Realization of a System to Control and Monitor the Operation of a Liquid Xenon Time Projection Chamber

Bachelorarbeit in Physik

von

Elvar Steinn Kjartansson

JOHANNES GUTENBERG
UNIVERSITÄT MAINZ

vorgelegt dem Fachbereich Physik, Mathematik und Informatik (FB 08)
der Johannes Gutenberg-Universität Mainz
am 8. Mai 2012

1. Gutachter: Prof. Dr. Uwe Gerd Oberlack
2. Gutachter: Prof. Dr. Lutz Köpke

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Elvar Steinn Kjartansson
Mitternacht 14, 55116 Mainz, Deutschland
kjartans@students.uni-mainz.de

Abstract

Modern experiments in physics, as in other areas of science, often depend on the function and cooperation of large amounts of different hardware components, each of which can malfunction at any time due to either human interraction, natural processes or unexpected hardware or software failure. As large amounts of funding and work go into these experiments, which often are supposed to run for weeks, months or years without interruption, it is important to monitor both the experiment's components and its physical parameters. While human monitoring is preferable it is not always possible due to practical and technical reasons, so other ways must be sought to ensure that in the case of a malfunction actions are taken to prevent the interruption of the experiment, or in the very least, further damage to its components.

This thesis documents the design and creation process of a Slow Control System and monitoring software, intended for use with the liquid Xenon time projection chamber currently being built by the Mainz XENON group[1] lead by Professor Uwe Oberlack at the Department of Physics at the Johannes Gutenberg University in Mainz, Germany.

The construction of such a system presents many challenges in both hardware and software as it needs to be capable of both reliably monitoring and controlling many different devices from different manufacturers, with both analog and digital inputs/outputs, and be reconfigurable for various possible experimental setups with minimal effort. It also needs to log the anomalies it detects and reliably alert the researchers in charge of the experiment by means of electronic mail and/or mobile phone text messages. Last but not least, the system needs to be expandable for future use in other experiments.

[1] For more information, see http://xenon.physik.uni-mainz.de/. Mainz XENON is part of the XENON Collaboration (http://xenon.astro.columbia.edu/)

Contents

1 Introduction

The luminous baryonic matter we are able to directly observe only accounts for around 4.5% [JBD+11] of the universe's total energy. Whereas 73.3% of the remaining energy is *dark energy* (Einstein's cosmological constant), the remaining 22.2% reside in so-called *dark matter*. What particles constitute dark matter is yet unknown but any candidate would need to fulfill certain criteria, discussed in the next section.

Numerous experiments are underway attempting to either directly or indirectly detect dark matter. One of these is the XENON100 experiment (and its successor, XENON1T) at the Gran Sasso underground laboratory in Italy. The Mainz XENON group at the University of Mainz is a part of the XENON collaboration and is independently working on a smaller-scale liquid Xenon two-phase time projection chamber to investigate the scintillation efficiency and ionization yield for nuclear recoils of liquid Xenon and the exact shape of the so-called *S1* pulse [Bes12], specifically at low energies (1 - 100 keV). The results of this experiment will be useful to improve background discrimination techniques in future liquid xenon experiments.

This bachelor thesis documents the design of a *Slow Control System* (SCS) for the *Mainz Liquid Xenon TPC* (Mainz TPC) and in particular the design, programming and testing of the accompanying monitoring software, dubbed *lXeMon* (Liquid Xenon Monitor).

This paper is structured as follow: after an introduction into dark matter physics and detection techniques, the planned Mainz TPC and its xenon circulation system will be described, followed by a detailed description of the SCS and the lXeMon software, for which the programming makes up the largest part of this project. In the appendices, one can find a guide to configuring lXeMon's server component and detailed programming documentation for lXeMon Server.

The information in this introductory chapter is mostly based on references [BHS05] and [AD10], unless stated otherwise.

1.1 Dark Matter

Almost simultaneously in 1932 and 1933, Jan Oort and Franz Zwicky, respectively, independently and on very different scales first found evidence of nonluminous *dark matter* [Tri87]. While Oort analyzed the stars and their velocities around the sun, Zwicky did measurements of the velocity dispersions of galaxies in the Coma cluster. Both came to the same baffling conclusion that significant amounts of mass seemed to be missing or invisible; Oort concluded that the stars were missing 30-50% of the matter implied by their velocities and Zwicky gathered that the galaxy clusters required 10 to 100 times their mass visible to stay bound.

Today, the existence of dark matter is considered all but certain, but its exact nature is still unknown and the subject of many theories and experiments, as well as the search for its particle's signature. The Standard Model in its current form does not provide a viable candidate particle for dark matter - meaning that dark matter searches represent the search for *new physics*, beyond the Standard Model.

1.1.1 Evidence for Dark Matter

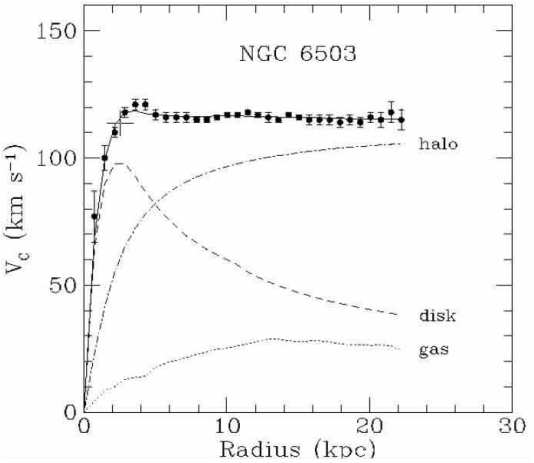

Figure 1.1: The rotational curve of the NGC 6503 spiral galaxy. The dotted line shows the contribution of gas, the dashed line that of a disk of luminous matter observed and the dotted-dashed line shows the proposed contribution of invisible dark matter (from [BHS05], original from [BBS91]).

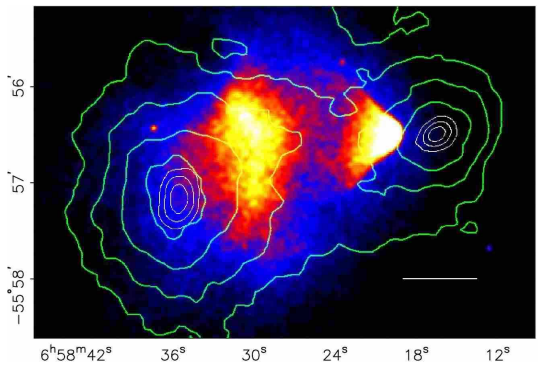

Figure 1.2: The 1E0657-558 *Bullet Cluster*. The colored image is taken in the X-ray region and shows the luminous mass while the green contour lines depict the reconstructed total mass potential lines (from weak lensing). From [CBG$^+$06].

There is substantial evidence for the existence of dark matter, the most convincing example of which is perhaps the historically oldest one: the failure to explain the observed rotational speed curves of galaxies (plotted over the distance from the center). Whereas Newtonian dynamics and the laws of gravitation predict a curve falling $\propto 1/\sqrt{r}$, the observed curves exhibit a roughly constant speed. A famous example of such a curve is provided in Figure 1.1. This discrepancy can only be explained by either introducing modifications to the laws of gravitation (e.g. for large scales only) or by assuming a halo of unseen mass.

Another piece of evidence in the case for dark matter is the observed strong gravitational lensing of some elliptical galaxies, the study of which can be found in reference [KT03].

In 2006, D. Clowe et. al. provided empirical proof for dark matter making up the dominant part of the 1E0657-558 galaxy cluster's mass using weak gravitational lensing (see reference [CBG$^+$06] and Fig. 1.2), independent of the cosmology of the universe. The two subclusters in the image are moving apart from each other after having collided approximately 100 Myr ago, and while the dominant part of their luminous mass can be seen with X-ray telescopy (colors in the picture), the total mass potential can be traced by using weak gravitational lensing seen in the visible region and is shown on the image as the green contour lines. While the visible matter (mostly plasma) was slowed down during the collision due to ram pressure, the vast majority of the mass seems to have passed through faster and is ahead of the visible cloud, indicating a form of matter that does not interract as readily. MOND (*MOdified Newton Dynamics*) and other alternative theories all fail to explain the results found by Clowe et. al., leaving dark matter as the hypothesis best in agreement with all modern findings.

Today's top dark matter candidates stem from theoretical models that go beyond the Standard Model as it is known today and therefore, we can hope to gain a better understanding of the universe by searching for direct signs of dark matter.

1.1.2 Properties of Dark Matter

The two most important properties of dark matter are already conveyed in its name: it does not emit electromagnetic waves (or interact with the electromagnetic force) and it is massive.

Analysis of the *cosmic microwave background* (CMB) by the WMAP (*Wilkinson Microwave Anisotropy Probe*, see [JBD+11]) experiment (see reference [JBD+11]) provide constraints for the abundance of dark matter in the universe, suggesting the densities of matter and baryonic matter listed in Table 1.1.

Description	Symbol	Density
Baryon density	Ω_b	0.0449 ± 0.0028
Dark matter density	Ω_c	0.222 ± 0.026
Dark energy density	Ω_Λ	0.734 ± 0.029

Table 1.1: Abundancies of energy and matter in the universe (data taken from [JBD+11]).

1.1.3 Candidates for Dark Matter

The many different suggested candidates for dark matter stem from many different theories, often extensions to the Standard Model, such as (most notably) *Supersymmetry* (SUSY) and *Kaluza-Klein theory* (KK-Theory). Here a few historic and current candidates are shortly introduced; for further candidates and information, refer to [BHS05].

MACHOs

MACHOs or *massive compact halo objects* are astronomical bodies that do not emit light, such as brown dwarfs, large dark planets and black holes. It was thought that an abundance of such MACHOs in the outer regions of galaxies could make up at least part of the dark matter halos, and while this is still feasible for small parts, our current understanding of the *Big Bang* and the early universe (through *Big Bang nucleosynthesis* and CMB analysis) rules out the possibility of dark matter being largely baryonic. Work by the MACHO Collaboration (see reference [MACHO00]) has largely confirmed this, finding that mass in MACHOs could only constitute 20% of a dark matter halo's mass.

Neutrinos

One of the more obvious candidates for dark matter are the Standard Model's own neutrinos, as they are both known to exist and believed to be massive. While neutrinos can certainly be categorized as *hot*[1] dark matter assuming they have mass, current limits on their maximum mass and abundance exclude them as an explanation for the vast majority of dark matter ($\Omega_\nu \lesssim 0.14$ using laboratory constraints on m_ν ($m_\nu < 2.05\,eV$), cosmological constraints from WMAP results lead to an abundancy of a whole magnitude less). Since neutrinos and MACHOs are effectively ruled out, the Standard Model does not provide any candidates for dark matter, making dark matter an important clue to *new physics* or physics *beyond the Standard Model*.

WIMPs

Weakly interating massive particles (WIMPs) are a class of dark matter candidates that are perhaps today the most widely accepted explanation for dark matter. WIMPs include the

[1] *Hot* dark matter refers to dark matter travelling at relativistic speeds, like neutrinos do. Current dark matter candidates are believed to be *cold*.

likely most popular dark matter candidate in SUSY's lightest stable particle, the *neutralino*, as well as the lightest Kaluza-Klein particle (the first KK excitation of the photon) from *universal extra dimensions* (UED) theories.

As their name suggests, WIMPs are subject to the weak force and so can interact with ordinary nuclei and be detected.

1.2 WIMP Detection

WIMPs can be directly detected when scattering off of ordinary nuclei, either elastically or inelastically. Although this sounds simple enough to detect, the cross section for such interactions is extremely small due to the fact they only interract via the weak force, making background noise discrimination (e.g. from cosmic muons and background gamma and beta radiation) a serious challenge.

1.2.1 Indirect detection

WIMPs are expected to undergo annihiliations, creating other elementary particles such as neutrinos, gamma-rays, cosmic positrons and anti-protons, which can be detected. Earth-based Čerenkov telescopes (e.g. VERITAS) and gamma-ray space telescopes (e.g. GLAST) can be used to attempt to identify WIMP annihilation products. Indirect detection efforts, when possible, generally focus their efforts on areas where dark matter is thought to be most dense and/or their annihilations most common, e.g. the Galactic center or inside the Sun.

Neutrino telescopes such as IceCube and ANTARES ([Zor12]) are also trying to search for neutrinos stemming from WIMP annihiliations. While neutrinos are not a direct product of such annihilations, they are expected to be produced in secondary annihilations and neutrino telescopes can provide unrivaled signal purity due to the nature of neutrinos.

1.2.2 Creation at Accelerators

Dark matter searches also take place in accelerators, such as the LHC, where possible SUSY or UED particles could to be produced in high energy collisions and indirectly detectable through MET (*Missing Transverse Energy*) signals and some of their attributes determined through spin/parity analysis. Such experiments are highly complimentary to astrophysical ones to obtain a better understanding of WIMP candidates. ATLAS is an example for an experiment at the LHC which partly focuses on searching for signs of dark matter (see reference [ATLAS08] for more information).

1.2.3 Direct detection

Direct detection of a WIMP is possible in the form of detecting a nuclear recoil caused by it. Dark matter WIMPs interact with baryonic matter through the weak force only, which means the cross-section for such interactions is extremely small and also depends on other unknown attributes of the WIMPs such as mass.

The main challenges for direct detection techniques are the extreme amounts of background from other natural sources which have a much higher cross section and so dominate the raw signals. To avoid cosmic muon background, for example, most WIMP detectors are placed underground and inside large veto volumes made of low-radiation heavy elements such as liquid xenon and shielded with lead or copper. However, that still does not eliminate noise from gamma rays and alpha/neutron radiation background in the detector itself, which can only be mitigated with careful material selection and very efficient background

discrimination techniques during the data analysis.

In the following section, the principle of noble liquid TPC detectors is explained, which are currently the leading WIMP detectors for direct detection [XENON10011].

1.3 Noble Liquid Two-Phase TPCs as Dark Matter Detectors

Noble liquid two-phase time projection chambers are highly sensitive state-of-the-art detectors for detecting nuclear recoils caused by WIMPs. Utilizing a noble gas in both its liquid and gas phases as its target and scintillation material, a TPC can reconstruct the x, y and z coordinates of detected events using two separate signals called *S1* and *S2*. S1 is the primary signal, caused by the photons released when a noble gas excimer relaxes into its ground state after having been excited by a recoil event. Due to an applied strong electric field within the chamber, the electron-ion pairs produced in the event do not get a chance to recombine but instead drift towards the anode and cathode, respectively. Once the electrons drifting towards the anode enter the gas phase, their drift speed increases drastically allowing them to quickly gain enough energy to excite noble gas atoms, causing scintillation light proportional to the electron count measured as the S2 signal. A schematic of a dual-phase noble liquid TPC can be seen in Fig. 1.3

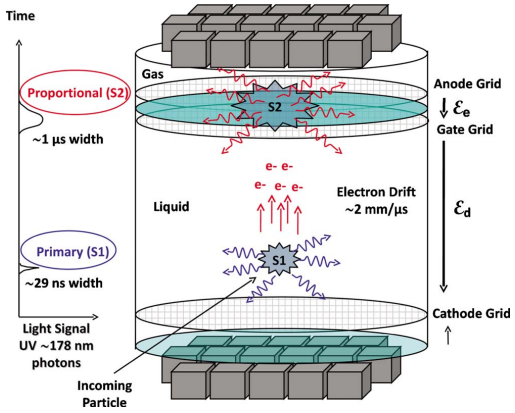

Figure 1.3: A schematic of a dual-phase noble liquid TPC (from [AD10]).

Using arrays of PMTs or APDs, a two-phase TPC can reconstruct the event's position, By measuring the time difference between the S1 and S2 signals and applying known drift velocities of electrons in noble gases and liquids, the z coordinate can easily be calculated, while the x and y coordinates can normally be deduced from the distribution of the S2 signal across the PMT arrays. The ratio between the S1 and S2 intensities depends on the event type and can be used to discriminate between WIMP recoils, gamma radiation and other background noise.

1.3.1 Choice of Target Material

Noble gases in their liquid form are the preferred target material for a TPC like the one described above as they can be readily purified to extremely high degrees. If not purified, impurities in the liquid would stop the electrons drifting to the anode and so prevent the S2 signal happening.

Noble liquids also have an extemely high scintillation yield, comparable to that of a NaI(Tl) crystal[2], and are transparent at the wavelengths they generate, making them excellent scintillators.

Among the noble liquids, only xenon and argon share the unique ability to produce both photons and charge carriers in response to radiation, allowing for detection of two separate signals, making

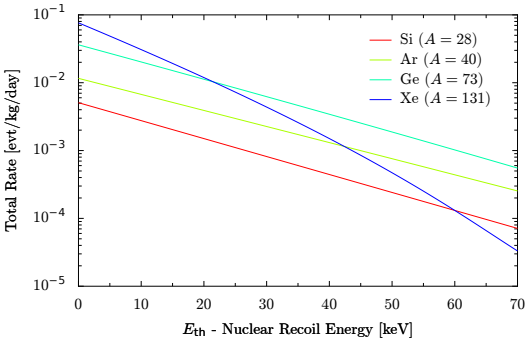

Figure 1.4: The calculated total rate of WIMP interactions for different materials, using a WIMP mass of 100 GeV/c^2 and cross-section of 10^{-43} cm^2. (From [ABC07])

[2] LAr: $4.0 \cdot 10^4\, ph/MeV$, LXe: $4.2 \cdot 10^4$, NaI(Tl): $4.3 \cdot 10^4$

them the top choices as targets for a TPC of the type discussed here.

Some of the most important attributes of liquid xenon and liquid argon are listed in Table 1.2. The high density (desirable to increase detection efficiency) and relatively high boiling point (allowing for simple and reliable cryogenic cooling systems) of xenon clearly speak in its favour, while argon certainly is more readily available (hence cheaper) and has some very interesting attributes such as the large difference between its slow and fast scintillation decay times.

Material	Scintillation yield [ph/MeV]	Light wavelength [nm]	Density [g/cm3]	Fast signal delay τ_f [ns]	Slow signal delay τ_s [ns]	Boiling point [K]
LXe	$4.2 \cdot 10^4$	177.6	2.95	4.3±0.5	21±2	165.15
LAr	$4.0 \cdot 10^4$	128	1.40	6.8±1.0	1550±100	87.35
NaI(Tl)	$4.3 \cdot 10^4$	415	3.67	≈250	N/A	N/A

Table 1.2: Comparison of some attributes of liquid xenon and liquid argon. Sodium Iodide is included for comparison. (Sources: [HTF+83] and wolframalpha.com)

As seen in Figure 1.4, xenon is the clear front runner when it comes to WIMP interactions at low energies, beating argon's interaction rate by a factor of 10 when approaching zero energy. For this reason, xenon is used in experiments such as the Mainz Xenon TPC discussed in 2, the XENON100/XENON1T experiments and more, while argon is the material of choice in other experiments such as WArP (see http://warp.lngs.infn.it/).

1.3.2 The XENON100 and XENON1T Experiments

The current leader in setting nuclear recoil cross section limits on WIMPs is the XENON100 experiment. XENON100 uses a two-phase liquid xenon TPC similiar to the one described in this paper (see chapter 2 and Figure 1.3) with 62 kg of liquid xenon target material and an additional 99 kilograms of active scintillator veto. 242 PMTs in two arrays (top and bottom) detect the scintillation signals and provide x,y and z resolution for detected events.

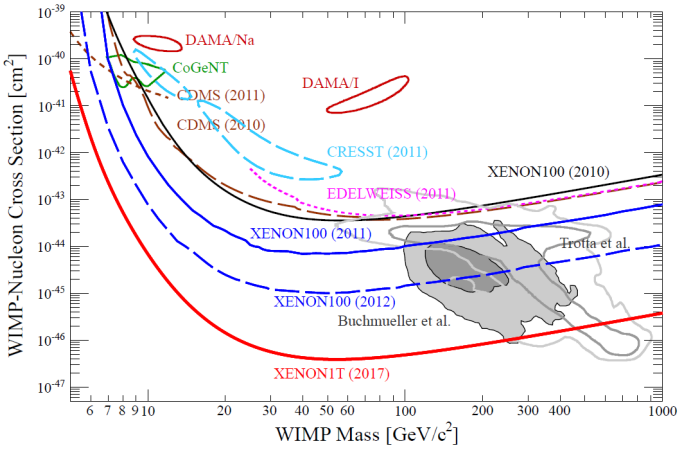

Figure 1.5: The WIMP cross section limits set by current and future dark matter experiments. Possible parameter spaces from selected simulations and theories are drawn as areas. (Source: http://xenon.astro.columbia.edu/XENON100_Experiment/ [retrieved on 3.5.2012])

XENON1T is currently in the planning phase and scheduled to start operation in 2014. As a successor to XENON100, it is expected to have enough sensitivity to cover the majority of the theorized parameter space for WIMPs (see Figure 1.5).

2 The Mainz Xenon TPC and Xenon Circulation System

The *Mainz Xenon Time Projection Chamber* (Mainz TPC) is a two-phase liquid xenon TPC currently being planned and built at the Department of Physics at the University of Mainz. Lead by Prof. Uwe Oberlack, the Mainz TPC's aim is to study the scintillation efficiency L_{eff} as well as charge and scintillation yield of both gamma and nuclear recoils in liquid xenon at lower energies (1-100 keV) where data is still lacking (see Figure 2.1), and to verify and/or improve old values[1] for decay times and ratios of the singlet and triplet excited states of xenon using state of the art high resolution electronics. Of special interest is the exact shape of the S1 pulse and the S1/S2 ratio for low-energy nuclear recoil signals.

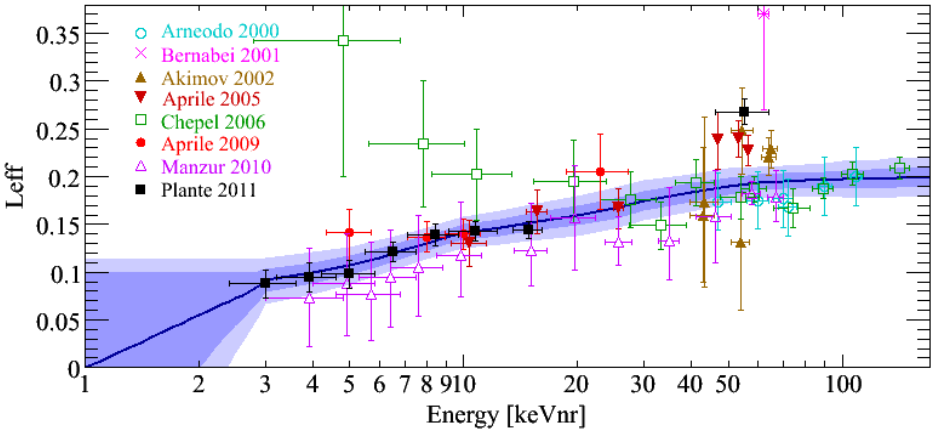

Figure 2.1: The scintillation efficiency L_{eff} over the nuclear recoil energy E_{nr} for liquid xenon. Notice the lack of data for $E_{nr} < 3\,keV$ (from [XENON10011])

During the first few months of Mainz TPC's operation, a 122 keV gamma source (^{57}Co) will be used to study signals caused by photon scattering events and their light yield, later to be replaced by a neutron source for the second operational phase where nuclear recoil events will be studied and their light yield compared to that of the gamma source to calculate the scintillation efficiency.

Better knowledge of the scintillation efficiency and the S1/S2 ratio for both gamma radiation and neutron recoils at such low energies will help improve background discrimination of ongoing and future dark matter experiments that use liquid xenon as the target scintillation material, as WIMP recoils are harder to distinguish from gamma background in the lower energy region. Studying the exact pulse shape of S1 will also help improve background discrimination.

[1] $\tau_{triplet} = 21 \pm 2\,ns$, $\tau_{singlet} = 4.3 \pm 0.5\,ns$ for neutron recoils without electric field, from [HTF$^+$83].
$\tau_{triplet} = 27 \pm 1\,ns$, $\tau_{singlet} = 2.2 \pm 0.3\,ns$ for high-energy electron scattering with a $4\,kV/cm$ electric field [KHR78]

2.1 Experimental setup

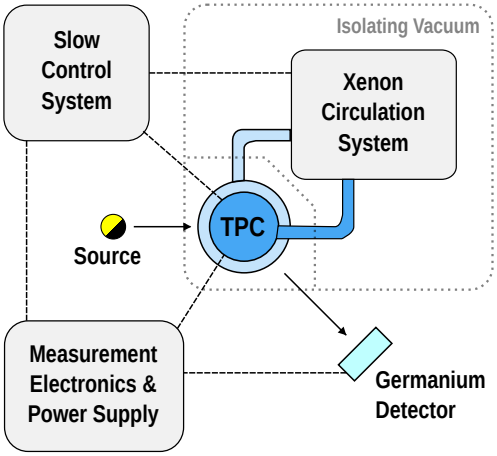

Figure 2.2: Overview of the Mainz TPC setup.

The Mainz TPC setup will consist of 3 subsystems: The TPC itself, its xenon circulation system (XCS) and the Slow Control System (SCS). The TPC is the core of the experiment, providing us with the measurement data required to draw physical conclusions and improve our understanding of scintillation in liquid xenon. As a means to keep the physical parameters in the TPC correct (temperature, liquid level, xenon purity etc.) the XCS circulates, purifies and cools the xenon while the SCS controls heater voltages and monitors many critical parameters of the experiment.

A radioactive gamma or neutron source (depending on the experiment phase) will be placed outside the TPC and cryostat, providing the particles for the electron scattering or nuclear recoil events, respectively, inside the TPC.

Additionally, a second germanium detector will be installed on an arm rotatable around the TPC. With this additional detector, the angle of particles scattered in the TPC can be determined, allowing calculations of the energy deposited by the particle and time-of-flight measurements as well as providing a possibility for external triggering of measurements, potentially reducing background noise.

Component	Amount	Manufacturer and model	Subsystem
PMTs	2	*Hamamatsu R6041*	TPC
APDs	8	*RMD S1315P*	TPC
HV Power Supply	1	*iseg NHQX3X*	TPC
Flash ADC (PMTs)	1	*Struck SIS3305*	TPC
ADC (APDs)	1	*Struck SIS3301-105*	TPC
Pump	1	TBD	XCS
Flow meter	1	*Hastings HFM-200*	XCS
Vacuum gauge	1	*Inficon VGC402*	SCS
Level sensors (TPC)	3-4	TBD	SCS / TPC
Level sensor (CV)	1	TBD	SCS / XCS
Cryocontroller	1	*SRS CTC 100*	SCS
USB DAQ/ADC	1	*MC USB 1608G*	SCS

Table 2.1: A list of the components used for the Mainz Xenon TPC. TBD stands for *to be decided*.

2.2 The Time Projection Chamber

The time projection chamber itself is the most important part of the setup and is comprised of a specially designed cylindrical container made of carefully chosen materials to house the liquid xenon target, two compact *Hamamtsu R6041* PMTs for measuring the S1 and S2 signals, eight APDs for the x/y event location measurement and additional S2 signal sensitivity, and a sealed outer container to house the TPC container, PMTs and APDs submerged in liquid xenon.

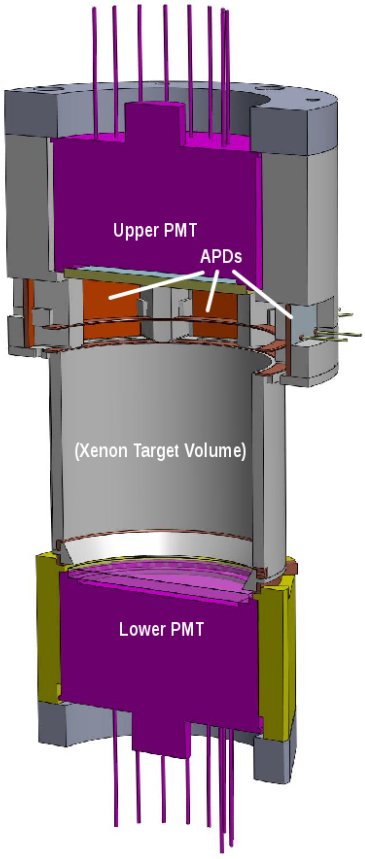

Figure 2.3: A cut through the inner TPC container with PMTs and APDs (Source: Mainz XENON group).

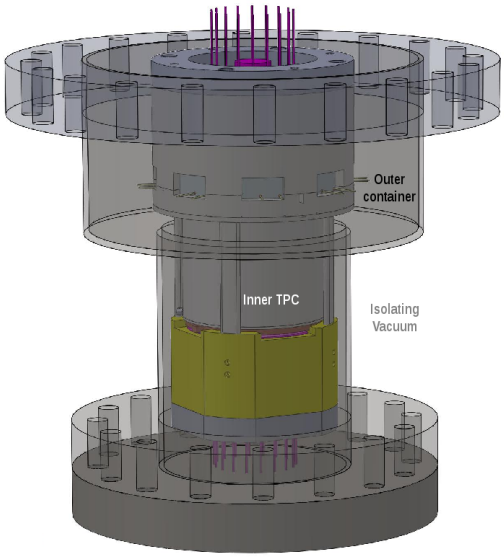

Figure 2.4: A view of the outer container containing the TPC shown on the left. (Source: Mainz XENON group).

The TPC's inside volume will have a diameter of 53 mm and the distance between the cathode and gate meshes will measure 50 mm, yielding a total of 110.3 cm^3 liquid xenon target volume. A schematic showing the inner container is shown in Figure 2.3. Above the liquid xenon, roughly a centimeter (exact measure to be decided) of gaseous xenon will serve as the target for drifting electrons to generate the S2 signal.

The TPC volume's inside cylinder will be made of PTFE teflon which reflects the scintillation light and outside it will be a printed circuit board which is designed to help form the electric field within the TPC to allow maximum uniformity. The electric field is applied with the use of 3 oxygen-free copper meshes: a cathode mesh (-20 kV) at the bottom of the volume, a gate mesh (-5 kV) 50mm above the cathode mesh and 2.5 mm below the liquid level of the xenon and an anode mesh (0 kV) 5mm above the gate mesh and 2.5 mm above the liquid level. This setup yields a 3 kV/cm electric field in the target volume and a much stronger 10 kV/cm electric field over the liquid-gas interface, enabling efficient extraction of the drift electrons.

Three or more capacitance level sensors will be placed in the outside TPC container, equally distributed around the inner cylinder covering liquid level heights of 45-55mm (measured from the cathode; the normal operating level will be at 52.5 mm). Using these level meters, the inclination of the detector with regards to earth's gravity can be precisely determined. A similiar but longer level sensor will also be placed inside the control volume.

Unlike noble liquid TPCs searching for WIMPs (XENON100 etc.), the outer container's size should be minimal as the xenon in it is a very effective shielding material, blocking the entry of gamma rays and neutrons used for experimenting. As in any other scattering experiment, the amount of material between the source and the scattering volume should be kept at a minimum.

2.3 The Xenon Circulation System

An important part of the Mainz TPC is its xenon circulation system, responsible for keeping the xenon in the TPC at the right temperature, purified and at the right level (with help from the SCS). A schematic of the currently proposed system is shown in Figure 2.5, but as it will not be finalized before this paper is due, it is only included to give a rough idea and not intended as a reference.

The xenon in the TPC is constantly circulated to keep it purified and cooled. A pump running at a constant rate provides the required flow, measured by the flow meter. A bottle containing pressurized xenon gas is connected to the system via the same route as the pump, so that by closing the valve between the flow meter and the pump and opening the valve between the gas bottle and flow meter, one can fill the system with new gas from the bottle.

Xenon gas entering the system or circulating through the pump will go through the *getter* to rid it of impurities, before entering the cooling volume, which the self-regulating cooler will keep at a constant temperature, inside an isolating vacuum chamber where it becomes liquified before entering the liquid phase of the TPC. A heat exchanger could reduce the required cooling power by letting the cold xenon leaving the isolating vacuum cool down the warm xenon entering it; whether such a heat exchanger will be included in the final design is however still uncertain.

To control the liquid level in the TPC, a separate control volume is directly connected to the TPC volume, containing a resistive heater element in the gas phase, to be controlled by the Slow Control System. Heating the gas causes it to expand in volume, causing the liquid level in the TPC to raise.

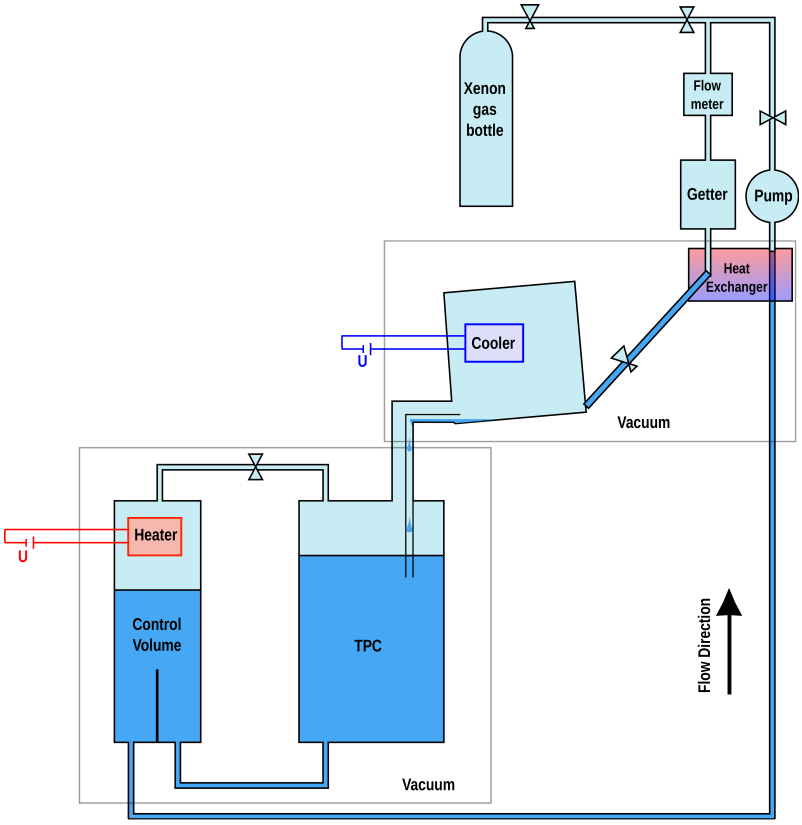

Figure 2.5: The planned Mainz Xenon TPC gas circulation system (modified, original from [Bes12]).

3 The Slow Control System

As the Mainz Xenon TPC needs to reliably conduct measurement runs for long periods (weeks or months), certain parameters need to be constantly monitored to ensure the stability of the TPC. Typical such parameters are temperatures, pressures and voltages to name only a few possibilities. Such a system is known as a *Slow Control System* (SCS). The following features were deemed critical when the system's design was first comissioned:

- Reliable local data logging for all control parameters,
- Slow control of heaters and pumps to maintain a correct xenon level in the TPC and prevent overheating,
- Flexible multi-level alarm system (e-mails, SMS)
- Remote viewing of data

To accomplish these goals, a distributed SCS concept shown in Figure 3.1 was developed as the main objective of this thesis. The Mainz TPC SCS is comprised of several devices and uses online services such as e-mail and short text messages to alert researchers when anomalies are detected in parameters.

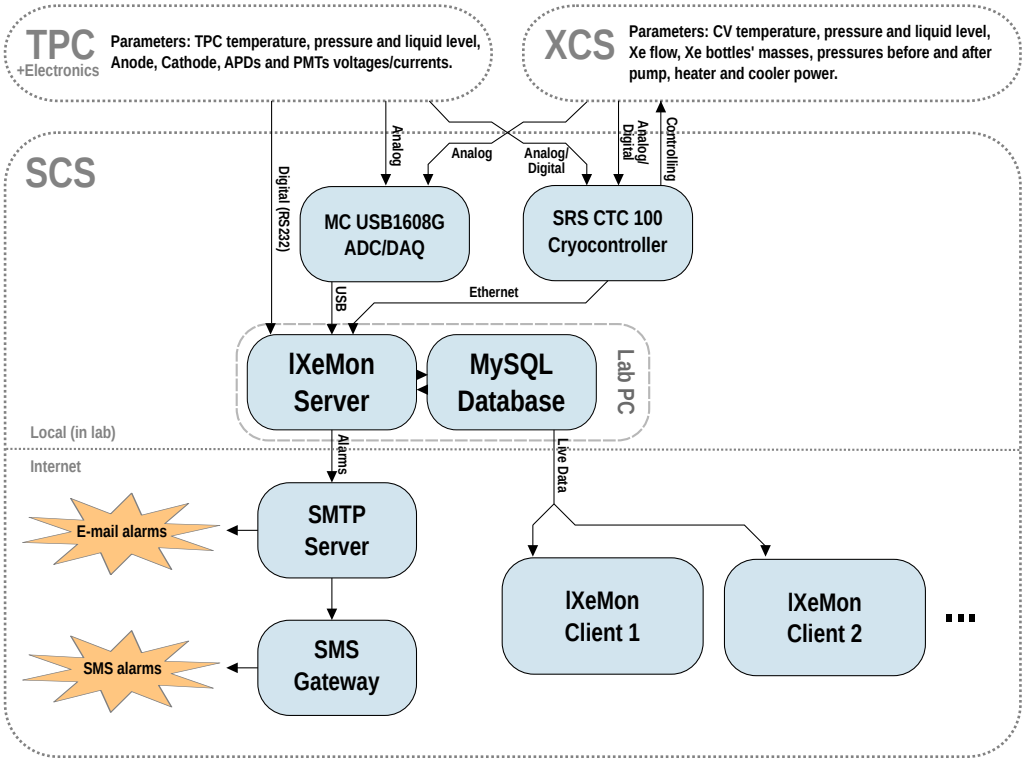

Figure 3.1: The distributed Slow Control System for the Mainz TPC.

A list of the parameters planned for monitoring can be found in Table 3.1. A *SRS CTC100 Cryocontroller* will be used to monitor and control the xenon's temperature while ensuring

a constant fill level at the same time. Many analog sensors will be monitored using a 16-bit ADC *MC USB-1608G Series* USB data aqcuisition device.

Parameter	Description	Unit	Controlled	Device/Interface
T_{tpc}	Temperature in TPC	K	Yes	SRS CTC 100
T_{cv}	Temperature in Control Vol.	K	Yes	SRS CTC 100
T_c	Temperature in cooler	K	Yes	SRS CTC 100
h_{tpc}	LXe level in TPC	mm	Indirectly	SRS CTC 100
h_{cv}	LXe level in Control Vol.	mm	Indirectly	SRS CTC 100
L_h	Power of heater	W	Yes	SRS CTC 100
L_c	Power of cooler	W	No	MC USB 1608G
$\dot{m}_{fl}$	Xenon flow	kg/s	No	MC USB 1608G
m_{fl}	Xenon mass (integrated flow)	kg	No	lXeMon
p_{tpc}	Pressure in TPC	mBar	No	MC USB 1608G
p_{bp}	Pressure before pump	mbar	No	MC USB 1608G
p_{ap}	Pressure after pump	mbar	No	MC USB 1608G
m_{bi}	Xe Bottle mass (x2)	kg	No	MC USB 1608G
V_{cat}	Cathode voltage	V	No	RS232 (Serial)
I_{cat}	Cathode current	A	No	RS232 (Serial)
V_{an}	Anode voltage	V	No	RS232 (Serial)
I_{an}	Anode current	A	No	RS232 (Serial)
V_{APDi}	APD voltage (x8)	V	No	RS232 (Serial)
I_{APDi}	APD current (x8)	A	No	RS232 (Serial)
V_{PMTi}	PMT voltage (x2)	V	No	RS232 (Serial)
I_{PMTi}	PMT current (x2)	A	No	RS232 (Serial)

Table 3.1: Some of the parameters that are planned to be monitored and controlled by the Slow Control System. Where multiple identical parameters are concerned, i stands for the enumeration of the parameter.

To gather and log the data from the different devices, determine when alarm conditions occur and to send out alers in such events, a specialized piece of software was required and consequently developed. The *lXeMon Server* fulfills these roles while the *lXeMon Client* allows for remote viewing of the data, as well as the setting of separate client alarms.

Combined, these components form the Slow Control System for the Mainz TPC, enabling easy monitoring of data both locally and remotely and controlling of the relevant temperatures through heaters.

4 The lXeMon Software

Specifically developed as part of this thesis, the lXeMon software package is intended to be a stable, reliable and flexible software package for use as the software part of the Slow Control System described in chapter 3.

Figure 4.1: The lXeMon logo

lXeMon consists of two parts: the *lXeMon Server* responsible for communication with the different measurement devices, data logging and the alarm system; and the lXeMon Client which provides researchers with easy online access to live monitoring data, independent of platform or device. The general layout of the lXeMon software can be seen in Figure 4.2

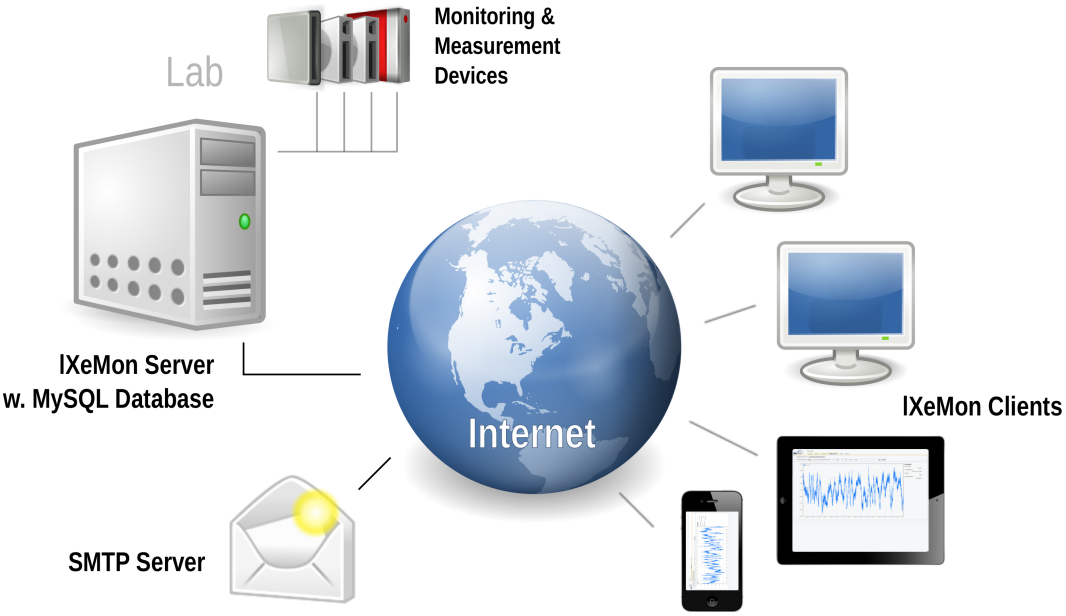

Figure 4.2: An overview of the lXeMon Software.

The lXeMon software package was developed on a *Dell Optiplex 990* personal computer with the following main specifications (for reference):

- Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz (4 cores)
- 4GB of RAM
- Debian Linux 6.0.4 (amd64, with Linux kernel v2.8)

It should also be noted that for consistency reasons, all times and dates used in the lXeMon software package are given in the UTC timezone (GMT+00, no daylight saving adjustments).

4.1 The lXeMon Server

The central piece of software to the SCS system is the *lXeMon Server* (lXMS). The lXeMon Server has three primary roles:

- Handle communication with the different measurement devices (including initialization, calibration and data aqcuisition) and other data sources for gathering data
- Process the data and log the experiment's parameters
- Alert researchers when/if a parameter exhibits abnormal behavior

The C++ programming language was chosen to best accomplish the goals stated in chapter 3, as it is well established, stable, well supported on a variety of platforms, very efficient and incorporates *object oriented programming* (OOP) methodoligy, allowing for future extensions and easier maintainability.

4.1.1 Usage

In this section, some instructions on the correct usage of the lXeMon Server software are given.

Configuration and Start-up

It is very important to be able to reconfigure the lXMS to work with different experimental setups, change the list of people that receive alarms (or which alarms they receive) or change conversion constants for data processing with minimal effort. For this reason, lXMS uses a XML configuration file at start-up.

The configuration file is a very important and powerful aspect of this software and should be carefully set before starting an experiment. In it, all parameters of the experiment must be defined (e.g. their names, data sources, conversion methods, units and descriptions) as well as all data sources (e.g. external devices), alarm conditions, contacts for alerting and database and e-mail configurations. An example XML configuration file with explanations is attached in appendix A.

lXMS is a console application with no graphical interface and as such would normally be executed in a terminal window using a command such as (in Linux):

```
1 ./path/to/lxemon-server -c config.xml
```

The "`-c <configuration file>`" flag is required. For debugging, the "`-d`" flag can be useful, as it makes lXMS give a much more verbose output to the *stderr*[1] stream. For letting lXeMonServer run in the background a "&" symbol at the end of the command will work.

Operation

During runtime, the user can interact with the lXeMon Server through the terminal. A list of commands and what they do is found in Table 4.1.

[1] stderr is the standard error output of a system and by default (in Linux and Windows) is the same as stdout, that is, the current terminal. A useful feature in Linux is using "`./lxemon-server -d -c config.xml 2> /dev/pts/X`" where X is the number of a another terminal window, redirecting the stderr output to that window.

Command	Effect
`help`	Lists all available commands
`log <message>`	Enters a short message to the experiment log
`period <length>`	Changes the period length (length in ms)
`listparameters`	Lists all parameters and whether they are controllable
`stop <parameter>`	Stops a parameter's controllable converters
`start <parameter>`	Starts a parameter's controllable converters
`reset <parameter>`	Resets a parameter's controllable converters
`listalarms`	Lists all alarms and their arming status
`arm <alarm>`	Arms or rearms an alarm
`disarm <alarm>`	Disarms an alarm
`quit`	Quits the lXeMon server

Table 4.1: A list of available commands in lXeMon Server

4.1.2 Periodicity

lXMS works periodically in real-time: in each period it executes a series of tasks which must finish before the next period. The duration of a period can be defined in the XML configuration file (see appendix A). When this project was started, while a default period of one second was the intended period length, lXMS has been tested successfully with periods of 500 ms, 300 ms, 200 ms and even lower, but it must be noted that the minimum period length mostly depends on both the types and amount of devices used for gathering data as well as the quality of the device implementation and the computer's specifications. The period length can also be set during runtime (see Table 4.1). If some tasks fail to complete before the next period is scheduled to start, the upcoming period will be skipped entirely.

The tasks lXMS performs during each period are (in the order of execution):

1. Send requests to all devices to obtain the current raw measured values.
2. Process the raw data into parameters (i.e. convert an analog mass sensor's voltage to a mass dimension such as kilogram).
3. Wait until 90% of the period has passed.
4. Run all defined parameter alarm checks. Send out alerts if alarms are triggered.
5. Insert the data set into the database.
6. Prepare a clean set of data with a real-time timestamp and wait for the next period to start.

There are some real challenges when programming real-time critical applications on non-dedicated hardware. Extensive use of multi-threaded[2] programming was necessary to enable the required fast parallel communication with multiple external devices via different interfaces (USB, ethernet, serial, etc.). As such, a multi-core processor is preferred for any platform running lXeMon as it facilitates the desired simultaneous[3] execution of multiple threads in parallel.

[2] Multi-threading refers to the technology of having multiple threads run in a single process concurrently. All threads share the same resources (memory) and as such this technology requires extra precaution with memory management to avoid segmentation faults and/or unexpected results. An alternative to multi-threading is *forking* a process, where the process gets duplicated (including its allocated memory resources), allowing for a simpler handling of the memory. Forking was not a viable approach in the case of lXMS as it would've required inter-process communication, which is both computationally intensive and more complicated.

[3] It should be noted that actual simultaneous execution of the measurements is not feasible on a commercial

4.1.3 Inner Structure

A class diagram of the lXeMon Server software is shown in Figure 4.3. In this section, a short description of the most important parts of the software, and how they work together is given.

The `lXeMonCore` Class

The `lXeMonCore` is the main hub of the application, binding the different parts together. It handles all initialization procedures when the program is run, including the initialization of the `lXeMonClock`, `-DB`, `-Alerter`, `-DataManager` and `-DeviceManager` objects and parsing of the XML configuration file, forwarding the configuration parameters to the correct objects (e.g. `Device` configurations to the `DeviceManager`).

When this *initialization phase* is finished and the *monitoring phase* begins, the core instructs the `DataManager` and the `DeviceManager` to start their periodic functionalities (see the respective class descriptions below) and assumes the role of user interaction, still running in the program's main thread.

The `lXeMonDB` Class

`lXeMonDB` serves as the connection to the local MySQL database (see section 4.1.4), both taking care of connecting to and preparing the database by creating the necessary database tables for each experiment run during initialization. During the monitoring phase, the `lXeMonDB` provides the `lXeMonDataManager` with helper functions to directly insert a `lXeMonDataSet` into the database and the core and `lXeMonException` classes with a helper function to log user messages, system messages and errors.

It is important to note that due to limitations of the underlying database, a single database connection cannot be safely shared amongst different threads. For this reason, the `lXeMonDB` object needs to be deep-copied for each thread that needs to contact the database (e.g. in the case of an error). Under normal circumstances, lXMS has two database connections alive; one for the main thread (messages during initialization and user notes during monitoring) and one for the publishing thread in which `lXeMonDataManager` operates.

The `lXeMonDataManager` Class

The `DataManager` handles the periodic *publishing* of data to the database. During initialization, the `DataManager` receives instructions from the core regarding what `Parameter` objects it should initialize, and on which *sources* they depend. It then forwards configuration information on Alarms and Converters to each `Parameter` (see below for more information on `Parameters`).

When the core has finished the initalization phase, it will tell the `DataManager` to start publishing data. At this point, the `DataManager` creates an empty `lXeMonDataSet` (with a timestamp) and launches a new thread which is scheduled to run the *publishData()* method when $\frac{9}{10}$s of a period has passed. *publishData()* will then, when run, check all the `Parameters`' current values, insert them into the current `lXeMonDataSet` and send it to the `lXeMonDB` for insertion into the database before scheduling a recurrent run of itself in another period's time. The reason for not running *publishData()* immediately is to wait for the `DeviceWorker` threads to deliver data before attempting to check and publish it.

PC limited by only one front side bus; even if multiple CPU cores enable simultaneous calculations, they do not provide actual simultaneous communication with external devices as they share the front side bus and other hardware components inside the PC.

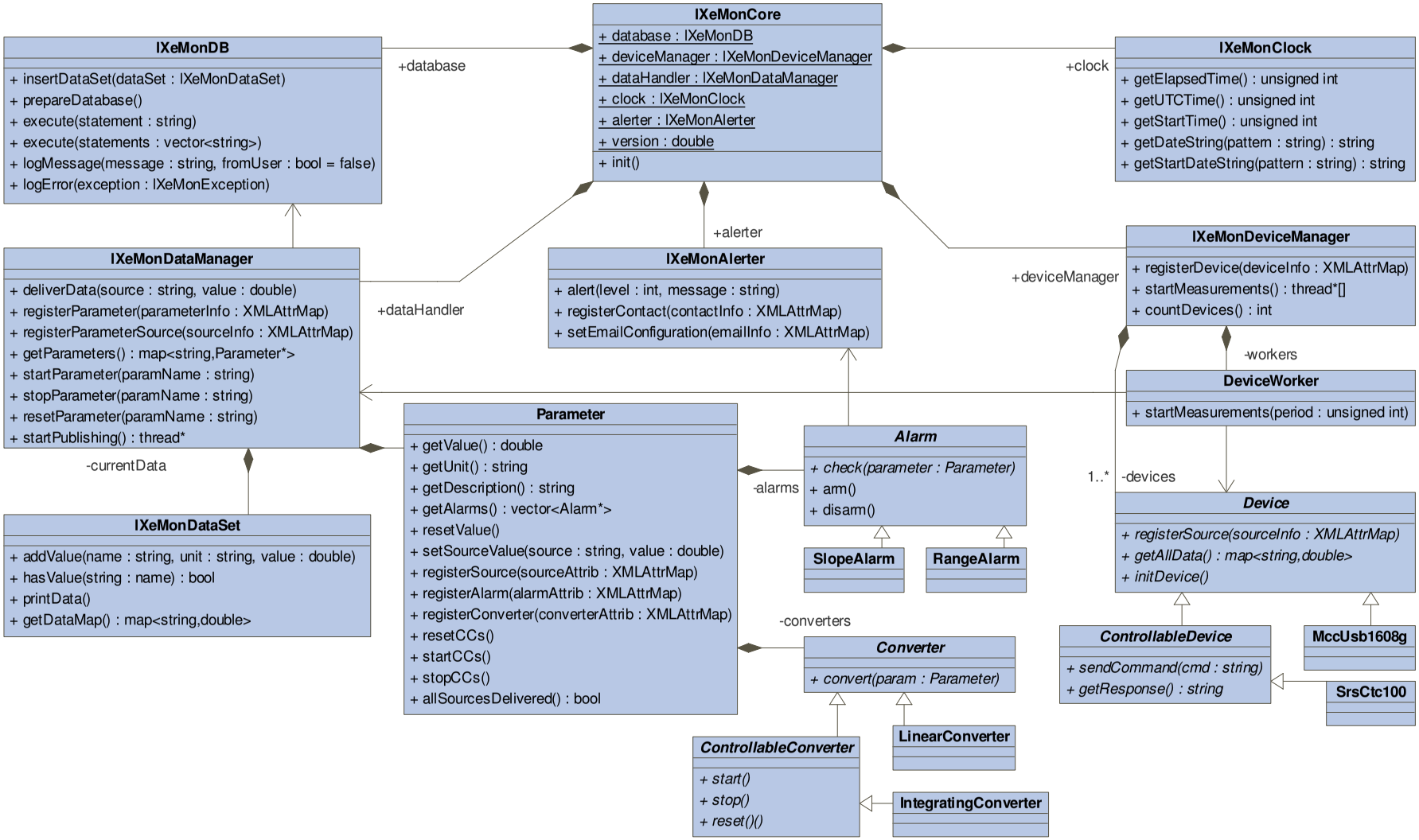

Figure 4.3: A UML2 Class Diagram for the lXeMon Server software. Only public members of each class are listed; for more detailed information consult appendix B.

The `lXeMonDeviceManager` and `DeviceWorker` Classes

The `DeviceManager` both acts as a `Device` factory and as the boss in a *boss-worker design pattern*[4], with one `DeviceWorker` for each `Device` as its worker object. Much like the `DataManager`, the `DeviceManager` receives instructions from the core regarding what `Device`s it should initialize and prepare.

After the core has instructed the `DataManager` to start publishing, it will immediately instruct the `DeviceManager` to start measurements. The `DeviceManager` will then tell each of its workers to spawn a new thread and in it start periodically querying the attached `Device` object for data. Each `DeviceWorker` will then periodically deliver the data received from the `Device` to the `DataManager`, who takes care of forwarding it to the `Parameter`s that depend on it.

The `Device` Class

The `Device` class is an *abstract class*[5] from which different device driver implementations must derive, and override only two methods, *getAllData()* returning an associative array of *source* names and values and *initDevice()* for initializing (and in the case of an error, should allow reinitializing) the device.

While the phrase "device" is used here for clarity, an implementation of the `Device` class need not neccesarily be a driver for a local physical device - it can be anything that provides one or more sources of data, for example a simulations program, a connection to a remote computer via the internet, or even to a satellite.

For the Mainz TPC two implementations of `Device` were prepared, listed in table 4.2

Implementation	Physical device	Interface	Sources provided
`MccUsb1608G`	MC USB 1608G 16-channel Data Acquisition device with a 16bit ADC	USB	8 differential or 16 single-ended channels, labeled 0 to 15
`SrsEthDevice`	SRS CTC100 Cryocontroller	Ethernet	4 temperature INs (In 1 - In 4), 2 heater OUTs (Out 1 - Out 2), 4 analog I/O (AIO 1 - AIO 4), 8 digital I/O (DIO 1 - DIO 8), 4 relays (Relay 1 - Relay 4)

Table 4.2: `Device` implementations in lXeMon Server v1.0 (for the Mainz TPC).

The `lXeMonClock` Class and Timekeeping Issues

The `lXeMonClock` is lXMS's timekeeper. It uses Linux's *monotonic* clock to keep track of real time, as the monotonic clock cannot be set (i.e. by the user or other programs, such as Linux's *Networt Time Protocol* (NTP)[6]), and provides a nanosecond resolution on modern systems. Using the monotonic clock bypasses the potential problem of NTP or the user changing the clock, but another feature of NTP is changing the clock *slew*, by which it

[4] The boss-worker design pattern is an implementation of thread-based applications where a boss object spawns multiple worker threads, each solving a task and returning the information to the boss who then takes care of it. For more information on different *design patterns*, consult the reference [SKG11].

[5] An abstract class is a class that provides only overridable (most often empty) methods, serving as a sort of "recipe" for a class, allowing the compiler to know roughly what to expect from a pointer pointing to it. The `Device` class actually does define one method itself, *getName()*, as it is required and independent of implementation. Additionally, it must called with a non-default *Device(string name)* constructor upon deriving.

[6] NTP runs by default on most Linux systems and adjusts the time using time provided by a remote server. Obviously, this is not a desired feature for a physics experiment since a clock jumping a few seconds back or forth can have serious consequences.

slows down or speeds up the system clock by a small amount for a while to allow it to catch up with the correct time. The only way to avoid this potentially undesired effect would be to use the *monotonic_raw* clock, which gives a purely hardware-driven clock unaffected by the system, but is only available in the linux kernel version 2.6.28 and higher combined with a glibc version 2.12-111 or higher[7]. The system used for development only had glibc version 2.11.3-2 and so did not support the monotonic_raw clock, but it should be considered for use in future versions of the lXeMon Server.

The monotonic clock is a real-time clock in the sense that it measures clock in real-time speed, but unlike Linux's actual *realtime* clock, it does not have a well-defined starting point. Hence, `lXeMonClock` asks the system for both the current UTC date and time (via gmtime(), which is much less precise than the monotonic clock, slower and susceptible to system clock changes, but always gives UTC time unlike the *realtime* clock) and the monotonic clock's time, using gmtime's information to correctly calibrate the monotonic clock. All subsequent calls to the *getElapsedTime()* will then return a nanosecond-resolution timestamp which has its zero at the classic *Unix epoch* of UTC 00:00 on January 1st, 1970.

The `Parameter`, `Converter` and `ControllableConverter` Classes

A `Parameter` represents a value that is to be monitored by the SCS software. During initialization it takes care of manufacturing its own Alarm and `Converter` objects, for which the configuration is provided by the core through the `DataManager`.

To ensure maximum flexibility, lXMS employs a system where each of its `Parameters`' *value* is composed of one or more *raw values* from *sources* (normally a Device provides one or more sources, and each `Parameter` acts as a source for other `Parameters` by default), and each source's raw value is first sent through any attached Converters. The `DataManager` takes care of delivering raw values to the `Parameter`, and when it has received all of its raw values, they are either multiplied or added together to form the final value, which then gets checked for alarm conditions, submitted as a source for other `Parameters` and, at the end of the period, inserted into the database. Each `Parameter` must have a unique name and (optionally) a unit, which for example determine the name of the database columns (see section 4.1.4) and so should not include special characters such as German umlauts (common symbols such as `/ - + _` and brackets etc. are allowed).

The `Converter` is a virtual class, whose implementations only need to define a *convert()* method to convert the value in some way. A list of implementations prepared for the Mainz TPC and their effects are listed in Table 4.3.

Implementation	Configurable Parameters	Effect	Controllable
`LinearConverter`	`slope, offset`	$\texttt{out} = \texttt{in} \cdot \texttt{slope} + \texttt{offset}$	No
`IntegratingConverter`	`offset`	continuous trapezoidal integration	Yes

Table 4.3: A list of `Converter` implementations in lXeMon Server v1.0

A requirement that was realized early on in the SCS's design process was that some continuous parameters (e.g. the integrated total mass that has gone through a flow meter) should be stoppable, resetable and restartable by a user. This was achieved by introducing the `ControllableConverter` abstract class, which derives from the `Converter` class but adds three new methods; *start()*, *stop()* and *reset()*. The `IntegratingConverter` is an example of one such class, where the area under the attached `Parameter`'s curve is integrated in real-time. If a user requests a `Parameter` to stop, the `Parameter` will stop all of its `Controllable-`

[7] Source for linux and glibc versions required: http://stackoverflow.com/a/8736472 - StackOverflow answer by jøgensen

Converters - using our total mass Parameter example, a user could stop integrating the curve at any time using the stop <parameter> command and resume it later with start <parameter>, or use reset <parameter> to set its value to zero. While a Parameter is stopped, it delivers the value it stopped at in each period

The lXeMonAlerter and Alarm Classes

The lXeMonAlerter and Alarm classes comprise the main parts of the alarm system (for more informaton on the alarm system, refer to section 4.1.5).

Alarm is a virtual class from which different implementations of alarms need to derive from. Alarms get attached to a Parameter and each Alarm implementation needs to include a method for checking whether its attached Parameter's value meets its alarm condition, and call lXeMonAlerter's *alert()* method if that is the case, which then forwards the alert to the lXeMonAlerter for processing. Available Alarm implementations in lXMS v1.0 and their required configuration parameters are listed in Table 4.8. In the initialization phase, the Parameter objects receive information from the core on what Alarms they should create and attach to themselves.

Implementation	Configurable parameters	Explanation
RangeAlarm	tolerance, min-value, max-value	Checks if a Parameter's value is within the allowed range [min-value, max-value]
SlopeAlarm	tolerance, min-value, max-value, avg-over	Checks if the Parameter's fitted slope over the last avg-over values is within the allowed range [min-value, max-value]

Table 4.4: The Alarm implementations available in lXeMon Server v1.0

During initialization, the core forwards information about contacts to the lXeMonAlerter, retrieved from the provided configuration file, including their e-mail addresses, mobile phone numbers, and what level alarms they should receive. lXeMonAlerter lists these contacts and notes which contact gets what level alarms and by which means (e-mail or SMS). The core also parses and forwards configuration for the SMTP[8] server to the lXeMonAlerter.

During the monitoring phase, when an Alarm object raises an alarm, the lXeMonAlerter processes it, first asking the lXeMonDB to log the alarm to the experiment's log and then determining if further actions should be taken for the particular alarm level, depending on the configured contacts. If a contact is found that should be notified of an alarm of the level in question (either by e-mail or by SMS), further actions are taken: a separate thread is started (as to not block the current thread too long, as the publishing thread is executing these instructions and could miss the next period otherwise) which connects to the configured SMTP server (using the *libESMTP* library), and sends the relevant e-mails. For SMS messages, e-mails are sent to a commercial online gateway[9], which then forwards them as SMS messages to the mobile phone network.

4.1.4 The Database

For storing the monitored data, as well as logs and information about measurement runs, lXMS uses a local MySQL database. By using a database instead of log files, for example,

[8] *Simple Mail Transfer Protocol* - The most commonly used protocol for sending e-mail.

[9] For the Mainz Xenon TPC, Clickatell.com will be used as the SMS gateway and as such is the preconfigured gateway in lXMS. For other gateways, the lXMS source would need to be edited and recompiled, as providing this configuration via XML is complicated and not deemed necessary and so exceeds the scope of this thesis.

we avoid a plethora of problems that may arise when multiple lXeMon Clients simultaneously need to read data while the lXeMon Server is still writing it, while also ensuring data integrity, optimal storage efficiency, usability[10] and speed.

The database also serves as the main link with the lXeMon Client (discussed in chapter 4.2), which plots data queried from the database in real-time.

Database Access

For security reasons it is recommended that a database is created and dedicated to the lXeMon Server (note that a MySQL database can have many sub-databases) along with a dedicated user having only `INSERT` and `CREATE` rights exclusively for that database. As the username and password must be provided in clear text in the configuration file, which is readable to anyone that gains access to the computer, more destructive rights such as `DELETE`, `UPDATE`, `DROP`, `ALTER`, etc. should not be given to this user (lXMS has no use of them anyway) in case of an unauthorized person gaining access to the configuration file.

Database Structure

At initialization, lXMS creates three new tables in the database named `run_YYYYmmdd_HHMMSS_data`, `run_YYYYMMDD_HHMMSS_log` and `run_YYYYMMDD_HHMMSS_params`, where Y (year), m (month), d (day), H (hour), M (minute) and S (second) are replaced with their current date and time counterparts at launch. A separate table called `_runs_info` contains a list of all the runs and their starting times. The structure of the `log` table is the same for every run and is shown in Table 4.5.

Column	Type	Attributes	Contents
`ID`	`bigint(20)`	`AI`[11], `unsigned`	Entry ID
`time`	`bigint(20)`	`unsigned`	Timestamp (in ns since unix epoch)
`type`	`varchar(64)`	`utf8`	Type[12]of log message
`message`	`varchar(512)`	`utf8`	Message
`info`	`varchar(1024)`	`utf8`	Misc. info (not in use so far)

Table 4.5: The column structure of the log table.

When parsing the configuration file provided by the user, lXMS automatically fills the `params` table with information about the parameters in the experiment run. This data is used by the client to facilitate clear presentation of data to users. The structure of the `params` is also consistent for every run and can be seen in Table 4.6

Column	Type	Attributes	Contents
`ID`	`bigint(20)`	`AI, unsigned`	Entry ID
`name`	`varchar(16)`	`utf8`	Name of the parameter (e.g. `T_tpc`)
`unit`	`varchar(16)`	`utf8`	Unit of the parameter (e.g. K)
`description`	`varchar(512)`	`utf8`	A short description of the parameter

Table 4.6: The column structure of the log table.

[10] For easy data exporting (in ASCII, Excel, XML or many other formats), viewing and manipulating, phpMyAdmin (http://www.phpmyadmin.net/) is a helpful tool that can run on the web server alongside the lXeMon Client and be accessed both remotely and locally at any time.

[11] AI denotes Auto Increment, a feature of MySQL which assigns automatically incrementing integer values to the field.

[12] Some currently defined log message types are USR_MSG (note from user), SYS_MSG (message from system), SYS_ERR (error message from system)

The structure of the data table depends on the parameters defined for the experiment run. Only two columns are the same for every run; `ID` and `time`. For each parameter in the experiment, a column with the name `parameterName[parameterUnit]` is added. An example of a experiment run's data table is shown in Table 4.7.

Column	Type	Attributes	Contents
`ID`	`bigint(20)`	`AI, unsigned`	Entry ID
`time`	`bigint(20)`	`unsigned`	Timestamp (in ns since unix epoch)
`param_1[kg]`	`double`	-	
`param_2[mBar]`	`double`	-	
`param_3[K]`	`double`	-	
...			

Table 4.7: The column structure of the data table

4.1.5 The Alarm System

One of the main requirements of the Mainz TPC SCS was a flexible, reliable and powerful alarm system. To meet these requirements, lXMS implements a per-`Parameter` multi-level alarm system capable of setting up almost unlimited[13] alarm conditions for each `Parameter` independently of each other, and with the use of combined `Parameter`s (see section 4.1.3), alarms that span many parameters. As a means to alert researchers of alarms that occur, lXMS supports sending both e-mail and SMS messages.

`Alarm`s are configured on a per-`Parameter` basis in the configuration file. Each `Parameter` can have many different alarms, each of which has configurable tolerance and level values. When the alarm's condition is met (e.g. a `Parameter` with a `RangeAlarm` attached is above the `RangeAlarm`'s maximum allowed value), a level 1 alarm is triggered once. If the `Alarm` in question has a non-zero tolerance, it will not trigger its designated alarm level until a number of consecutive values exceeding the tolerance set meet the alarm condition, at which point the designated alarm level is raised once. A tolerance of 0 means that an alarm of the designated level is raised immediately at the first breach, instead of the default level 1 alarm. There is also no limit on the levels of alarms that can be defined, allowing for example person-specific alarms for certain situations (e.g. by using alarms of levels 100+).

Level	Severity	Cause	Actions
1	Light	Any data anomaly	Log
2	Medium	Consistent data anomaly	Log, e-mail to person on shift
3	Serious	Consistent serious data anomaly	Log, e-mail to everyone, SMS to person on shift
4	Critical	Consistent critical data anomaly	Log, e-mail and SMS to everyone

Table 4.8: An example for a typical alarm scenario. lXMS allows for customizing this scenarion in almost any way possible.

This system allows for very flexible configurations. For example, Table 4.8 shows a typical alarm scenario where level 1 alarms only get logged while level 2 alarms cause an

[13] While there is no hard limit on the amount of alarms, the period length and the system's hardware need to be taken into consideration. A very large amount of alarms may take too much time for the computer to process (not to mention if they all get triggered), meaning the publishing thread may be busy and miss the next measurement period.

e-mail to be sent to certain people on shift. Level 3 alarms will cause everyone affiliated to receive an e-mail and the person on shift to receive a SMS also, while level 4 alarms are reserved for certain critical parameters exceeding critical values, sending a SMS to everyone involved with the experiment.

Using the Mainz TPC as a second real-life example, a temperature sensor for liquid Xenon kept at 162.5 K could have a level 2 alarm trigger when it exceeds a rather high value of 163.0 K. If the temperature would then keep rising and exceed 163.5 K, a second level 4 alarm could be triggered, reaching a wider audience and with more urgency. Independently the *SRS CTC100 Cryocontroller* should turn on the emergency cooling coil at this stage.

4.2 The lXeMon Client

The lXeMon Client (*lXMC*) was also developed as part of this thesis alongside the lXeMon Server and enables researchers to remotely monitor and view data collected by the lXeMon Server in real-time. As previously stated, the MySQL database serves as the only link between lXeMon Client and Server.

The client is implemented using a combination of modern web technologies (PHP, HTML5, JavaScript and AJAX) as a web application and runs entirely within a browser, needing no third party plugins. This allows lXMC to support a wide variety of platforms that can run a modern browser, including mobile iOS (iPad and iPhone) and Android (most other tablets and smartphones) devices. This feature is the main motivation behind using browser technologies rather than a standalone client (such as a Java program), as well as the convenience gained from not having to install any software at all on a client device.

4.2.1 Installation and Configuration

The server-side part of the lXeMon Client is developed in PHP 5, which is a powerful and widely supported server-side scripting language for web applications. As such, lXMC can be hosted on most typical HTTP servers (e.g. Apache with PHP 5 module) and does not neccesarily need to be installed on the same machine as lXeMon Server[14], but to ensure minimum latency between the client and the database, it is preferred and also more convenient.

lXMC's configuration takes place in a file called `lXeMonConfig.php` in the root directory. It is important to configure the correct database login information, as well as some environment information (root path, web server path).

Additionally, groups of parameters can be defined in the configuration file, as well as a list of parameters to be hidden from view, helping to keep the interface tidy and free of clutter. Using the grouping function, related parameters (e.g. all temperatures in Mainz TPC's xenon circulation system) can be grouped into groups of 4 (or less), allowing the user to simultaneously view graphs of these parameters. If lXMC detects parameters that are neither assigned to a group nor hidden, it will create its own groups named MiscParameters-XX where XX stands for an iterating integer number.

[14] Please note that it is recommended that the MySQL server be hosted on the same machine as the lXeMon Server, and by default does *not* accept connections from other machines. MySQL needs to be configured to accept remote connections (preferrably only from the Client's host) on the lXMS host machine if the lXeMon Client is to be hosted on a separate server.

4.2.2 Usage

To start using lXeMon Client to remotely view data, a user only needs to have a modern HTML5- and standards-compliant browser[15] and type in the IP or hostname of the lXMC host machine in the address bar[16].

For security reasons, the user will need to log in when he opens the client. Once logged in, a user stays logged in until the end of the session (e.g. closing the browser), or until he logs out. A timeout feature would not make sense here, as a user monitoring parameters can easily seem idle to the client for many hours.

A tab-style navigation can be found at the top of the client interface (see Figures 4.4, 4.5 and 4.6) where the Client's standard tabs are in yellow and the user-defined parameter group tabs are in white. Below it a user will find lXMC's control elements: drop-down selection boxes which allow easy selection of the experiment run (current or previous) the user would like to monitor, and where applicable the time period of the live data and the update rate.

The *Overview* Tab

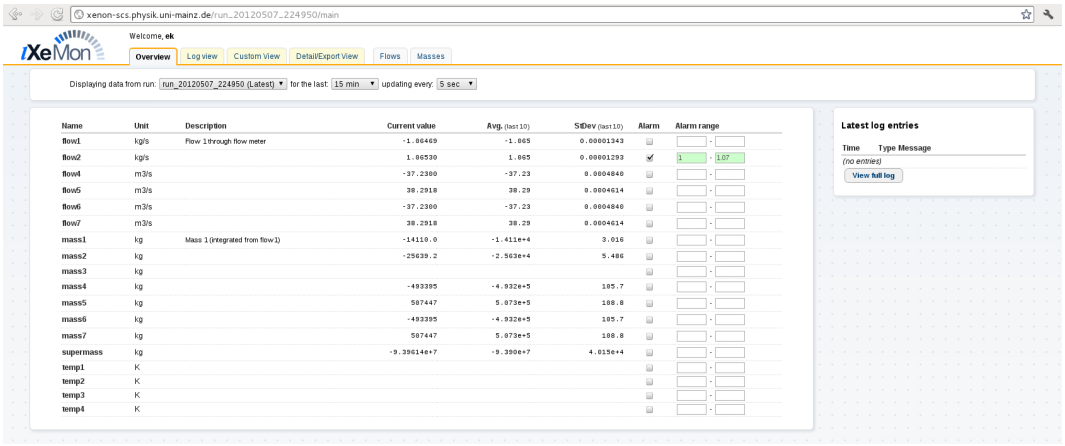

Figure 4.4: The lXeMon Client's Overview tab, presenting all the experiment's parameters at a glance and allowing the setting of client alarms.

On the overview tab shown in Figure 4.4, a user gets presented with real-time values from all parameters in a tabular form, as well as their standard deviations and average over the last 10 values. Additionally, a user can set *client alarms* for each parameter (see the next section) and view the most recent log entries from the experiment in the column on the right.

Client Alarms

On the overview tab, a user can set separate *client alarms* for each parameter. Currently only a *range alarm* is supported, i.e. an alarm that goes off when a parameter goes out of a defined range.

To set a client alarm, the user simply needs to enter a minimum and maximum value for the parameter and activate it by checking the alarm checkbox. Once an alarm is activated, the client will remember its existence, even if the view is changed, by saving a cookie on the user's computer. To deactivate an alarm, the user must simply uncheck the alarm's checkbox.

When an alarm triggers, the client will play a beeping sound[17] and the parameter in ques-

[15] This includes all recent versions of Firefox, Opera, Chrome, Safari and also IE9 (support for IE8 and below is not a realistic goal due to its lack of HTML5 support)

[16] The lXMC client for the Mainz TPC can be found at http://xenon-scs.physik.uni-mainz.de (login: guest/guest).

[17] Please note that not all browsers support this functionality (HTML5's native <audio> tag, serving a choice of

tion will receive a reddish background, in order to catch the attention of the user so that appropriate actions can be taken.

The *Log View* Tab

The log view enables the user to view the experiment's log, containing messages both from the lXeMon Server software (such as warnings, alarms or errors) and from the lXeMon Server's user (see section 4.1.1).

The *Custom View* and *Parameter Group* Tabs

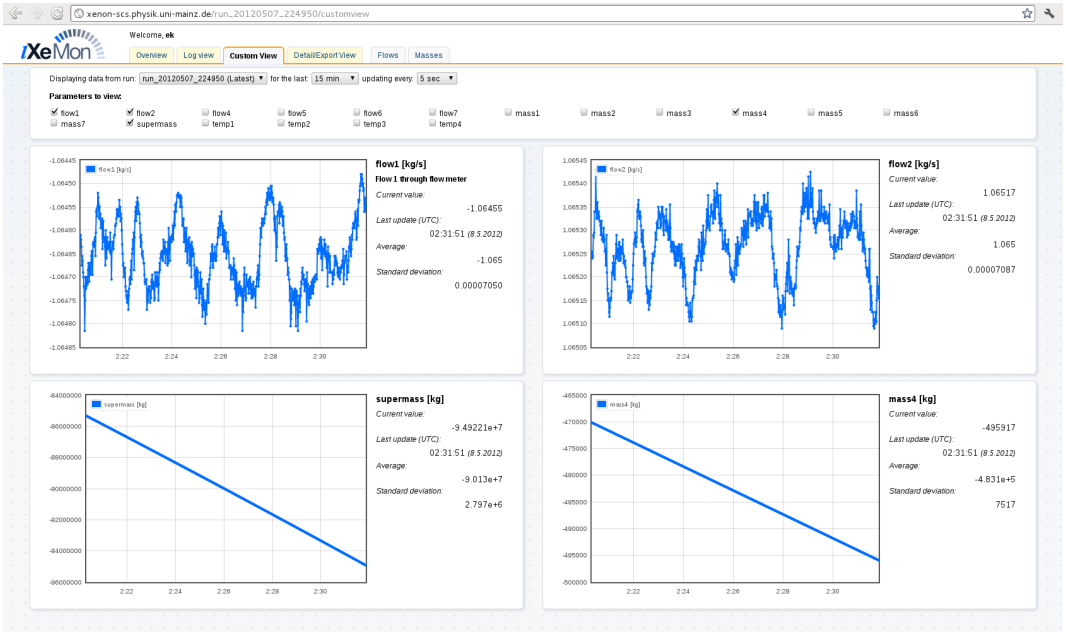

Figure 4.5: The lXeMon Client's *Custom View* tab, allowing the user to choose the parameters he would like to see plotted in real-time.

The custom view tab shown in Figure 4.5 supports showing up to eight freely selectable parameters simultaneously, dynamically addable and removable by the user and arranged on a two by four grid and with real-time updating. Each parameter is both plotted over the selected time period and its textual value shown, along with its average value and standard deviation calculated over the selected time period.

The parameter group tabs only differ from the custom view tab in that they contain predefined groups of four parameters each, arranged on a two by two grid. This allows for some common setups of parameter groups to be predefined in the configuration file to improve usability, especially on handheld devices where adding the devices in the custom view can sometimes be tedious on small touch-screen.

The *Detailed/Export* View

In the detailed/export view, the user can select a single parameter and a time frame from which the data is shown on a large graph. With the click of a button, the chosen data can also be exported and downloaded in the form of a CSV (*Comma seperated values*) file. The data is not updated in real-time in this view.

---

mp3, ogg and wav sound files in this case) yet, so testing is in order when using client alarms and the sound is important. If a given browser does not play a sound, either try updating the browser to a more recent version or try a different browser.

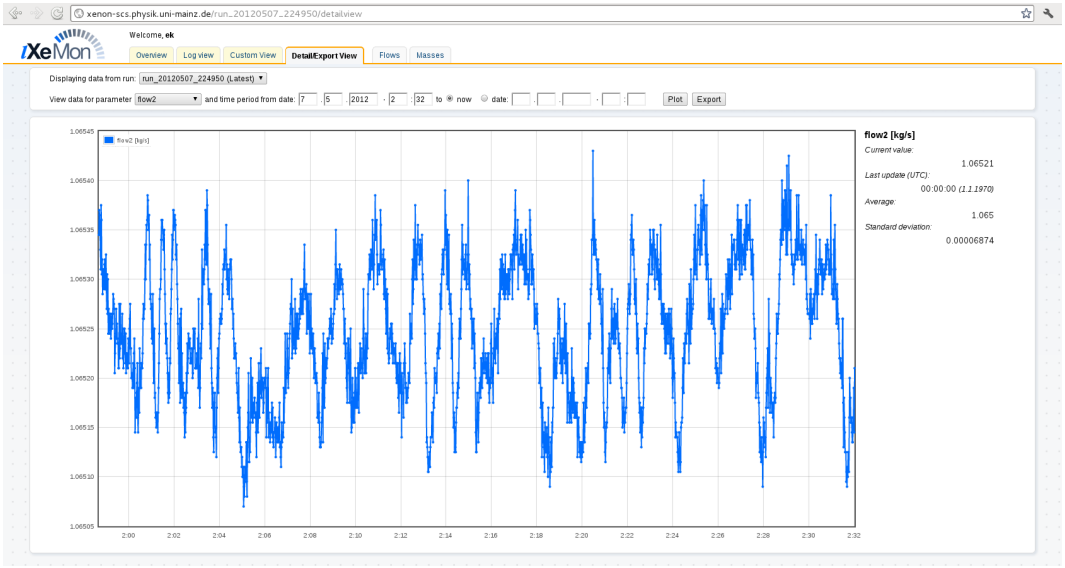

Figure 4.6: The lXeMon Client's *Detailed/Export View* tab, allowing the user to choose a parameter and timeframe to view a detailed plot of data from.

4.2.3 Programming Structure and Implementation Notes

The lXeMon Client code consists of two separate parts. The client's server component is programmed in object-oriented PHP5 which takes care of the login/user system, templates, the database connection and serves the user the HTML5 pages.

Meanwhile, lXMC's client-side programming is done in JavaScript and implemented largely as a plugin for the *jQuery*[18] library, which it uses extensively to ensure compatibility with different browsers. Using AJAX (*Asynchronous JavaSript and XML*) requests to communicate with the PHP server component in real-time and receive the experimental data. *Flot*[19] was used as the plotting library of choice, on the merits of being both lightweight and flexible.

4.3 Portability to Windows or Other Systems

As stated at the start of this chapter, the lXeMon Server software was developed on a Debian Linux platform and compiled with the GNU g++ compiler. At the time of this writing, compiling on other platforms has not been attempted, but for the sake of completeness some areas that could or would pose portability issues are briefly discussed here.

The lXeMon Client is implemented in PHP5 and with JavaScript and should be entirely platform-independent. Still, due to limitations of a 32-bit system when it comes to handling big integers (timestamps in this case), lXeMon Client will only work on a 64-bit system[20].

Care was taken to use only standards compliant HTML5/CSS markup and related technologies to ensure forward compatibility with future browsers.

[18] http://www.jquery.com

[19] http://code.google.com/p/flot/

[20] If necessary, lXMC could rather easily be ported to a 32-bit system, where some workarounds for large integers using strings could be used. It was refrained from doing this at the time being though, as it would affect performance.

Multi-threading

The parts of lXeMon Server that incorporate multi-threaded programming use libBoost's `boost::thread` threading library and should therefore be portable to any platform supported by libBoost, which include most unix-like systems and Microsoft Windows.

Timekeeping

The lXeMon Server's `lXeMonClock` class uses Linux's *clock_gettime()*, *time()*, *gmtime()* and *strftime()* system methods, as they offer superior time resolution (nanoseconds) to portable solutions such as the `boost::date_time::posix_time` library or the new C++11[21] `std::chrono` library. To compile lXMS on another platform, these functions would need to be replaced with time functions supported by the platform; alternatively the aforementioned boost library could be used instead.

[21] C++11 is a new version of the C++ standard with some very useful features for this project, but was not used since the g++ compiler version used for this project does not yet fully support it.

5 Conclusion and Outlook

It is quite possible that the modern mystery that is dark matter will be solved in the years ahead, such is the ever-growing interest in the subject. A clashing of particle- and astrophysics, the study of dark matter could possibly open doors to new physics and become a catalyst for a new way of thinking in physics, whether it be supersymmetric or in extra dimensions or some other exotic way.

The Mainz Xenon TPC is scheduled start operating sometime in the next year and the planning stage of the experiment is well underway, as this paper demonstrates. Even a rather small-scale modern physics experiment like the Mainz TPC requires considerable amounts of thorough planning and solid organization to be successful.

While the Mainz TPC is unlikely to unravel the dark matter mystery itself, which is by no means its goal, it could help by providing a missing part of the puzzle and improving our knowledge of low-energy scintillation efficiency, enabling better background discrimination for larger future experiments.

lXeMon's Future Role

It is important to any experiment that critical parameters are monitored, both constantly by an automated system capable of alerting humans and when required, by the researchers themselves. The lXeMon software package accomplishes both of these tasks; the lXeMon Server taking care of automated monitoring and alerting while the lXeMon Client allows unrivalled usability when it comes to monitoring off-site, whether it be from the comfort of one's home office, in an airplane or on a remote beach - thanks to modern technology and the open standards lXeMon Client makes full use of.

At the same time, the lXeMon software was developed and implemented with extendability in mind and could be expanded upon for larger projects or used relatively unchanged for other small-scale experiments. The codebase is purely object-oriented and using the documentation found in appendix B, a programmer 50 years from now could easily reuse the program and improve upon it.

As an example of the flexibility lXeMon offers, with very little effort one could implement a device driver that fetches data from secondary instances of lXeMon running on separate machines, creating a network of lXeMon Servers working together in a hierarchical system to consolidate data but spreading the workload, and using the alarm system to make sure other lXeMon Servers do not go offline unnoticed.

Personal Notes

Working on this project, I got to know new aspects of physics and related subjects I had not been familiar with before, and learned a great deal about modern (but surprisingly accessible) dark matter physics and general scintillation experiments, not to mention a new and very powerful programming language in the form of C++. A lot of the time was spent on programming-related problem solving and debugging, but the end products have emerged stable and functional in the form of lXeMon Server and lXeMon Client and a lot of knowledge was gathered along the way, making the project an overall success.

In addition to the lXeMon software, the concept and foundations for the Mainz TPC's Slow Control System were successfully laid down as a part of this thesis in cooperation with

Professor Oberlack and graduate student Bastian Beskers, to both of whom I would like to extend my sincerest gratitude for their efforts. I would also like to thank the Mainz XENON group as a whole for providing very motivating and enthusiastic working conditions.

Bibliography

[ABC07] E. Aprile, L. Baudis and B. Cabrera, Search for Weakly Interacting Massive Particles with CDMS and XENON, Journal of Physics: Conference Series **60**(1), 58 (2007).

[AD10] E. Aprile and T. Doke, Liquid xenon detectors for particle physics and astrophysics, Rev. Mod. Phys. **82**, 2053--2097 (Jul 2010).

[ATLAS08] G. Aad et al. (ATLAS Collaboration), The ATLAS Experiment at the CERN Large Hadron Collider, Journal of Instrumentation **3**(08), S08003 (2008).

[BBS91] K. G. Begeman, A. H. Broeils and R. H. Sanders, Extended rotation curves of spiral galaxies: dark haloes and modified dynamics, MNRAS , 523 (1991).

[Bes12] B. Beskers, Private discussions and material, 2012.

[BFN96] D. Buttlar, J. Farrell and B. Nichols, PThreads Programming, O'Reilly Media, 1996.

[BHS05] G. Bertone, D. Hooper and J. Silk, Particle dark matter: Evidence, candidates and constraints, Phys.Rept. **405**, 279--390 (2005), hep-ph/0404175.

[CBG+06] D. Clowe, M. Bradac, A. H. Gonzalez, M. Markevitch, S. W. Randall et al., A direct empirical proof of the existence of dark matter, Astrophys.J. **648**, L109--L113 (2006), astro-ph/0608407.

[HTF+83] A. Hitachi, T. Takahashi, N. Funayama, K. Masuda, J. Kikuchi and T. Doke, Effect of ionization density on the time dependence of luminescence from liquid argon and xenon, Phys. Rev. B **27**, 5279--5285 (May 1983).

[JBD+11] N. Jarosik, C. L. Bennett, J. Dunkley, B. Gold, M. R. Greason, M. Halpern, R. S. Hill, G. Hinshaw, A. Kogut, E. Komatsu, D. Larson, M. Limon, S. S. Meyer, M. R. Nolta, N. Odegard, L. Page, K. M. Smith, D. N. Spergel, G. S. Tucker, J. L. Weiland, E. Wollack and E. L. Wright, Seven-year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Sky Maps, Systematic Errors, and Basic Results, ApJS **192**, 14 (February 2011), 1001.4744.

[KHR78] S. Kubota, M. Hishida and J. Raun, Evidence for a triplet state of the self-trapped exciton states in liquid argon, krypton and xenon, Journal of Physics C: Solid State Physics **11**(12), 2645 (1978).

[KHSzR79] S. Kubota, M. Hishida, M. Suzuki and J. zhi Ruan(Gen), Dynamical behavior of free electrons in the recombination process in liquid argon, krypton, and xenon, Phys. Rev. B **20**, 3486--3496 (Oct 1979).

[KNT+78] S. Kubota, A. Nakamoto, T. Takahashi, T. Hamada, E. Shibamura, M. Miyajima, K. Masuda and T. Doke, Recombination luminescence in liquid argon and in liquid xenon, Phys. Rev. B **17**, 2762--2765 (Mar 1978).

[KT03] L. Koopmans and T. Treu, The structure and dynamics of luminous and dark matter in the early-type lens galaxy of 0047-281 at z=0.485, Astrophys.J. **583**, 606--615 (2003), astro-ph/0205281.

[MACHO00] C. Alcock et al. (MACHO Collaboration), The MACHO project: Microlensing results from 5.7 years of LMC observations, Astrophys.J. **542**, 281--307 (2000), astro-ph/0001272.

[See01] G. M. Seed, An Introduction to Object-Oriented Programming in C++: With Applications in Computer Graphics, Springer, 2001.

[SKG11] N. A. Solter, S. J. Kleper and M. Gregoire, Professional C++, John Wiley & Sons, second edition edition, 2011.

[Tri87] V. Trimble, Existence and nature of dark matter in the universe, Annual review of astronomy and astrophysics **25**, 425--472 (1987).

[XENON10011] E. Aprile, K. Arisaka, F. Arneodo, A. Askin, L. Baudis, A. Behrens, K. Bokeloh, E. Brown, T. Bruch, G. Bruno, J. M. R. Cardoso, W.-T. Chen, B. Choi, D. Cline, E. Duchovni, S. Fattori, A. D. Ferella, F. Gao, K.-L. Giboni, E. Gross, A. Kish, C. W. Lam, J. Lamblin, R. F. Lang, C. Levy, K. E. Lim, Q. Lin, S. Lindemann, M. Lindner, J. A. M. Lopes, K. Lung, T. Marrodán Undagoitia, Y. Mei, A. J. Melgarejo Fernandez, K. Ni, U. Oberlack, S. E. A. Orrigo, E. Pantic, R. Persiani, G. Plante, A. C. C. Ribeiro, R. Santorelli, J. M. F. dos Santos, G. Sartorelli, M. Schumann, M. Selvi, P. Shagin, H. Simgen, A. Teymourian, D. Thers, O. Vitells, H. Wang, M. Weber and C. Weinheimer (XENON100 Collaboration), Dark Matter Results from 100 Live Days of XENON100 Data, Phys. Rev. Lett. **107**, 131302 (Sep 2011).

[Zor12] J. Zornoza, Dark matter search with the ANTARES neutrino telescope, (2012), 1204.5066.

A lXeMon Server Configuration XML

```
<!--
    This is an example lXeMon Server (lXMS) Configuration XML file.

    We start by making a parent node for lXMSs configuration.
    The measurement period lenght (in ms) can be defined as an
        attribute
    of the parent node.
-->
<lXeMonConfiguration period="500">

    <!--
        All the configuration for lXMS takes place inside this node.
        The order is not important, only the structure hierarchy is.
    -->

    <!-- Database configuration, used to connect to the database -->
    <Database host="localhost" user="lxemon_server" password="xyz"
        database="lxemon_data" />
    <!--
        Notice that the Database node is self-closing with a />,
        this is equivalent to writing <Database attr="abc"></Database>
    -->

    <!-- Devices configuration, used to initialize and set up devices
        -->
    <Devices>

        <!--
            Each device should have a "type" and a "name" attribute.
                Other attr-
            ibutes are implementation-specific to each device
                implementation.

            Supported Device types (As of 8.5.2012):
            - MccUsb1608g
            - SrsCtc100
        -->

        <!-- Adding a MccUsb1608g devie -->
        <Device type="MccUsb1608g" name="MCC USB-1608G" serial="016
            A4C28" samples="100">
            <!--
                Each device can have multiple Sensors attached to it.
```

```
                Each
            Sensor must have a unique "name" attribute, other
                attributes for
            Sensors are specific to device implementations.

            A MccUsb1608G Sensor represents a channel on the devide
                 and
            has a "mode" (either DIFF (differential) or SE (single
                ended).
            Samples is the amount of samples the Server requests
                and then
            averages over.
        -->
        <Sensor name="ADC_ch0" channel="0" mode="DIFF" voltage="10"
             samples="100" />
        <Sensor name="ADC_ch1" channel="1" mode="DIFF" voltage="10"
             samples="100" />
        <Sensor name="ADC_ch2" channel="2" mode="DIFF" voltage="10"
             samples="100" />
        <Sensor name="ADC_ch3" channel="3" mode="DIFF" voltage="10"
             samples="100" />
    </Device>

    <!--
        Adding a second device of the type SrsCtc100, which uses an
             Ethernet
        connection and so needs an ip attribute. The SRS100 has
            more
        channels than the ones shown here, but these will suffice
            as an
        example
    -->
    <Device type="SrsCtc100" name="SRS CTC100 Cryocontroller" ip="
        192.168.1.2">
        <Sensor name="SRSCTC_In1" channel="In 1" />
        <Sensor name="SRSCTC_In2" channel="In 2" />
        <Sensor name="SRSCTC_In3" channel="In 3" />
        <Sensor name="SRSCTC_In4" channel="In 4" />
        <Sensor name="SRSCTC_Out1" channel="Out 1" />
        <Sensor name="SRSCTC_Out2" channel="Out 2" />
        <Sensor name="SRSCTC_Relay1" channel="Relay 1" />
        <Sensor name="SRSCTC_Relay2" channel="Relay 2" />
    </Device>
</Devices>

<!-- Parameters configuration -->
<Parameters>
    <!--
        Each Parameter needs a "name" and a "type" attribute.
            Optionally a
        "unit" attribute should be set (or it defaults to an empty
```

```
            string).
        A description should be provided as well for a clearer
            distinction
        between parameters for viewers.

        A parameter normally has one or more Sources, from which it
             receives
79      its raw data.

        If multiple sources are defined for a single parameter,
            they will be
        combined to make up the parameter value depending on the "
            method"
        attribute of the Parameter. Supported methods are "sum" (
            adding all
84      sources together) and "mult" (multiplying them).
    -->

    <Parameter name="flow1" unit="kg/s" method="sum" description="
        Flow 1 through flow meter">
        <!--
89          This is a flow parameter which gets its data from the
            MccUsb1608g device, namely from the previously defined
                Sensor
            named "ADC_ch0".
        -->
        <Source type="sensor" name="ADC_ch0">
94          <!--
                Converters are used to convert a raw value from a
                    Source
                (e.g. voltage from a sensor) and convert it to the
                    intended
                unit/dimension.

99              Supported Converters (As of 5.4.2012):
                    - LinearConverter (attributes: "factor" and "
                        offset")
                         converts linearly
                    - IntegratingConverter (attributes: "offset" (
                        baseline),
                             "unit-in-ns" (the amount of ns in one
                                 time unit one is integrating over))
104                      Integrates the value over time
            -->
            <Converter type="LinearConverter" factor="1" offset="0"
                 />
            <!-- The above converter actually will do nothing (
                multiply by
                1, but is included for demonstration purposes -->
109     </Source>
```

```
        <!--
            Each Parameter can have different Alarms set to watch
                over their
            value. If an alarm object determines a value is not
                acceptable,
            it will trigger an alert of level 1 (Logged only, no
                action
            taken) and if the value is not acceptable for a number
                of
            consecutive checks surpassing the "tolerance" attribute
                , it will
            trigger a higher-level warning defined by the "alert-
                level"
            attribute. In the Contacts section below one can
                configure how
            each contact receive each level of alerts.

            Supported Alarms (As of 8.5.2012):
                - RangeAlarm (attributes: "min" and "max")
                - SlopeAlarm (attributes: "min", "max" and "avgOver
                    ")
        -->
        <Alarm type="RangeAlarm" min="-8.0" max="8.0" tolerance="3"
             alert-level="2" />

    </Parameter>

    <!-- Now add 2 more Parameters without comments for
        demonstration -->

    <Parameter name="temperature1" unit="K" method="sum"
        description="T1">
        <Source type="sensor" name="SRSCTC_In1">
            <Converter type="LinearConverter" factor="1.543" offset
                ="-0.345" />
        </Source>
        <Alarm type="RangeAlarm" min="-8.0" max="8.0" tolerance="3"
             alert-level="2" />
        <Alarm type="RangeAlarm" min="-10.0" max="10.0" tolerance="
            3" alert-level="3" /> <!-- Higher level for larger
            values -->
    </Parameter>

    <Parameter name="mass1" unit="kg" method="sum" description="
        Mass 1 (integrated from flow1)">
        <Source type="parameter" name="flow1">
            <Converter type="IntegratingConverter" offset="0" unit-
                in-ns="1000000000" />
        </Source>
    </Parameter>

```

```
        <!--
            Notice that the last Parameter uses the first defined
                Parameter
            (flow1) as a source. It uses an IntegratingConverter to
                integrate
            the flow over time, yielding mass.
        -->

    </Parameters>

    <!--
        E-mail SMTP server configuration
        Required so that the lXeMon Server can send out e-mails when
            alarms
        happen. TLS Encryption is supported as well as authentication.
    -->
    <EmailConfiguration
        address="lxemon.jgu@googlemail.com"
        server="smtp.googlemail.com"
        port="587"
        encryption="TLS"
        authentication="true"
        user="lxemon.jgu@googlemail.com"
        password="liqxenmon"
    />

    <Contacts>
        <!--
            Contacts for alerts get added here.

            Each contact should include name, email, mobile, email-
                alert-level
            and sms-alert-level fields.

            When an alarm triggers of a certain level, it checks the
            email-alert-level and sms-aler-level fields of a contact
                and only
            takes the appropriate action if it is less than or equal to
                 the
            Alarm's level
        -->
        <Contact
            name="Elvar Kjartansson"
            email="kjartans@students.uni-mainz.de"
            mobile="0123123123"
            email-alert-level="2"
            sms-alert-level="3"
        />

    </Contacts>

```

```
        <!--
            And this demo configuration is hereby finished.
            Naturally, a real-world usage scenario would likely define many
                 more
            Parameters and sources, as well as many alarms for each
                Parameter
194         possibly.
        -->
    </lXeMonConfiguration>
```

B IXeMon Server Documentation