → Time and Date**
- Set hardware clock to UTC: "*vi /etc/default/rcS*" and set line "*UTC=yes*"
- Set localtime to GMT+0: "*rm /etc/localtime*" and "ln -s /usr/share/zoneinfo/Etc/GMT+0 /etc/localtime"

Activate NTP

- "*apt-get install ntp*" (maybe also install ntpdate and ntp-server"
- Set local NTP servers for "ntpdate": "*vi /etc/default/ntpdate*"
  - Set line "*NTPSERVERS= "192.168.208.1 192.168.208.4 192.168.208.5"*" (delete the existing NTPSERVERS line)
- Set local NTP servers for "ntpd": "*vi /etc/ntp.conf*"
  - Set line "*server 192.168.208.1*"
  - Set line "*server 192.168.208.4*"
  - Set line "*server 192.168.208.5*"
  - Set all existing server lines as comments (starting '#')
- Set current time once
  - " */etc/init.d/ntp stop*"
  - "*ntpdate -s 192.168.208.1*"
  - " */etc/init.d/ntp start*"

Run a software update

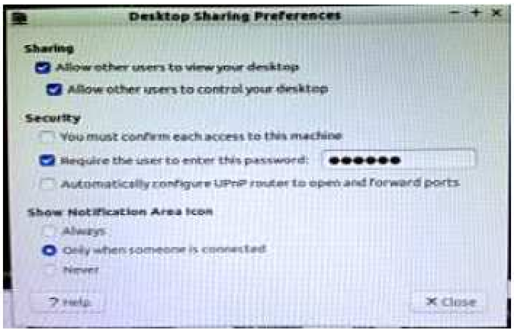
- Run a software update with **Start menu → System tools → Software Updater** (it will take a while)

Install SSH

- Install a SSH server with *"apt-get install openssh-server"*
- **Hint: Getting X11 forwarding through ssh working after running su**
    - Run *"xauth list $DISPLAY"* to get the cookie of the SSH connection, e.g. *"somehost.somedomain:10 mit-magic-cookie-1 4d22408a71a55b41ccd1657d377923ae"*
    - Change user with *"sudo su"*
    - Run *"xauth add «<cookie>»"*, e.g. *"xauth add somehost.somedomain:10 mit-magic-cookie-1 4d22408a71a55b41ccd1657d377923ae"* to add the forwarding cookie to the new user

(Install and) configure Vino VNC

- Configure the "Desktop Sharing Preferences" by calling *"vino-preferences"* as user "oper" (define a VNC password: here "+oper!")



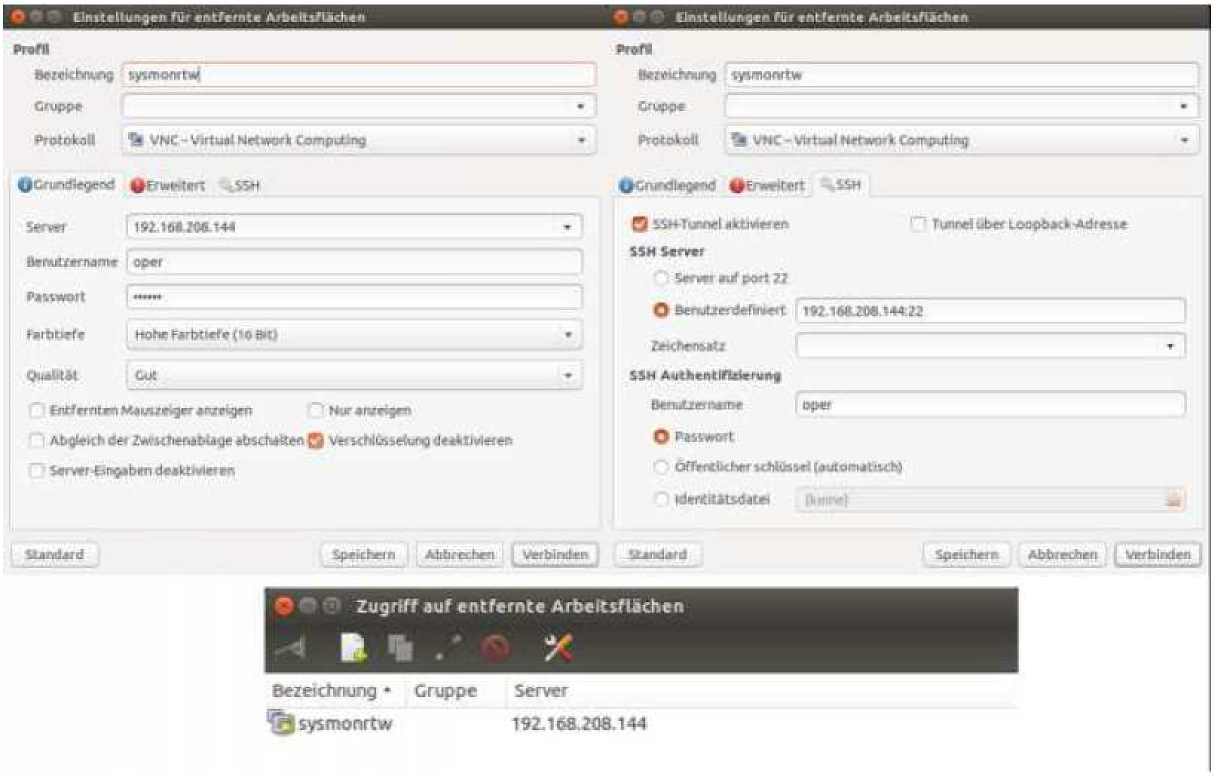
- Disable encryption, to easily allow the access with all VNC clients, with *"gsettings set org.gnome.Vino require-encryption false"*
- Create a new directory (if not yet available) as user "oper": *"mkdir /home/oper/Software"* and *"mkdir /home/oper/Software/scripts"*
- Change into the new directory with *"cd /home/oper/Software/scripts"*
- Create a start script "vinovnc.sh" with an editor in the new folder and add the following content:

```
#!/bin/bash
/usr/lib/vino/vino-server > /dev/null 2> /dev/null &
```

- Change the access rights of the new script with *"chmod 744 ./vinovnc.sh"*
- Create a desktop starter file with *"vi /home/oper/.config/autostart/vinovnc.desktop"* and add the following context (it can also be created with the program "lxshortcut"):

```
[Desktop Entry]
Type=Application
Name=Vino VNC server
Comment=Automatic start of the VINO VNC server
Exec=/home/oper/Software/scripts/vinovnc.sh
Terminal=false
```

- Test the automatic start: log-out and -on again, which should start the application
- The VNC Ports are: **5800** and **5900**
- An example configuration of a remote VNC client can look like the following setup for the Ubuntu "Remmina Remote Desktop Client" (similar settings can also be used for other VNC clients, like "Real VNC" under windows or xvnc4viewer under Linux; just if a tunneling is required, it must be set manually, using a separate SSH client)



Install Subversion and the local repository for VLBI SysMon

- Install Subvserion as root with "*apt-get install subversion*"
- Create a directory "Software" in the home directory of the user oper with "*mkdir /home/oper/Software* "
- Change into the new directory and fetch the SysMon source with the Subversion command "*svn co http://xsamba.wtz/svn/vlbi/trunk/code/vlbisysmon/ [http://xsamba.wtz/svn/vlbi/trunk/code/vlbisysmon/]* "
- Install the GNU C++ compiler with "*apt-get install g++* "
- Compile and build the necessary sources
- For the further editing install suitable editors, e.g.
  - "*apt-get install geany* "
  - "*apt-get install nedit* "
- For the further testing of the serial interfaces install a COM-software, e.g.
  - "*apt-get install cutecom*"

Install and activate PostgreSQL (e.g. PostgreSQL 9.3)

- "*apt-get install postgresql-9.3*"
  - The PostgreSQl database is then at " */var/lib/postgresql/9.3/main*"
  - The PostgreSQL configuration is then at " */etc/postgresql/9.3/main/postgresql.conf*" (to find the current location of the configuration file use "*ps ax | grep postgres*", which prints the complete calling arguments of the server including the "config_file" parameter)
  - Enable remote access
    - "*vi /etc/postgresql/9.3/main/postgresql.conf*" and enable "*listen_addresses = 'localhost'*" and "*port = 5432*"
    - "*vi /etc/postgresql/9.3/main/pg_hba.conf*" and enable "host all all 127.0.0.1/32 trust"

```
# Database administrative login by UNIX sockets
local   all         postgres                        trust
# TYPE  DATABASE    USER        CIDR-ADDRESS        METHOD
# "local" is for Unix domain socket connections only
local   all         all                             trust
# IPv4 local connections:
host    all         all         127.0.0.1/32        trust
```

```
# IPv6 local connections:
host    all         all         ::1/128             trust
# Zabbix database access
local   zabbix      zabbix                          md5
```

- Restart PostgreSQL with " */etc/init.d/postgresql stop*" and " */etc/init.d/postgresql start*" (" */etc/init.d/postgresql-8.4 restart*" my not work correctly
- Test the connectivity with " *psql -h 127.0.0.1 -p 5432 postgres postgres*" (quit with Ctrl-D)
- Create role and database:
  - "*CREATE ROLE sysmon ENCRYPTED PASSWORD '+sysmon!' SUPERUSER NOCREATEDB NOCREATEROLE NOINHERIT LOGIN CONNECTION LIMIT 100;*"
  - "*CREATE DATABASE sysmon WITH OWNER=sysmon;*"
- Test the connectivity to the new database with " *psql -h 127.0.0.1 -p 5432 sysmon sysmon*" (quit with Ctrl-D)
- For the programming simple_psqlquery [http://xsamba.wtz/svn/tools/trunk/modules/simple_psqlquery/] can be used
- Further documentation can be found on http://www.postgresql.org/docs/9.3/static/index.html [http://www.postgresql.org/docs/9.3/static/index.html]

- Install the PostgreSQL library for the compiler
- "*apt-get install libpq-dev*"

Wettzell System Monitoring Software (SysMon)

- The Wettzell System Monitoring Software (SysMon) should be fetched to the directory /home/oper/Software (see section Install Subversion and the local repository for VLBI SysMon)
- The software can be found on the Wettzell Subversion repository http://xsamba.wtz/svn/vlbi/trunk/code/vlbisysmon/ [http://xsamba.wtz/svn/vlbi/trunk/code/vlbisysmon/]

Graphical interface for the data in the database (Zabbix 2.2.2)

(a basic manual can be found here: https://www.zabbix.com/documentation/2.2/manual [https://www.zabbix.com/documentation/2.2/manual])

- Install the Zabbix software with
  - "*apt-get install zabbix-server-pgsql*"
  - "*apt-get install zabbix-agent*"
  - "*apt-get install zabbix-frontend-php*"
  - Configure the server with "*vi /etc/zabbix/zabbix_server.conf*"

```
DBHost=localhost
DBName=zabbix
DBUser=zabbix
DBPassword=zabbix
```

- Create the zabbix database after connecting with " psql -h 127.0.0.1 -p 5432 postgres postgres" (quit with Ctrl-D)

```
CREATE USER zabbix WITH PASSWORD 'zabbix';
CREATE DATABASE zabbix OWNER zabbix;
```

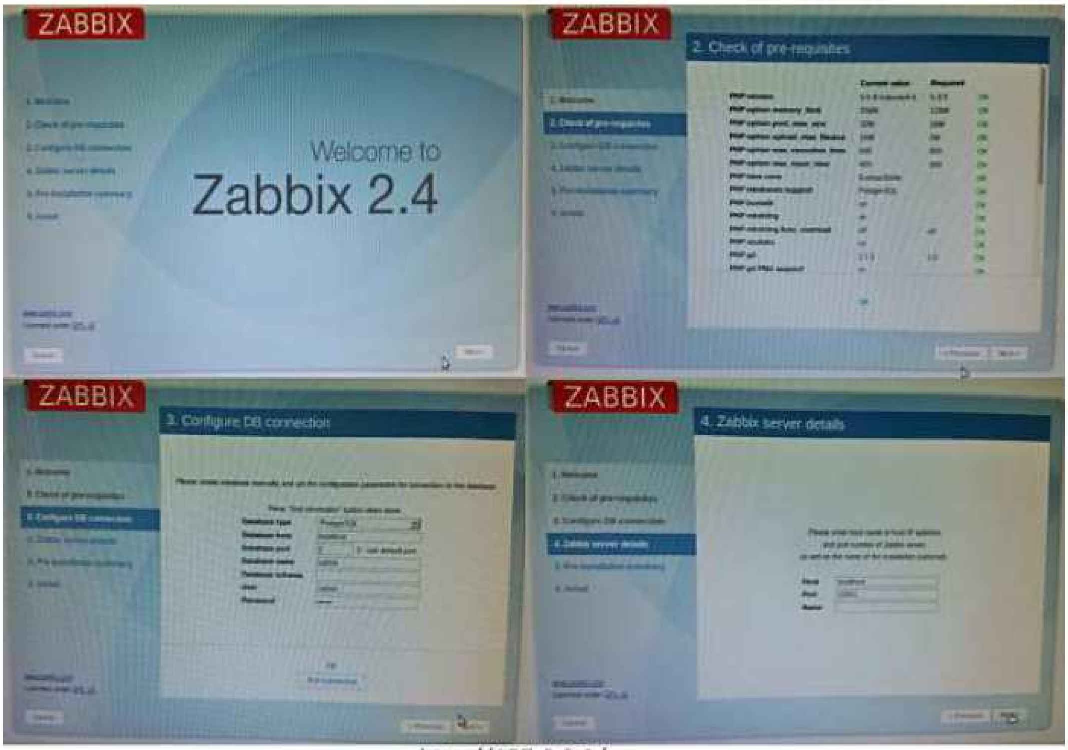
- Download the Zabbix sources which fit to the Zabbix installation of the operating system: e.g. for Ubuntu 14.04. LTS it is Zabbix 2.2.2 (to check, start "*zabbix_server*" with "*DebugLevel=3*" in the configuration file " */etc/zabbix/zabbix_server.conf*" and read the log file at " */var/log/zabbix-server/zabbix_server.log*", which is also defined in the configuration file of the server):zabbix-2.2.8.tar.gz (or zabbix-2.4.3.tar.gz) from http://www.zabbix.com/download.php [http://www.zabbix.com/download.php] to the directory /home/oper/Software/ and extract the package with "*tar -zxvf zabbix-2.2.8.tar.gz*"
- Change into folder /home/oper/Software/zabbix-2.2.8/database/postgresql and run (see https://www.zabbix.com/documentation/2.2/manual/appendix/install/db_scripts [https://www.zabbix.com/documentation/2.2/manual/appendix/install/db_scripts])

```
psql -U zabbix zabbix < schema.sql
# stop here if you are creating database for Zabbix proxy
psql -U zabbix zabbix < images.sql
psql -U zabbix zabbix < data.sql
```

- Start Zabbix server process
  - Enable startup with "*vi /etc/default/zabbix-server*" by setting "*START=yes*" and
  - start the server with " */etc/init.d/zabbix-server start* "
- Configure PHP with "*vi /etc/php5/apache2/php.ini* " and restart the Apache2 server with " */etc/init.d/apache2 stop* " and " */etc/init.d/apache2 start*"

```
[Date]
; Defines the default timezone used by the date functions
date.timezone = Europe/Berlin
max_execution_time = 600
post_max_size = 32M
memory_limit = 256M
mbstring.func_overload = 0
upload_max_filesize = 16M
max_input_time = 600
```

- Create Web front-end as root
  - " *cd /var/www* "
  - " *mv /var/www/html/ /var/www/html_original* "
  - " *chown -R www-data:www-data /var/www/html_original* "
  - " *mkdir html* "
  - " *cp -R /home/oper/Software/zabbix-2.2.8/frontends/php/* ./html/.* "
  - " *chown -R www-data:www-data /var/www/html* "
  - Restart the Apache2 server with " */etc/init.d/apache2 stop* " and " */etc/init.d/apache2 start*"
  - Open a browser and connect to "http://127.0.0.1 [http://127.0.0.1]" and follow the instructions (if the configuration file cannot be saved automatically, then download it and save it at /var/www/html/conf/.



http://127.0.0.1/

- Login as "Admin" with password "zabbix" at the Web front end mask

- Using Zabbix as user front-end for SysMon

Create an autostarter to automatically start firefox with the Zabbix Web page

- Create a new directory (if not yet available) as user "oper": *"mkdir /home/oper/Software/scripts"*
- Change into the new directory with *"cd /home/oper/Software/scripts"*
- Create a start script "firefox.sh" with an editor in the new folder and add the following content:

```
#!/bin/bash
/usr/bin/firefox -width 1024 -height 800 -new-tab http://127.0.0.1/ > /dev/null 2> /dev/null &
```

- Change the access rights of the new script with *"chmod 744 ./firefox.sh"*
- Create a desktop starter file with *"vi /home/oper/.config/autostart/firefox.desktop"* and add the following context (it can also be created with the program "lxshortcut"):

```
[Desktop Entry]
Type=Application
Name=Firefox Zabbix Autostart
Comment=Automatic start of the Firefox browser with the Zabbix interface
Exec=/home/oper/Software/scripts/firefox.sh
Terminal=false
```

- Test the automatic start: log-out and -on again, which should start the application

Installed sensors

- 20m RTW
    - Invar strain meter

vlbi/sysmon.txt · Last modified: 2015/02/03 15:59 by neidh

Zabbix 2.2.8

for

SysMon

by

Katharina Kirschbauer

Alexander Neidhardt

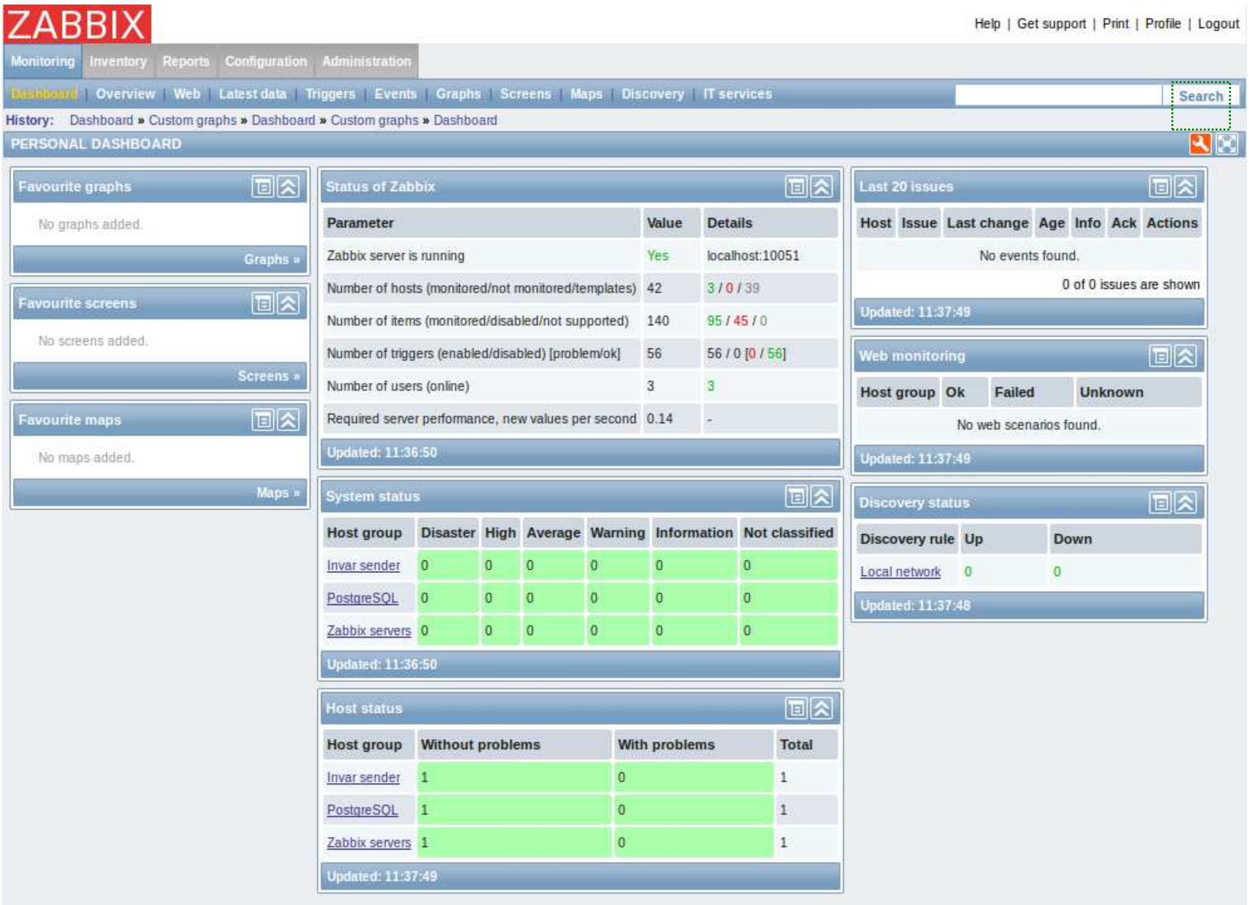
Zabbix 2.2.8 Workshop

(http://lab4.org/wiki/Zabbix_Schnellstart#Den_ersten_Host_anlegen)

I. LOGIN TO THE USER FRONT-END

II. DASHBOARD

For configurations: use the tool-button in the right upper corner to customize your personal dashboard.

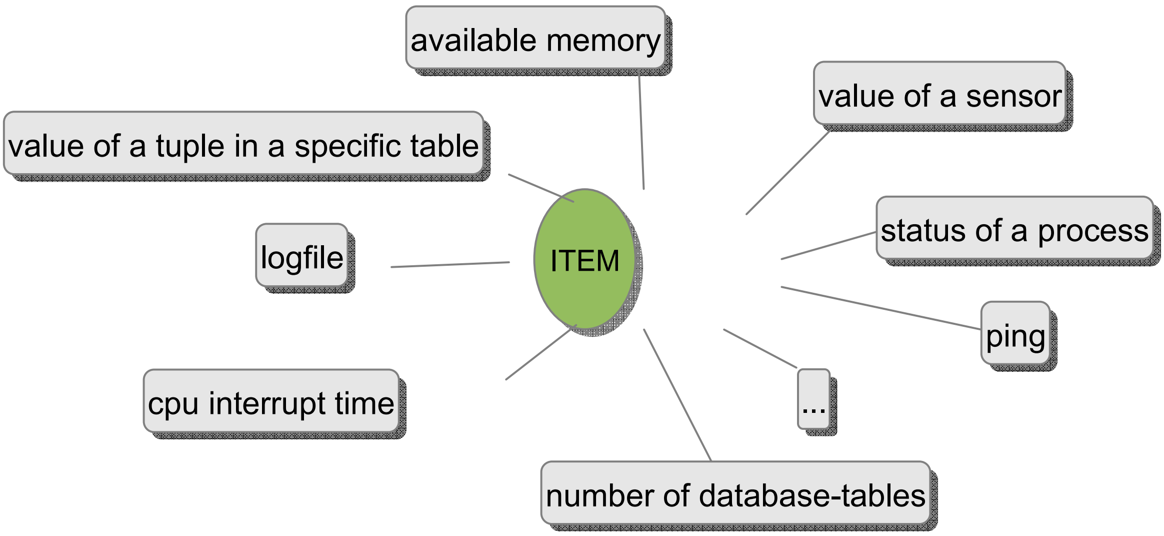


OVERVIEW

Monitoring	shows the configurations of the Zabbix-system	
	Dashboard	first page after login (customizable); most important configurations and statuses, including own graphs, screens and maps
	Overview	Type → Triggers: status of each trigger Type → Data: value of each item (like 'Latest Data'

		without details (e.g. Last check))
	Latest Data	value of each system- or customized-item with details (Interval, History, Trends, Type, Last check, Last value, Change, Graph ('spontaneous graphs' → system-generated graphs), Info)
Reports	reports about the status of zabbix, items and triggers	
Configuration	customizing of the zabbix-system	
	Hosts	configuration of hosts; with the linked items, triggers, graphs and templates; an overview about their status and availability
	Screens	configuration of screens
	Maps	configuration of maps
Administration	administration of the zabbix-system	
	Users	configuration of users and groups and specified permissions
	Queue	items that are waiting to be updated are displayed; ideally, it should all be green (no items in the queue); red = lacking server performance, connection problems or problems with agents

a) Item



b) Trigger
   - check the items' value (e.g. throw a warning, if the value is too low)

c) Action
   - 'error handling'
   - which action should be executed when a trigger appears (e.g. send and e-mail to the systems' administrator)

d) Graph
   - simple graph: system-made graph for each item of a numeric type
   - graph: ability to bring all the interesting items for a customer into one graph

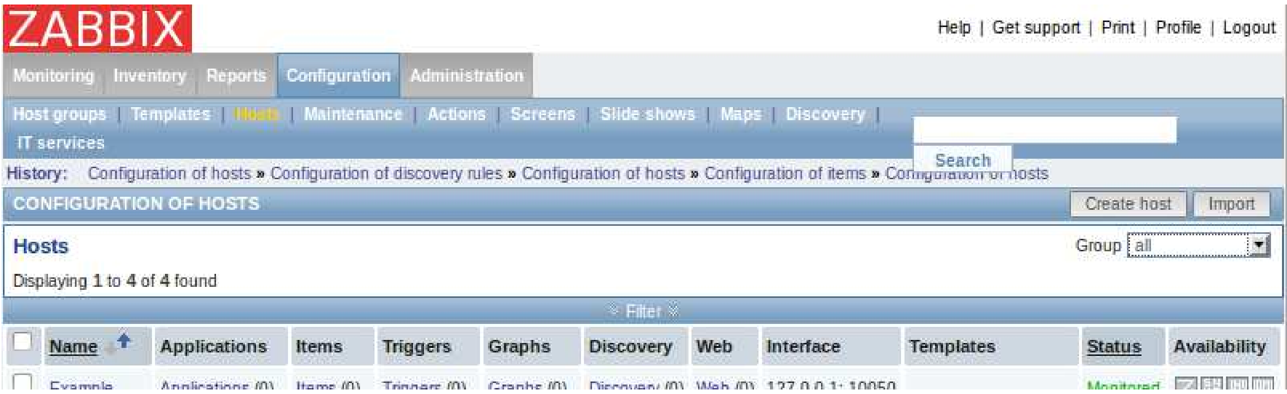
e) Map
   - to create a view of the network/interfaces you use

f) Screen
   - to bring all the important graphs/maps/information into one screen

g) Template
   - the zabbix-system contents pre-installed templates
   - a template inherits pre-defined items, triggers, graphs, maps, screens, …
   - you can export them or import a self-customized template (xml-file)

a) Create a host (logged in as 'Admin')

- *Configuration → Hosts*



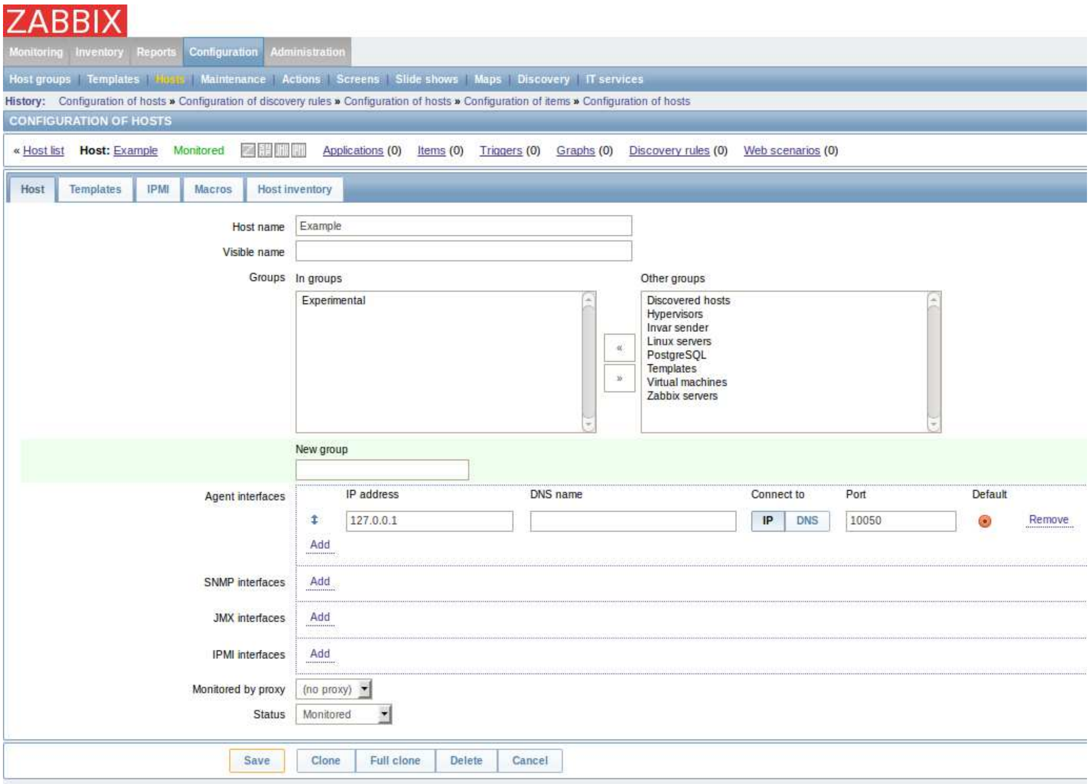
- Select *Group: all* (dropdown-menu, right upper corner)
- *Create host*

Name = Example

New Group = Experimental

IP = 127.0.0.1

Port = 10050



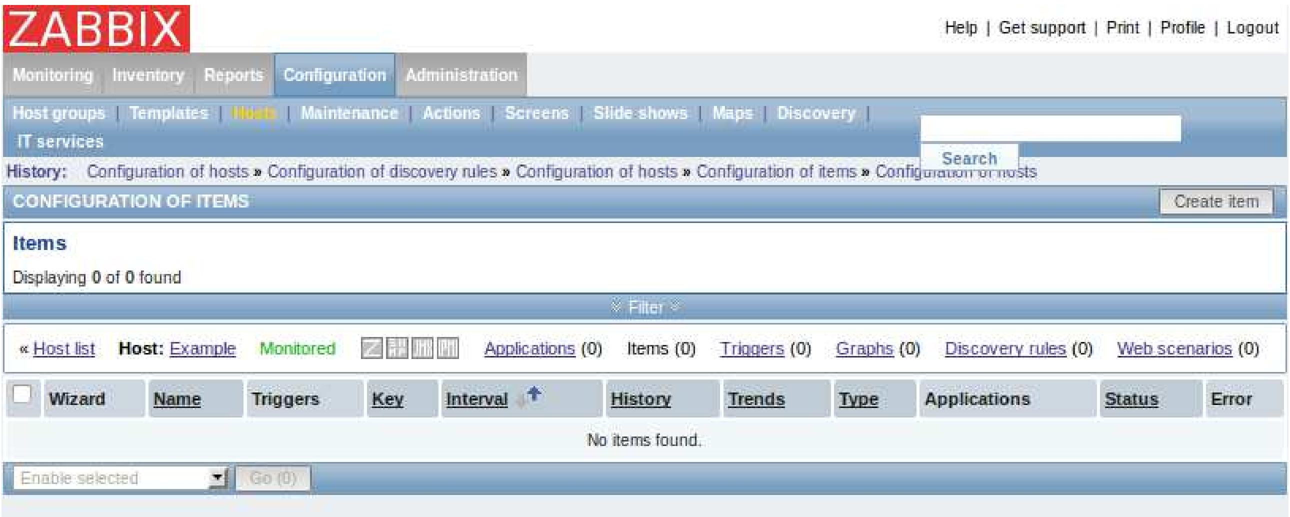
- Save

b) Create the first item

- Configuration → Hosts

- The new Host 'Example' is now available

- Select *Items(0)*



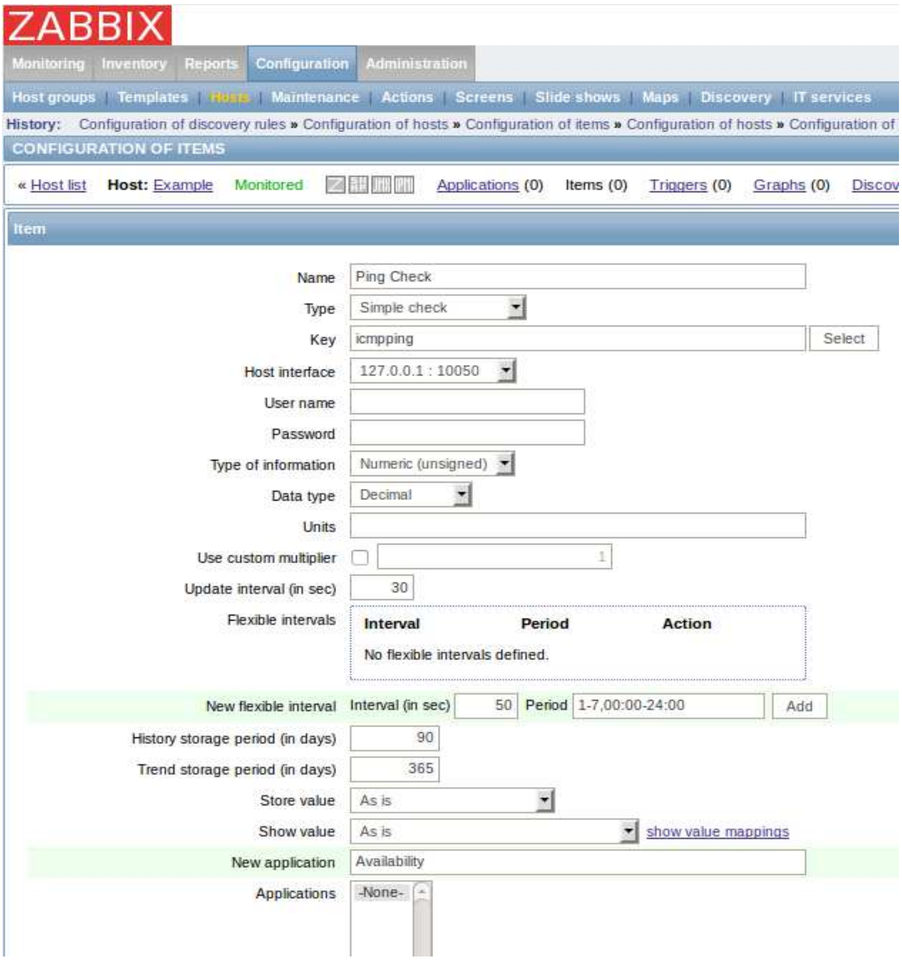
- Create Item

Name = Ping Check

Type = Simple Check

Key = icmpping (delete the options and all brackets < >)

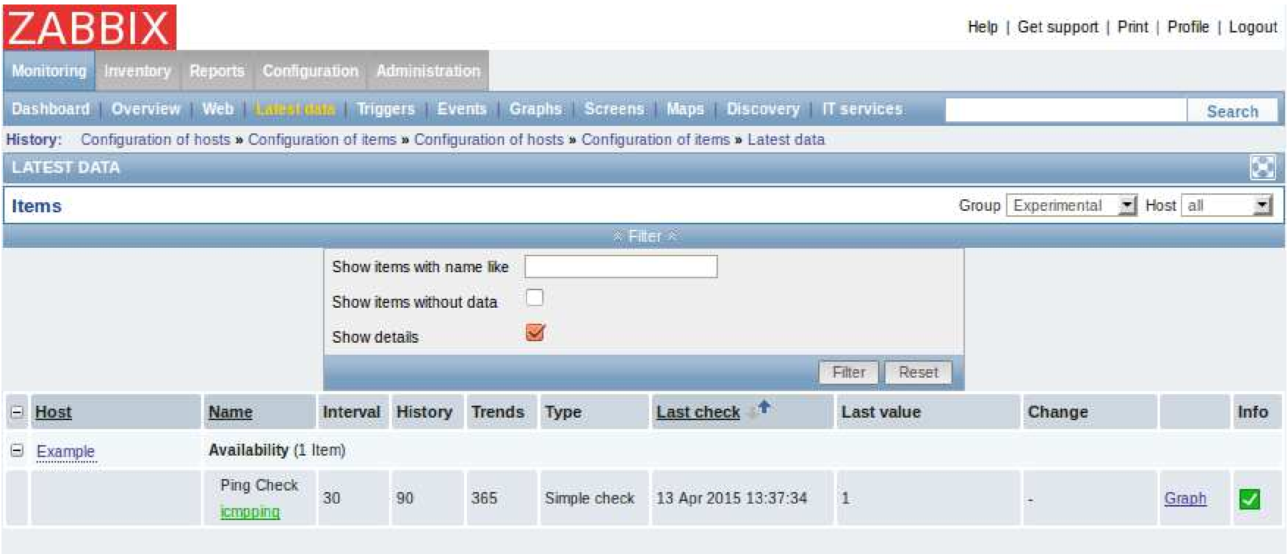
New Application = Availability


ZABBIX
Monitoring
Inventory
Reports
Configuration
Administration
Host groups | Templates | Hosts | Maintenance | Actions | Screens | Slide shows | Maps | Discovery | IT services
History: Configuration of discovery rules » Configuration of hosts » Configuration of items » Configuration of hosts » Configuration of
CONFIGURATION OF ITEMS
« Host list Host: Example Monitored Applications (0) Items (0) Triggers (0) Graphs (0) Discov
Item
Name Ping Check
Type Simple check
Key icmpping Select
Host interface 127.0.0.1 : 10050
User name
Password
Type of information Numeric (unsigned)
Data type Decimal
Units
Use custom multiplier 1
Update interval (in sec) 30
Flexible intervals Interval Period Action
No flexible intervals defined.
New flexible interval Interval (in sec) 50 Period 1-7,00:00-24:00 Add
History storage period (in days) 90
Trend storage period (in days) 365
Store value As is
Show value As is show value mappings
New application Availability
Applications -None-

- Save

c) Read the measured value

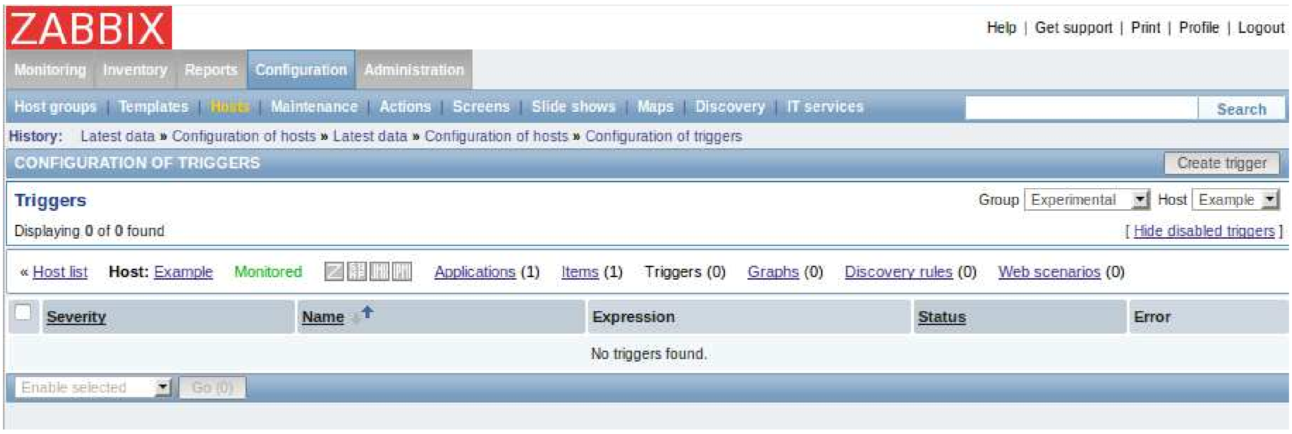
- *Monitoring → Latest data*
- *Group: Experimental (*wait for some seconds and reload the page)
- Last value = 1 → Ping is successful (0 = not successful)



d) Create the first trigger

- *Configuration → Hosts*

- Select *Triggers(0)*



- *Create Trigger*

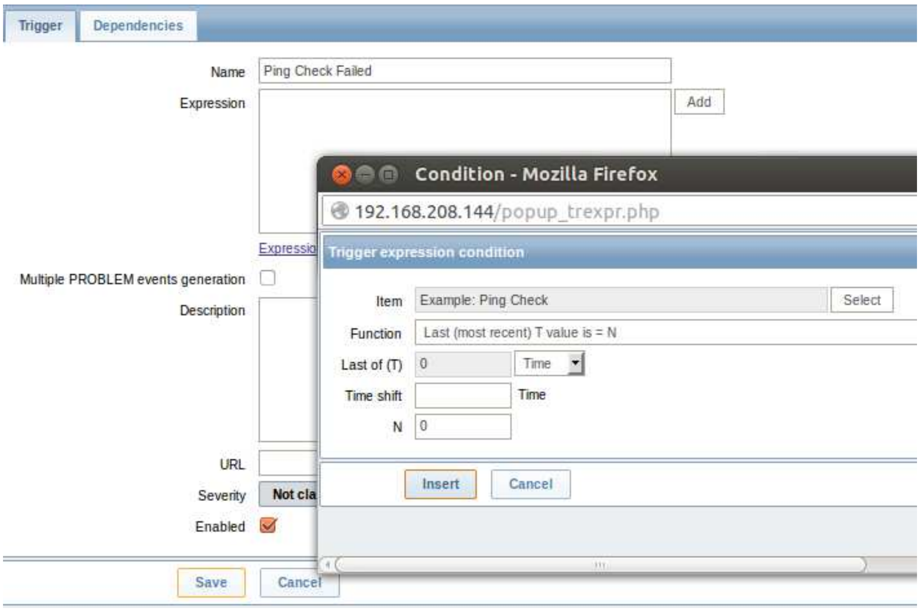
  Name = Ping Check Failed

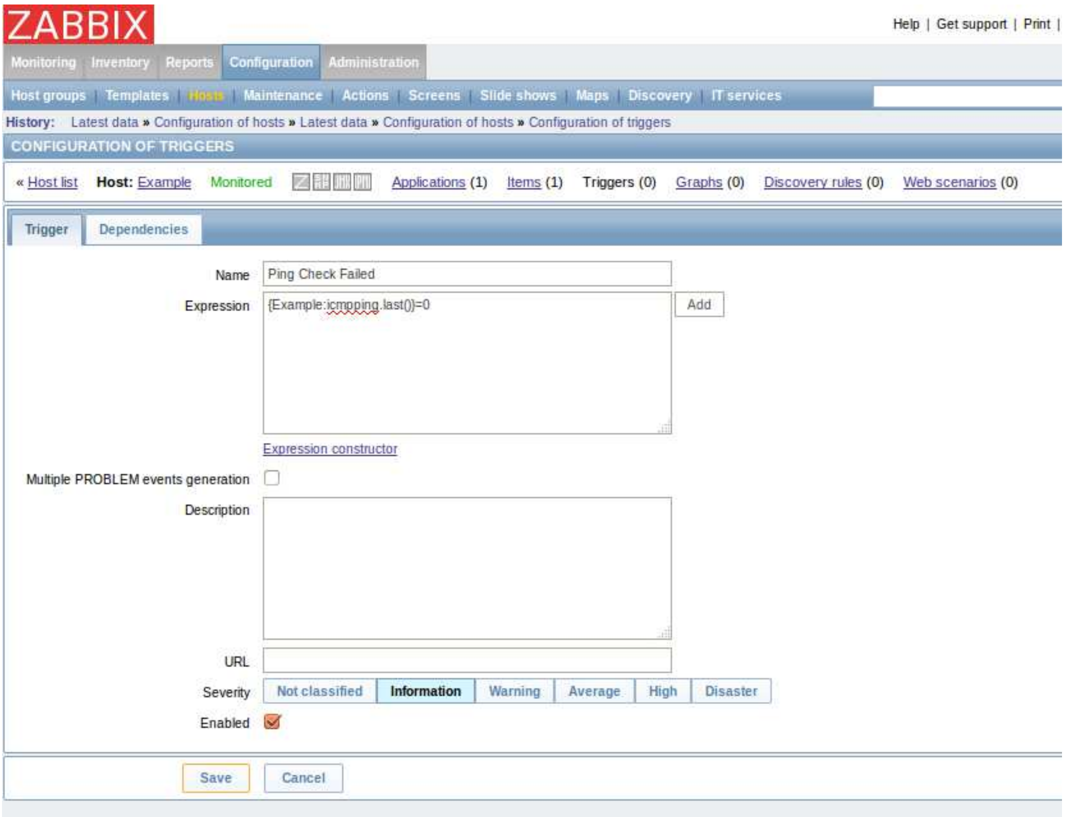
  Expression > Select *Add*

  Item > Select *Ping Check*

  Insert

  Severity = Information


Trigger
Dependencies
Name
Ping Check Failed
Expression
Add
Condition - Mozilla Firefox
192.168.208.144/popup_trexpr.php
Trigger expression condition
Item
Example: Ping Check
Select
Function
Last (most recent) T value is = N
Last of (T)
Time shift
N
Insert
Cancel
Multiple PROBLEM events generation
Description
URL
Severity
Enabled
Save


ZABBIX
Help | Get support | Print |
Monitoring Inventory Reports Configuration Administration
Host groups | Templates | Hosts | Maintenance | Actions | Screens | Slide shows | Maps | Discovery | IT services
History: Latest data » Configuration of hosts » Latest data » Configuration of hosts » Configuration of triggers
CONFIGURATION OF TRIGGERS
« Host list Host: Example Monitored Applications (1) Items (1) Triggers (0) Graphs (0) Discovery rules (0) Web scenarios (0)
Trigger
Dependencies
Name
Ping Check Failed
Expression
{Example:icmpping.last()}=0
Add
Expression constructor
Multiple PROBLEM events generation
Description
URL
Severity
Not classified Information Warning Average High Disaster
Enabled
Save
Cancel

- *Save*

e) Status of the trigger (wait for a few seconds and refresh page)

- *Monitoring → Triggers*

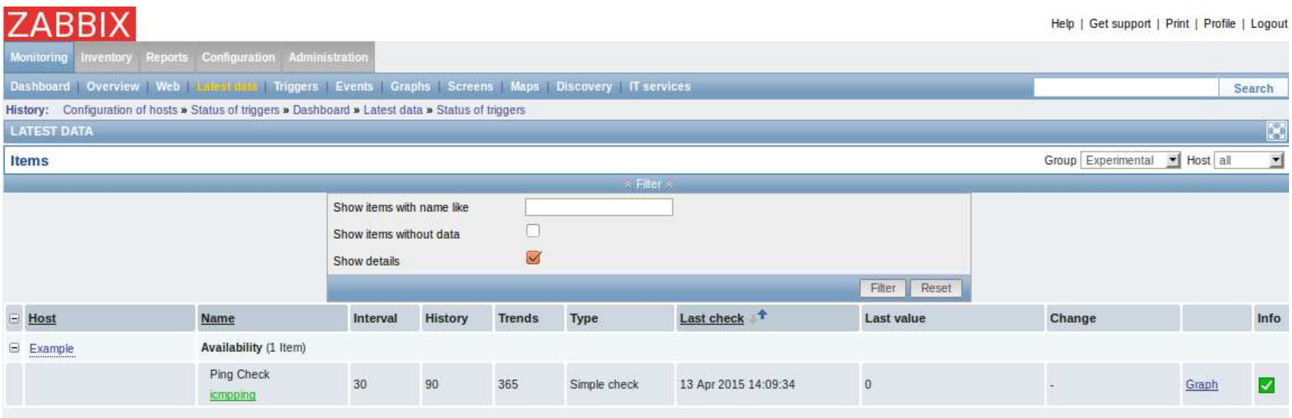
- *Status = OK*

f) Test your trigger

- *Configuration → Hosts*: set an incorrect IP (e.g. 192.0.0.1) for your host 'Example' and save the configuration
- *Monitoring → Triggers* (wait for a few seconds and refresh page)
- *Status = PROBLEM*

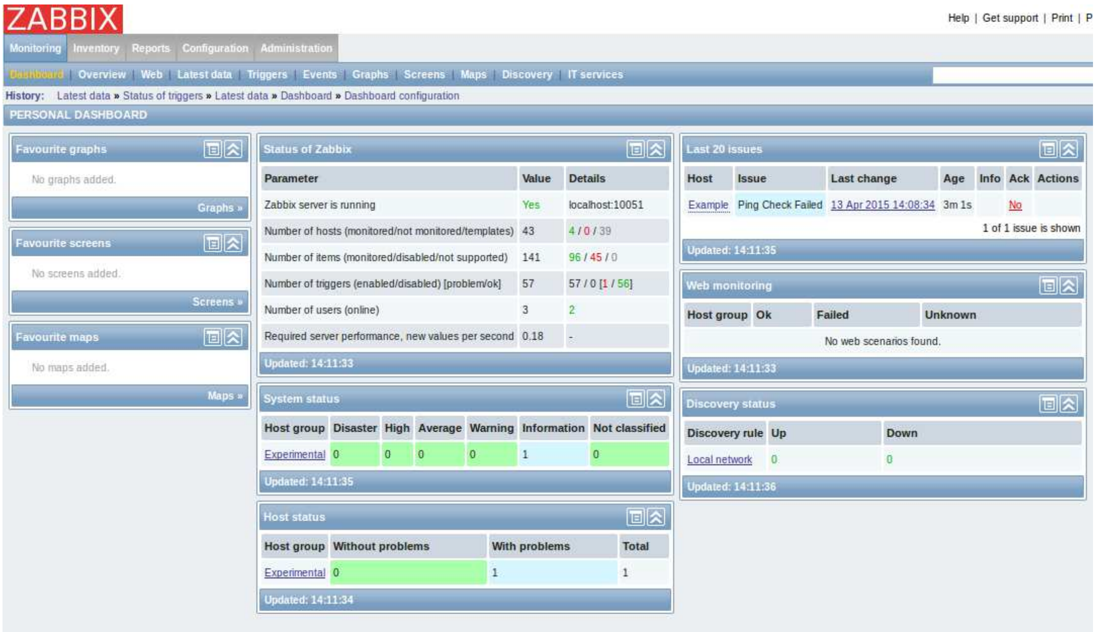
- have a look at other monitoring pages:

  Monitoring → Latest Data



Last Value = 0

Monitoring → Dashboard



IV. MONITORING OF THE INVAR-VALUES IN ZABBIX AS EXAMPLE

a) Create a host

Name = Invar sender

New Group = Invar Sender

IP = 127.0.0.1

Port = 10050

b) Create items

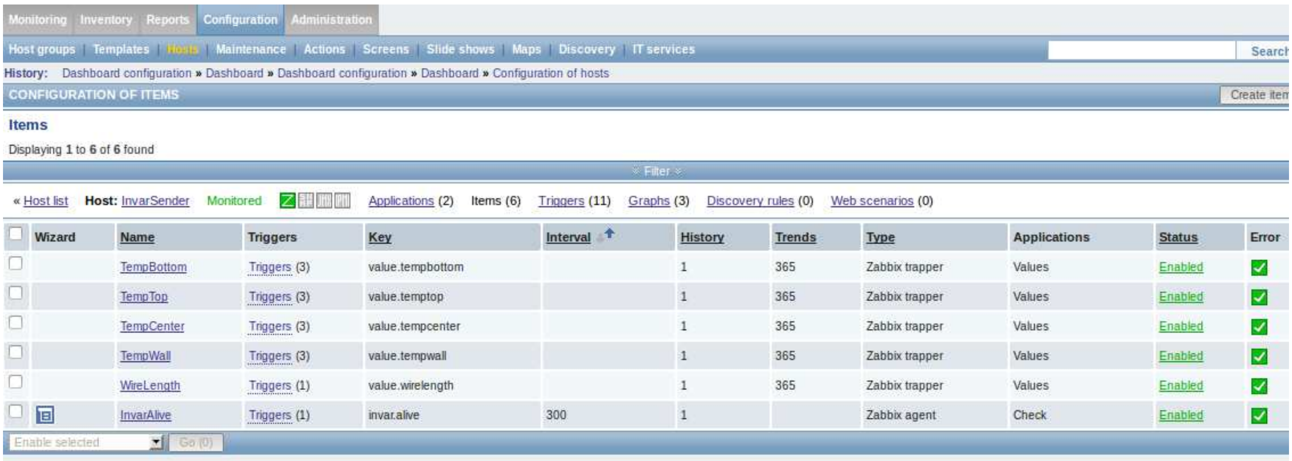
e.g. *Name = TempBottom (sensor temperature bottom)*

Type = Zabbix trapper

Key = value.tempbottom

Type of Information = Numeric (float)

Units = °C



Wizard	Name	Triggers	Key	Interval	History	Trends	Type	Applications	Status	Error
	TempBottom	Triggers (3)	value.tempbottom		1	365	Zabbix trapper	Values	Enabled	
	TempTop	Triggers (3)	value.temptop		1	365	Zabbix trapper	Values	Enabled	
	TempCenter	Triggers (3)	value.tempcenter		1	365	Zabbix trapper	Values	Enabled	
	TempWall	Triggers (3)	value.tempwall		1	365	Zabbix trapper	Values	Enabled	
	WireLength	Triggers (1)	value.wirelength		1	365	Zabbix trapper	Values	Enabled	
	InvarAlive	Triggers (1)	invar.alive	300	1		Zabbix agent	Check	Enabled	

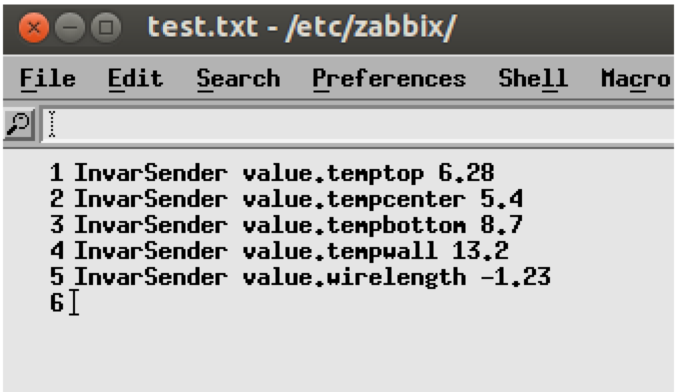
Store Value = As is

Save

(Sensors: TempBottom, TempTop, TempCenter, TempWall, WireLength)

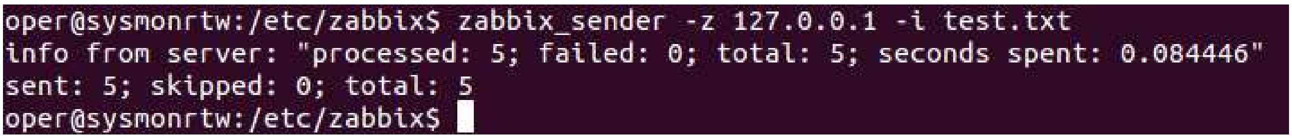
c) Create a Textfile with the values

Format: <hostname> <key> <value> (hostname and key like those you created in zabbix, values with a point)

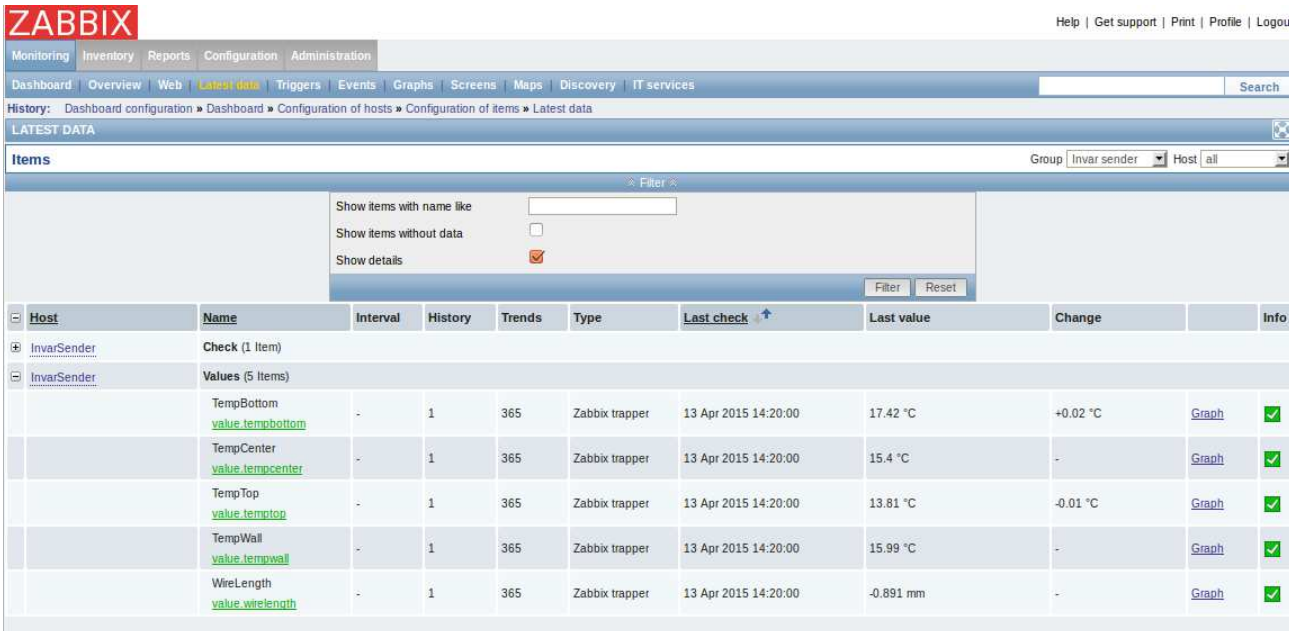


d) Send the file to zabbix (typing the following command into your terminal) and check the values of the items in the zabbix-system

zabbix_sender -z 127.0.0.1 -i test.txt

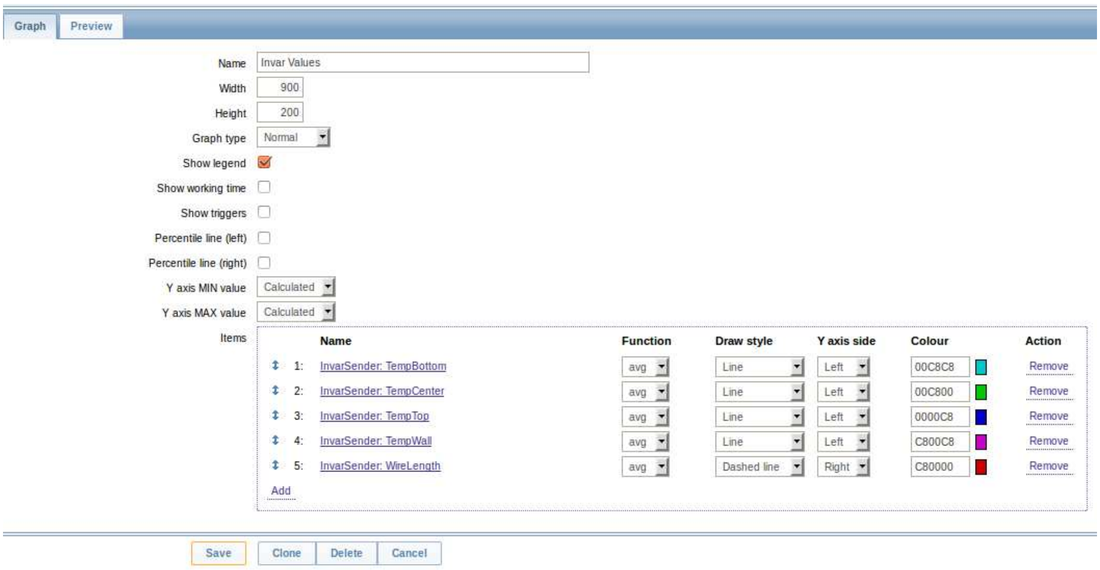


Configuration → Latest Data

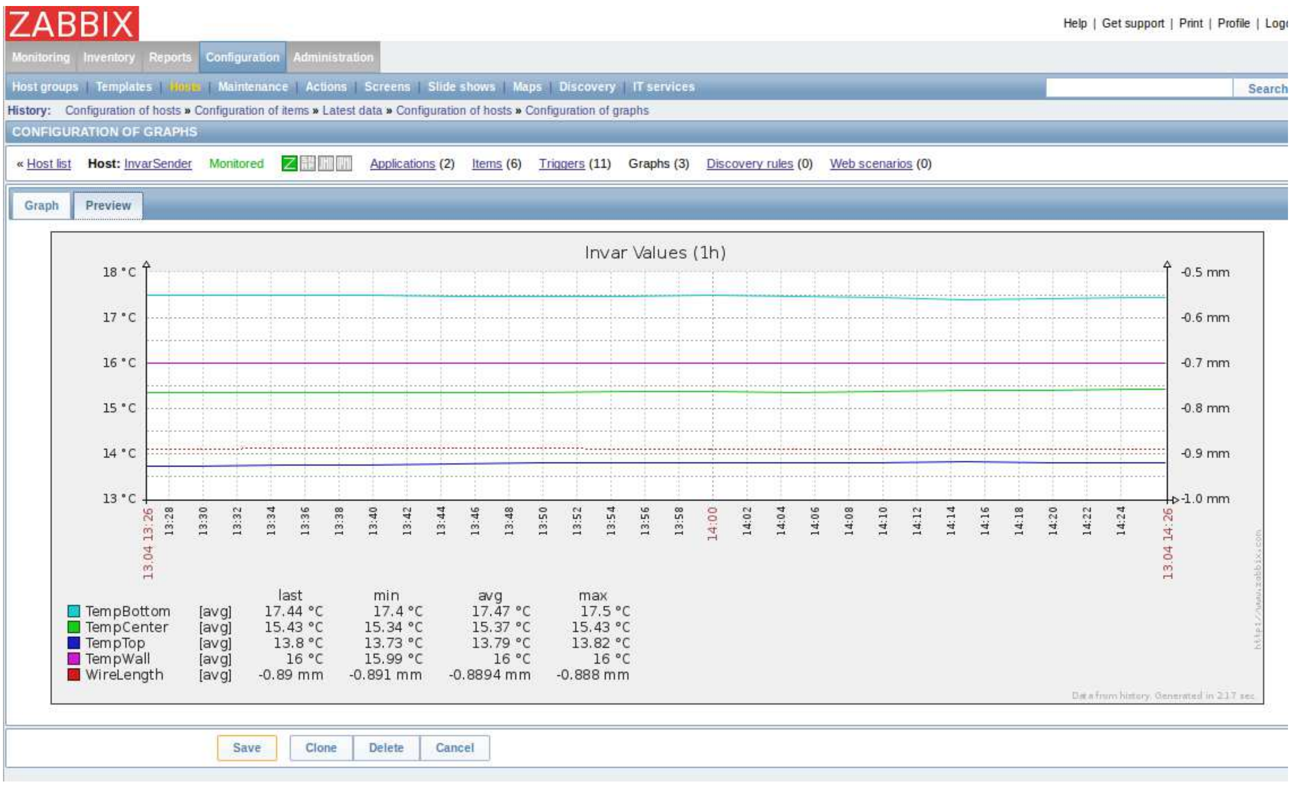


e) Create a graph

- *Configuration → Hosts*
- Select *Graphs(0)* of our host *InvarSender*
- *Create a graph* with our InvarSender-Items



- Control the *Preview* and *Save*



f) Create triggers for our items

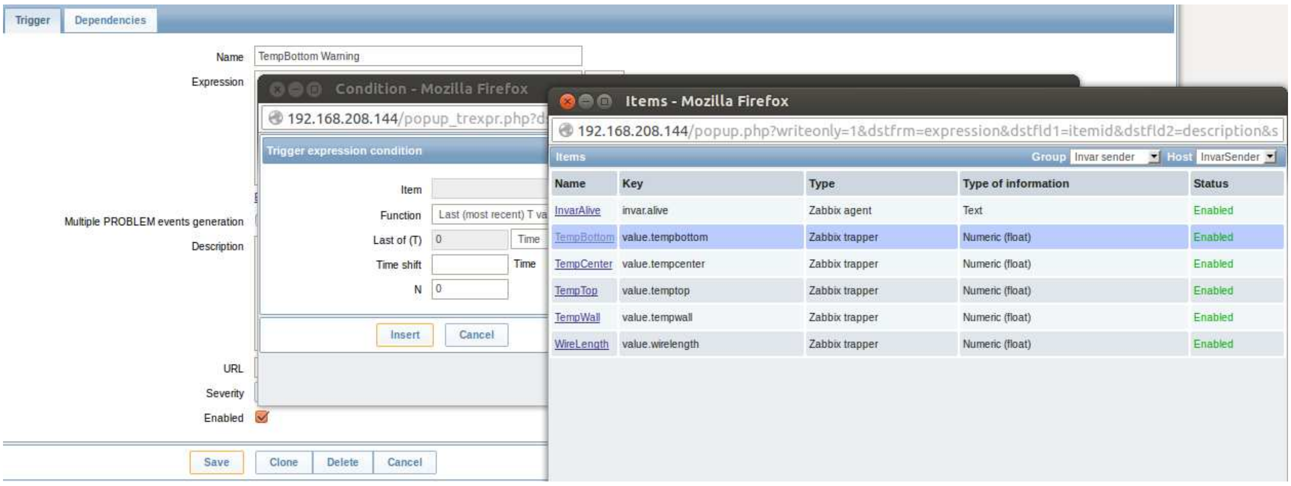
e.g. *Name = TempBottom Warning*

Expression > Add

Item > Select TempBottom

Function = Last (most recent) T value is < N

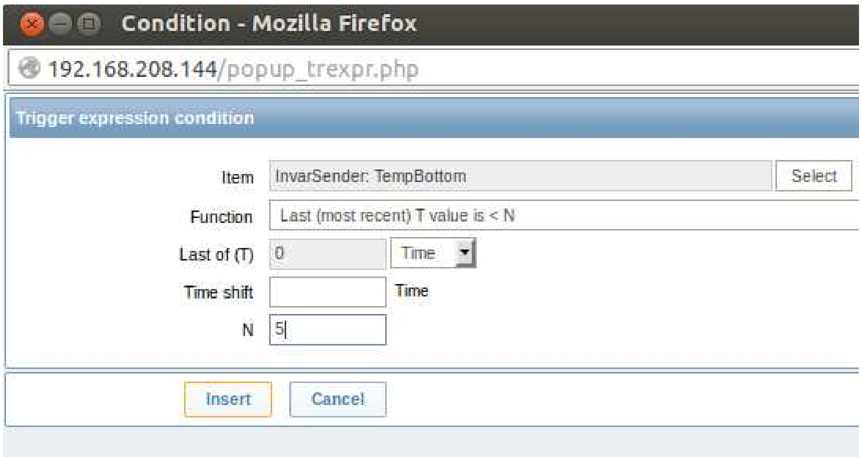
N = 5

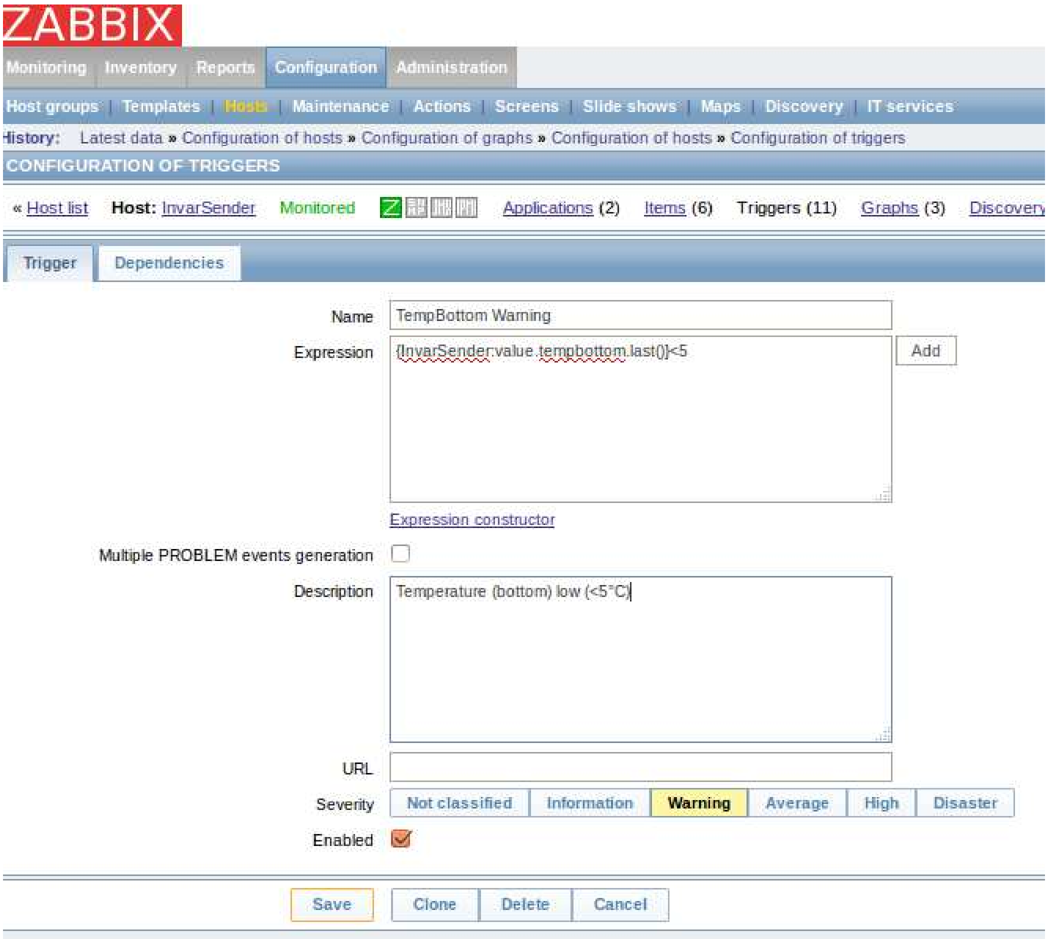


Insert

Severity = Warning

Save


Condition - Mozilla Firefox
192.168.208.144/popup_trexpr.php
Trigger expression condition
Item
InvarSender: TempBottom
Select
Function
Last (most recent) T value is < N
Last of (T)
0
Time
Time shift
Time
N
5
Insert
Cancel


ZABBIX
Monitoring Inventory Reports Configuration Administration
Host groups | Templates | Hosts | Maintenance | Actions | Screens | Slide shows | Maps | Discovery | IT services
History: Latest data » Configuration of hosts » Configuration of graphs » Configuration of hosts » Configuration of triggers
CONFIGURATION OF TRIGGERS
« Host list Host: InvarSender Monitored Applications (2) Items (6) Triggers (11) Graphs (3) Discovery
Trigger Dependencies
Name
TempBottom Warning
Expression
{InvarSender:value.tempbottom.last()}<5
Add
Expression constructor
Multiple PROBLEM events generation
Description
Temperature (bottom) low (<5°C)
URL
Severity
Not classified Information Warning Average High Disaster
Enabled
Save Clone Delete Cancel

7) Add our new invar graph to the dashboard

- *Monitoring → Dashboard*
- Select 
- Add our customized graph *Invar Values* to *Favourite graphs*

8) Create a map

- Configuration → Maps

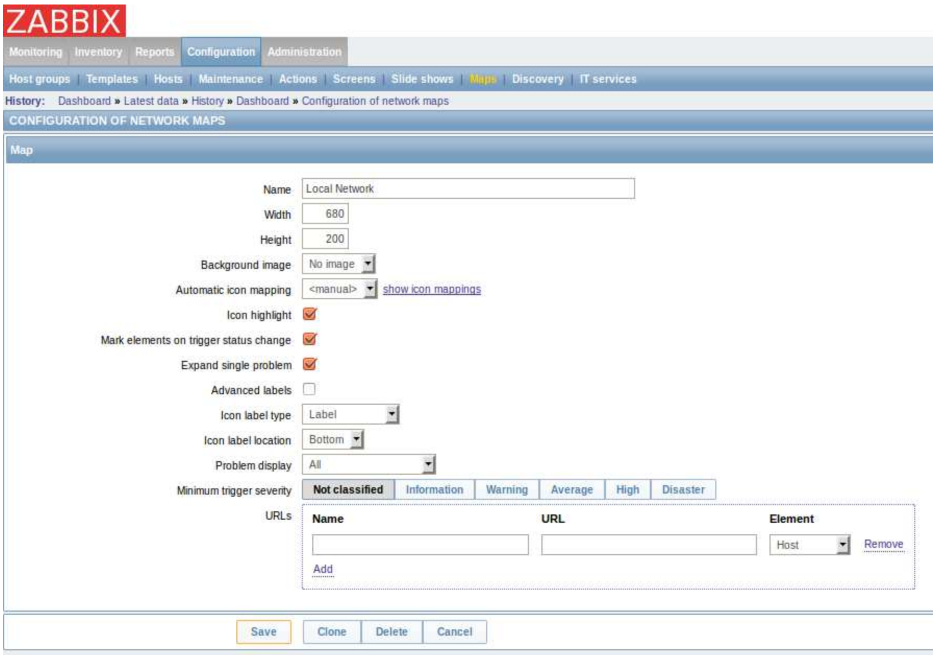
- Create Map

Name = Local Network

Width = 680

Height = 200

Icon Highlight = ☑



Mark elements … = ☑

Expand single problems = ☑

Save

- Select our new map *Local Network*

- *Icon +* (a new icon appears, for the invar-sender)

- *select* the new icon to configure it

  Type = Host

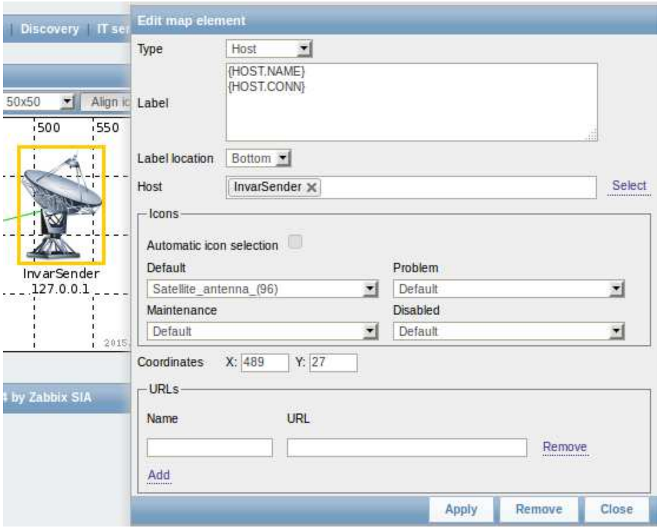
  Label = {HOST.NAME}

  {HOST.CONN}

  Host = InvarSender

  Icons = Satellite_antenna

  Apply



- *Icon + (for the zabbix-system itself)*

- select the new icon

  Type = Host

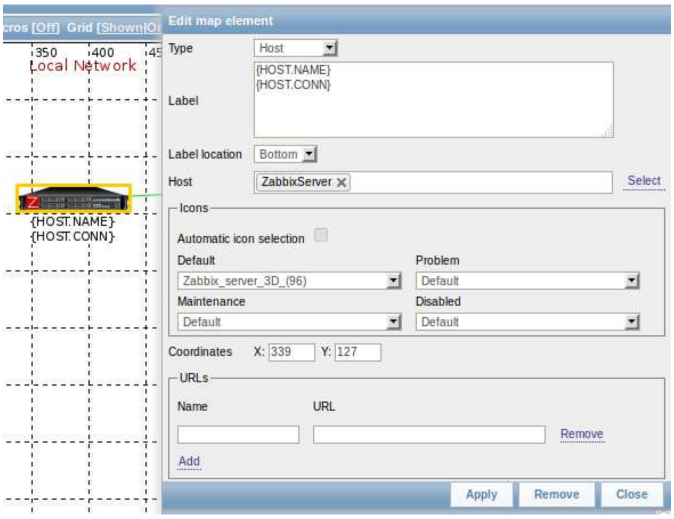
  Label = {HOST.NAME}

  {HOST.CONN}

  Host = ZabbixServer

  Icons = Zabbix_server_3D

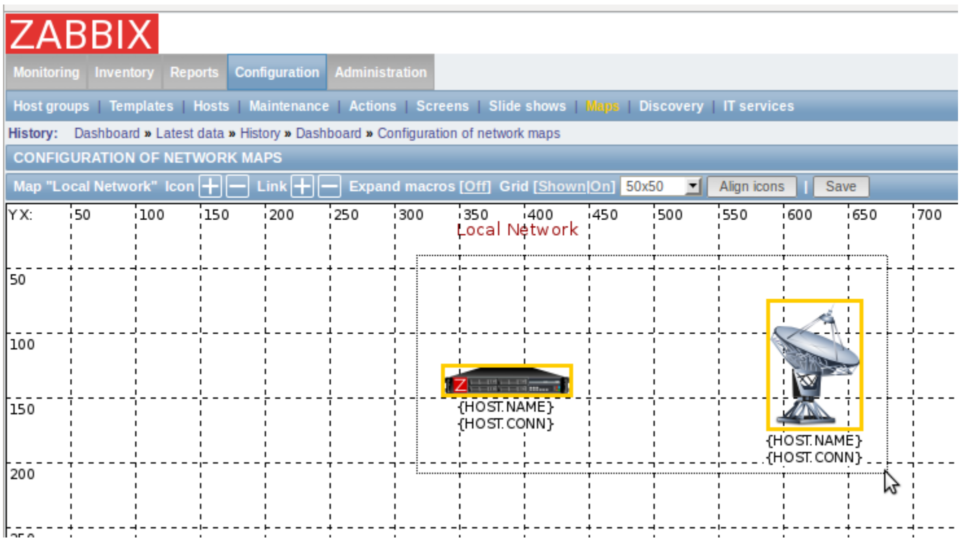
  Apply

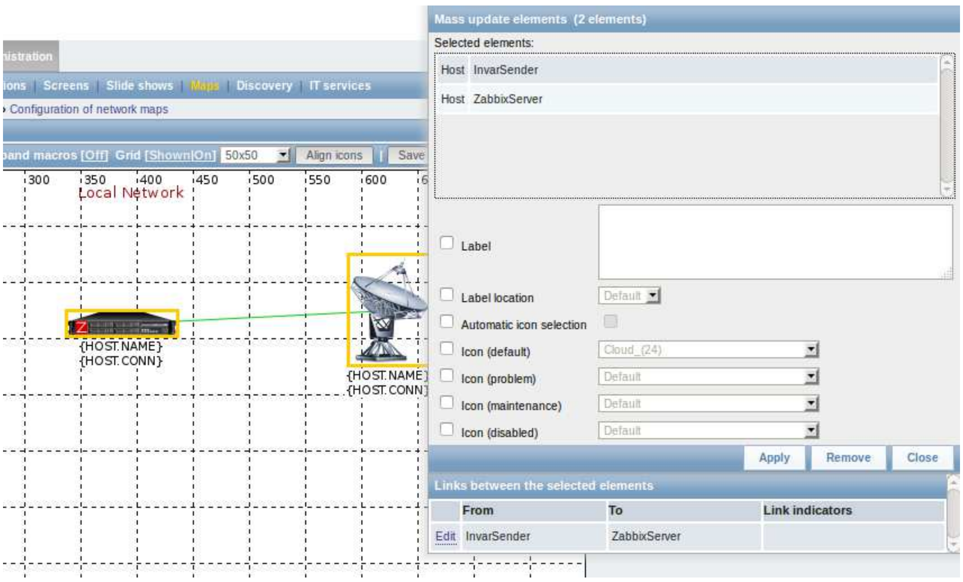


- Link the icons

    mark them and select *Link +* and *Apply1*

    Save the new map





9) Add the map to *Favorite Maps* in our dashboard

(see 7)

10) Create a screen

- *Configuration → Screens*
- *Create screen*

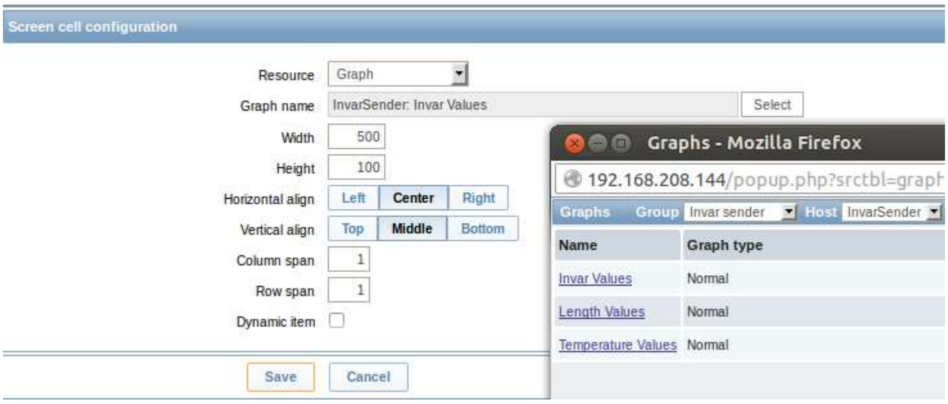
  Name = Invar Screen

  Columns = 2

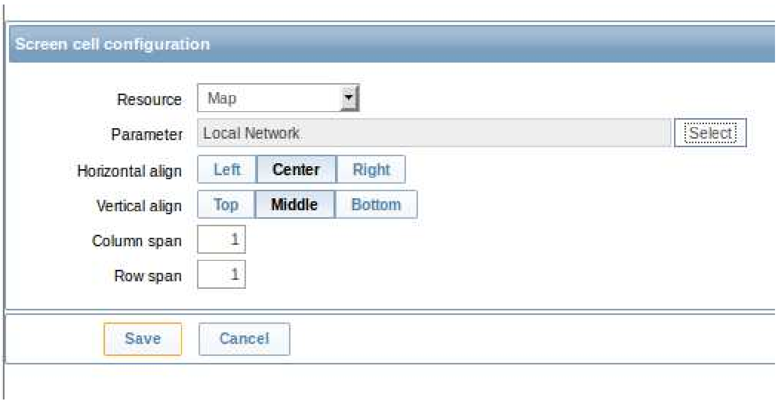
  Rows = 2

  Save

- Select the new screen
- Click on *Change* an select our graph *Invar Values*
- *Save*



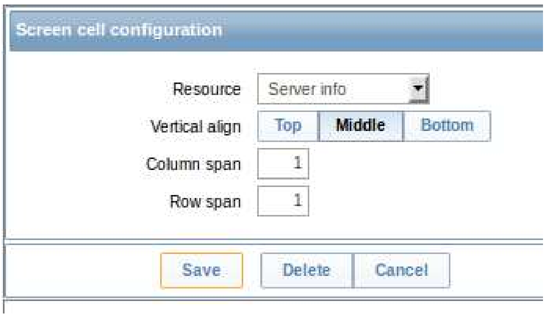
- Click *Change* in the other column and select our map *Local Network*
- *Save*



- *Click Change in a remaining column/row and select*

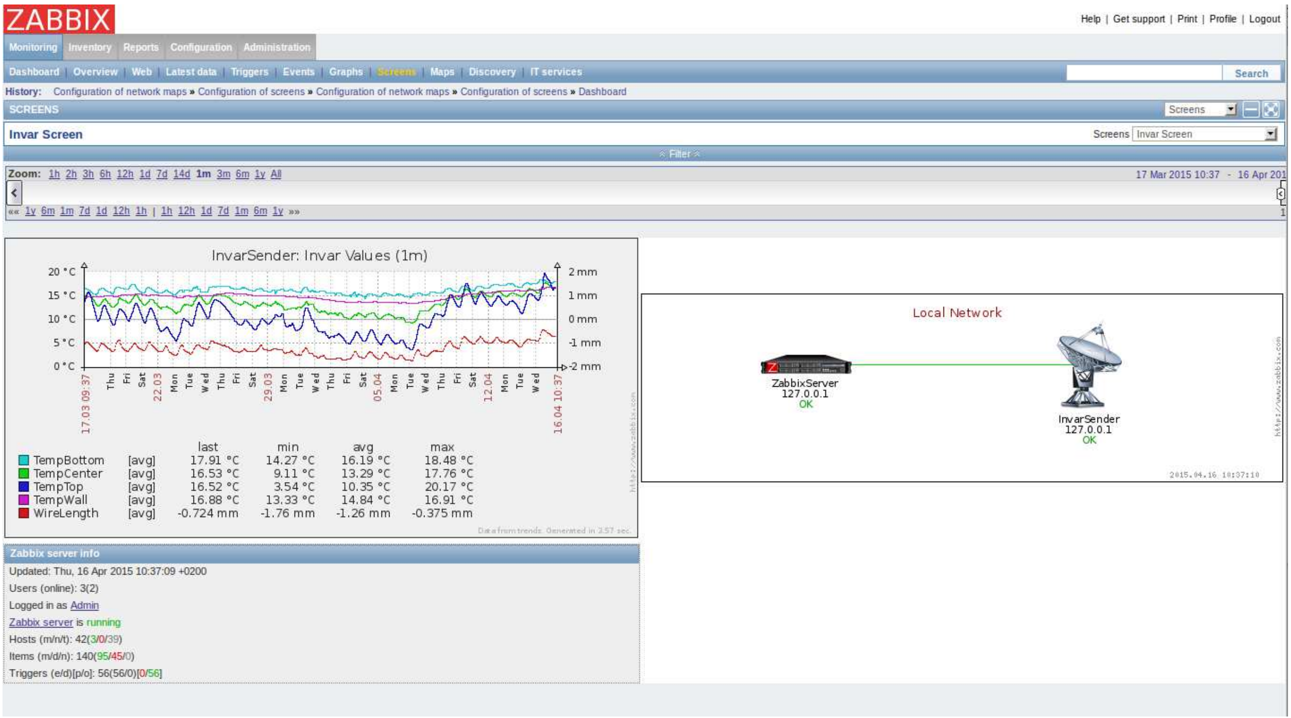
  Resouce = Server info

Save

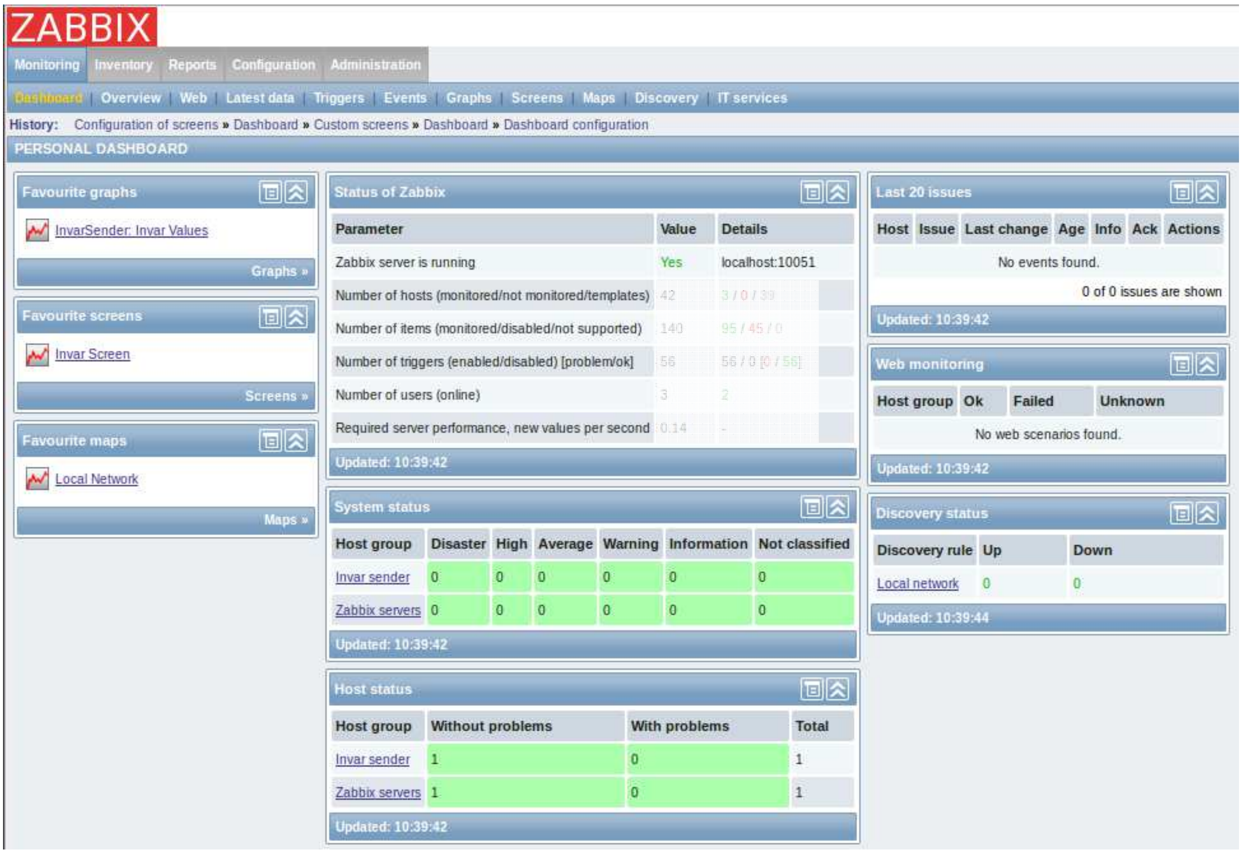

Screen cell configuration
Resource
Server info
Vertical align
Top
Middle
Bottom
Column span
1
Row span
1
Save
Delete
Cancel

11) Add the screen to the Favourite Screens in our dashboard

see 7) and open it



The zabbix-dashboard at the end of our workshop



12) Send a notification to the administrator/user in case of an error

- Administration → Media types (pre-defined media types)

  E-Mail: send an e-mail

  Jabber: instant messagging

  SMS: send a sms to the administrator/user

  or create an own media type

- Configuration → Actions

  adjust and enable the pre-defined action called 'Report problems to Zabbix administrators' with a new condition (e.g. Trigger = InvarSender TempBottom Warning) and the user you would like to notify

- Administration → Users (select user) → Media

  add a new media and the 'send to' user (e.g. e-mail, admin@zabbix.com)

Now, if the defined trigger occurs, the action will sent a notification to the user you defined.