SAVONIA

Extending the Advanced Data Extraction Infrastructure

Research on HTML5 usage, server monitoring tool, support for multidimensional datasets

Riku Hytönen

Bachelor's Thesis

___.___.______ __________________________________

Ammattikorkeakoulututkinto

SAVONIA UNIVERSITY OF APPLIED SCIENCES **THESIS Abstract**

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Riku Hytönen			
Title of Thesis Extending the Advanced Data Extraction Infrastructure			
Date	30 May 2012	Pages/Appendices	28/20
Supervisor(s) Mr. Jussi Koistinen, Lecturer, Mr. Sami Lahti, Lecturer			
Client Organisation/Partners Dr. Suren Chilingaryan, Dr. Andreas Kopmann, Karlsruhe Institute of Technology			

Abstract

The Advanced Data Extraction Infrastructure (ADEI) project functions as a bridge between the control systems that collect the measurements of various subsystems controlling the flow of data acquisition from experiments, and scientists analysing the flow of experiments and evaluating the collected data. The project is still in development and new features are added constantly.

During the thesis the current version of the ADEI was extended by implementing a new data reader module to deal with Round Robin Database files that are used to monitor the status of the computers operating as part of control and data acquisition subsystems of the ADEI.

The future development of the ADEI was also researched. The current server-side graphing approach has significant limitations in scalability and requires a rather high bandwidth between the server and clients. The use of HTML5 markup language was investigated. The investigation focused on the usage of canvas along the JavaScript drawing library RGraph to implement the ADEI's graph drawing features on the client side of the application to lessen the need for high bandwidth. The test was successful and it is recommended to use the RGraph library as the platform for a new ADEI frontend.

The current version of the ADEI can only present simple time series data while many experiments generate multidimensional time series data. To achieve the visualization of multidimensional data, a prototype module handling several new types of data that could not be presented with the current ADEI was created to provide a foundation for the further development of these features.

Keywords
HTML, PHP, JavaScript, Round Robin Database, Multidimensional Data

SAVONIA-AMMATTIKORKEAKOULU **OPINNÄYTETYÖ Tiivistelmä**

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Riku Hytönen	
Työn nimi Web-sovelluksen laajentaminen	
Päiväys 30.05.2012	Sivumäärä/Liitteet 28/20
Ohjaaja(t) Lehtori Jussi Koistinen, Lehtori Sami Lahti,	
Toimeksiantaja/Yhteistyökumppani(t) Tohtori Suren Chilingaryan, Tohtori Andreas Kopmann, Karlsruhe Institute of Technology	

Tiivistelmä

ADEI projektin kehittämä web-pohjainen sovellus toimii linkkinä erilaisten mittaustuloksia kerääavien laitteistojen ja tuloksia analysoivien tutkijoiden välillä. Projektia kehitetään ja uusia ominaisuuksia lisätään jatkuvasti.

Osana opinnäytetyötä tutkittiin HTML5 merkkauskielen mahdollista käyttöä ADEIn jatkokehittämisessä keskittyen tarkastelemaan uutta canvas elementtiä ja piirtämiseen tarkoitetun RGraph JavaScript kirjaston käyttöä toteuttamaan ADEIn nykyiset kuvaajien piirto-ominaisuudet asiakaslaitteen puolella. Näin pystytään vähentämään palvelimen ja asiakaslaitteiden välisen korkean kaistanleveyden tarvetta. Testi onnistui ja suosittelen Rgraphia käytettäväksi ADEIn uuden käyttöliittymän kehityksessä.

Lisäksi kehitettiin nykyiseen ADEI versioon uusi lukurajapinnan laajennus käsittelemään Round Robin Database tiedostoja jota tultaisiin käyttämään ADEI palvelimien suorituskyvyn tarkkailuun.

Moniulotteisen datan, jota ei pystytä esittämään selkeästi nykyisten piirto-ominaisuuksien avulla, esittämistä varten kehitettiin kokeellinen moduuli. joka muodostaa pohjan näiden ominaisuuksien tulevalle jatkokehittämiselle.

Avainsanat
HTML, PHP, JavaScript, Round Robin Database, Moniulotteinen Data

TABLE OF CONTENTS

ABBREVIATIONS AND NOTIONS

KIT	Karlsruhe Institute of Technology
IPE	Institute for Data Processing and Electronics
ADEI	Advanced Data Extraction Infrastructure
KATRIN	Karlsruhe Tritium Neutrino Experiment
TOSKA	Test Facility for Fusion Magnets
MySQL	A widely used open source database engine
CSV	Comma-separated values
XSL	Extensible Stylesheet Language
HTML5	Hypertext Markup Language
AJAX	Asynchronous JavaScript and XML
PHP	A server-side HTML embedded scripting language
XSLT	Extensible Stylesheet Language Transformations
API	Application Programming Interface
WebGL	An API that conforms closely to the OpenGL ES 2.0 API for 3D rendering
OpenGL	A cross-language, cross-platform API for the writing of applications that produce 2D and 3D computer graphics
2D	Two dimensional
3D	Three dimensional

1 INTRODUCTION

Various scientific experiments generate a lot of data and researchers need a way to investigate their experiment data. In fusion experiments the data can be stored at rates exceeding several kHz. Meteorological data on the other hand needs to be archived for over hundreds of years. The Advanced Data Extraction Infrastructure (ADEI) project was started in the Institute for Data Processing and Electronics (IPE) at the Karlsruhe Institute of Technology (KIT) to solve this problem. The ADEI stores and displays data from various experiments with different timescales and rates.

The development of the ADEI project will be further continued during the thesis. The possibilities of using HTML5 markup language in the future development of the ADEI will be investigated, a new interface for reading and presenting the data generated by the server performance monitoring tool Munin will be created and the ADEI will be extended to support the visualization of multidimensional data measured by the meteorological equipment operating at the KIT weather tower.

The ADEI has been developed to provide scientists with data exploration capabilities to a broad range of physical experiments dealing with time series data. Examples of such systems include slow control systems for maintaining archives of sensor generated data for long periods of time, cosmic ray experiments evaluating fluxes of cosmic rays incident on the detector surface, meteorological systems measuring various aspects of the weather, etc. (Chilingaryan S., Beglarian A., Kopmann A. and Vöcking S. 2010)

These systems have very different characteristics from each other: the amount of data channels, the types of the channels and their sampling rates, etc. Different data acquisition and control subsystems store their data in varying ways utilizing various data formats and database engines. Furthermore, scientists require their data to be in specific formats that are supported by the tools they use to analyze their experiment data.

The ADEI project functions as a bridge between the acquisition and the analyzing of experiment data. This is done by reading data from various different experiments and storing it in its own MySQL databases from which the data can be exported in different formats such as CSV and XSL for further analysis. The ADEI's ability to draw graphical representations of the experiment data also provides scientists with the possibility to examine the validity and integrity of their measurements. The ADEI also provides the required interfaces to enable the expanding of these reading and exporting functions to support new types of data and new export formats for the data. (The ADEI Project 2012)

The ADEI is complex web based software implementing multiple functions needed for caching, aggregating, filtering and exporting of the experiment data. The software consists of well over a hundred files containing approximately 40 000 lines of code. Currently five different authors have contributed in the development of the code.

The architecture of the ADEI can be seen in Figure 1. There are three distinct parts to the figure; the example data sources found at the bottom (red), the actual ADEI classes in the middle (blue) and some clients that use the ADEI to access the data on top (green).

The clients get data from the ADEI as JSON/XML encoded data by using HTTP GET and POST requests. Some of the ADEI classes deal with exporting the data to the clients from the ADEI databases in the format required by the client.

Other classes function close to the data sources, dealing with reading the data from different sources, iterating it and storing it in ADEI’s MySQL databases while also trying to ensure that the data being stored is valid and consistent.

There are also classes that handle searching through the chosen data and creating a graphical representation of it for the user. This also includes the filtering, aggregation and caching of the data is fetched for the user

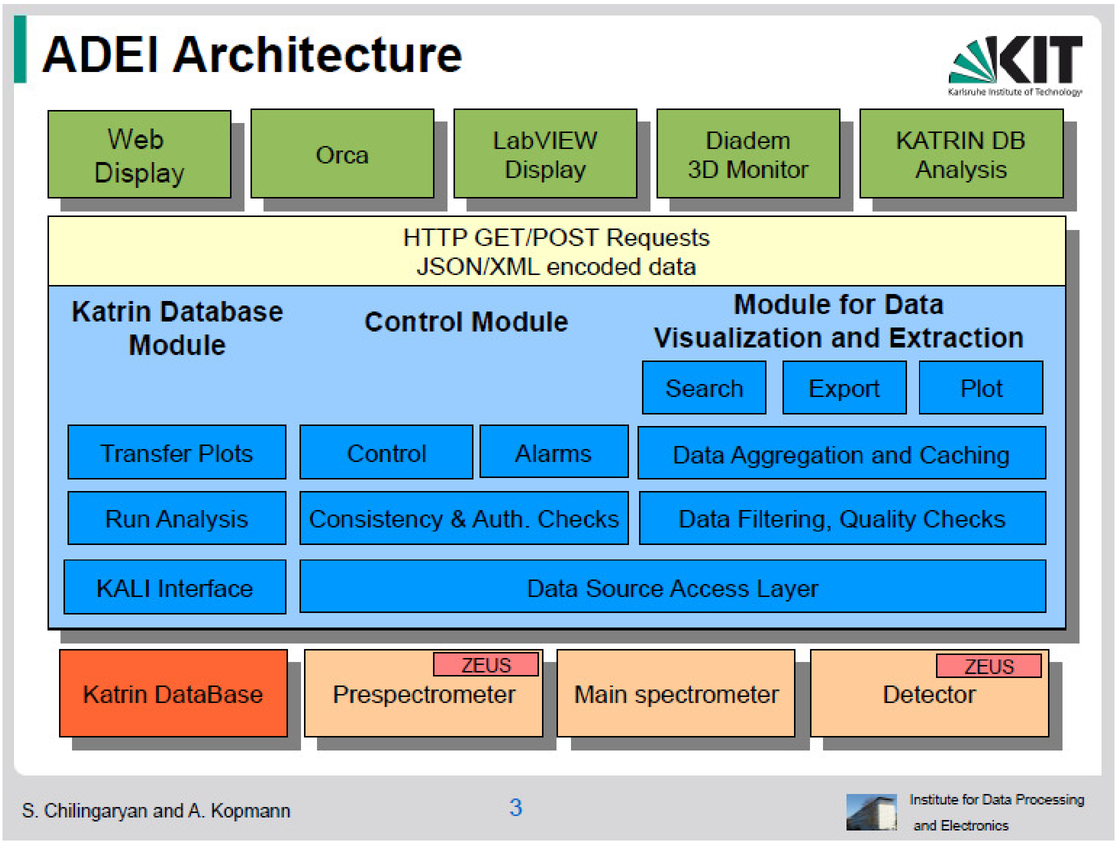


FIGURE 1. The ADEI architecture (Chilingaryan S., Kopmann A. 2011)

The ADEI is licensed under GNU General Public License and only free open source technologies are used in its development. It is currently used to support several experiments at KIT, including the slow control systems of the Karlsruhe Tritium Neutrino (KATRIN) experiment to measure the mass of the electron antineutrino and the Test Facility for Fusion Magnets (TOSKA), a low-temperature test facility for superconducting magnet coils and other technical components to be used in the study and development of fusion technology.

2 ENVIRONMENT

2.1 Used Techniques

The techniques used in the extensions of the ADEI project code include PHP, HTML, XML, XSLT and JavaScript. The server side functionality was implemented using PHP while the client side functionality utilizes JavaScript and the user interface is constructed from XML data returned by the server using XSLT templates.

2.2 Tools

The development work was done on a desktop computer running OpenSuse Linux. The prototype and proof of concept code files were created with the KEdit text editor and tested locally on an Apache2 server running on the development machine.

The extensions of an actual instance of the ADEI were done by using the Linux terminal to connect to a server machine running the ADEI instance. The editor used to write code while connected to the server machine was Midnight Commander Editor.

3 EXTENDING THE ADEI

The thesis consisted of three distinct parts dealing with the extension of the ADEI. These parts were the research work on using HTML5 in the future development of the ADEI, the creation of a new RRDREADER interface to be used for monitoring the performance of the ADEI servers and the creation of a new module to visualize multidimensional datasets in the ADEI.

3.1 HTML5

As the first part of the thesis the new features of HTML5 and their possible usage in the future development of the ADEI project were investigated.

HTML is a language used to structure the presentation of content on web pages. The aim of HTML5, which is currently in development, is to improve the language with new support features for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices. (W3C 2012)

3.1.1 Previous Research

The new features of HTML5 and their usage in the development of the ADEI had already been researched by the IPE before this thesis. The future changes that were suggested previously include the possible usage of the new web workers and web sockets to update the graphs in ADEI in real time. This can be seen in Figure 3. The currently implemented handling of graph updating using AJAX requests initiated by the user can be seen in Figure 2.

The web worker and web socket APIs are new features of HTML5. The workers are used to separate JavaScript tasks requiring heavy computations into different threads to be executed independently to prevent the code from making a website unresponsive for the remainder of its execution. The web socket API is used to create two way communication channels between web servers and their client applications thus enabling real-time bi-directional communication.

There would also be a possibility to present certain types of multidimensional data e.g. weather maps as Moving Pictures by using the new video tag of HTML5. In addition, some measurement data could be clarified by using the WebGL API to present the object being measured and the locations of the sensors on the object. The WebGL API, included in the HTML5 specification, offers 3D rendering in HTML context and is designed to be used with the canvas tag to provide rendering functionality similar to OpenGL.

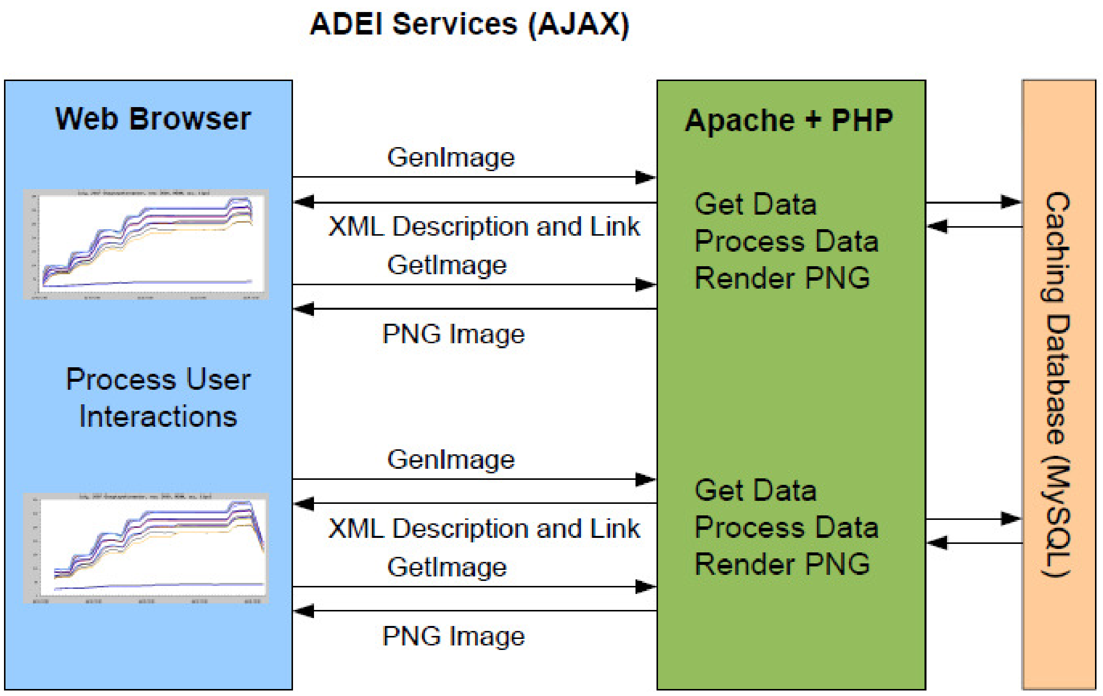


FIGURE 2. The current implementation of graphing in the ADEI (Chilingaryan S., Kopmann A. 2011)

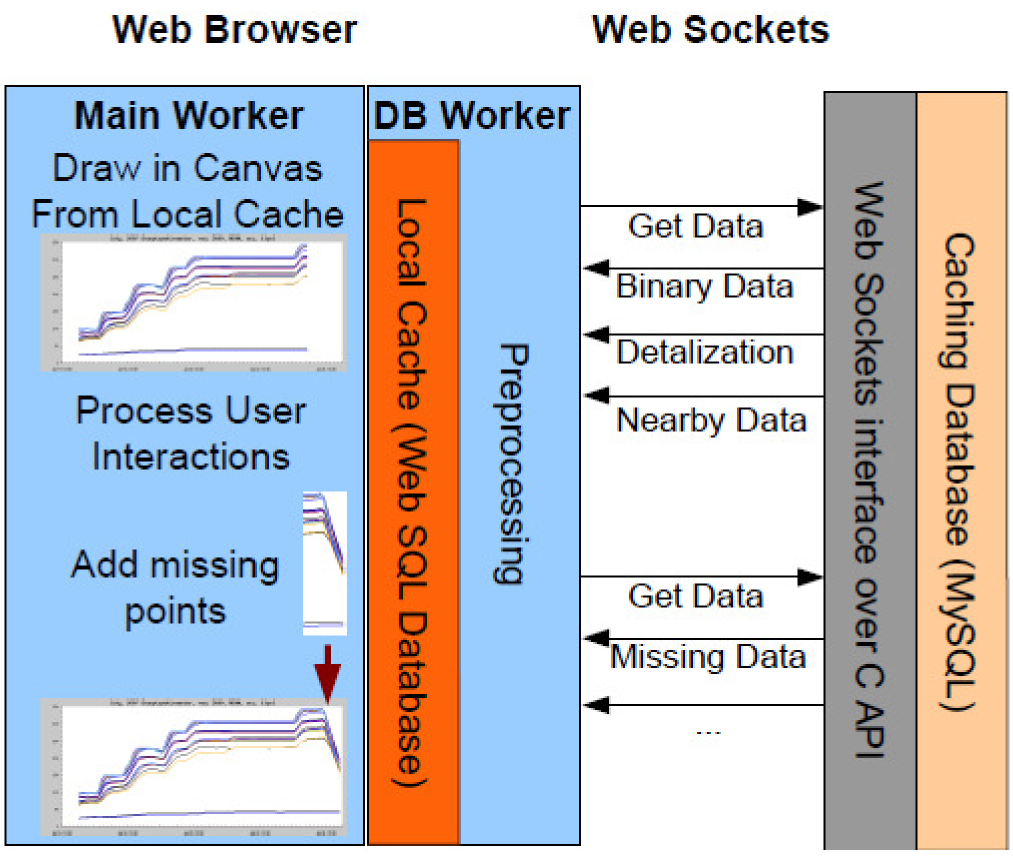


FIGURE 3. A suggested approach to graphing in the ADEI using HTML5 (Chilingaryan S., Kopmann A. 2011)

Most of the suggested changes, such as using the web worker and web socket APIs to remodel the graphing of the ADEI, would require a lot of reworking on the software. Thus, considering the time available for the research, the main focus was on the new canvas tag and the possibility of replacing the current graphing system using the RGraph JavaScript graphing library that had also been mentioned in the previous research as a possible tool for the future graphing tools of the ADEI.

The relevant new functionality offered by the new attributes introduced in HTML5 includes e.g. the ‘hidden’ attribute that determines whether an element is visible or not. However, these features have already been implemented in the current state of the ADEI by utilizing the dynamic changing of style sheets to e.g. alter the visibility of different objects in the interface. Still, if a future version of the ADEI will be developed using HTML5 to a greater extent, the adjusting of the look and feel of the web page could be made simpler by utilizing these new attributes.

3.1.2 Canvas Element

Out of the new tags provided by HTML5, the Canvas tag offers a promising way to present the experiment data in the ADEI. The Canvas tag enables the creation of images using JavaScript on the client side as opposed to the current version which builds the graphical presentation of the data on the server side using PHP. This would free up server and network capacity for other tasks such as filtering and sifting through the requested data thus enabling the application to run faster. For this reason, the usage of the RGraph JavaScript library to draw graphs from various types of data was tested.

3.1.3 RGraph JavaScript Library

RGraph is an HTML5 JavaScript library for drawing charts on the new canvas element of HTML5 (RGraph 2012). It has been previously proposed to be used in graph drawing for the future HTML5 version of the ADEI.

To test the library, several small scale tests on creating graphs were conducted locally using preset values for graph data. A simple example graph can be seen in Figure 4.

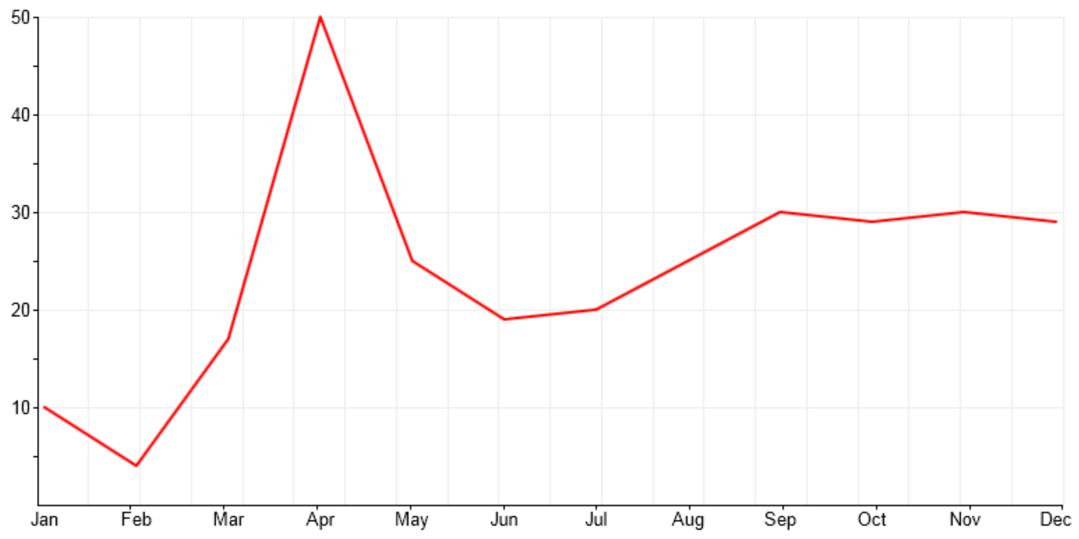
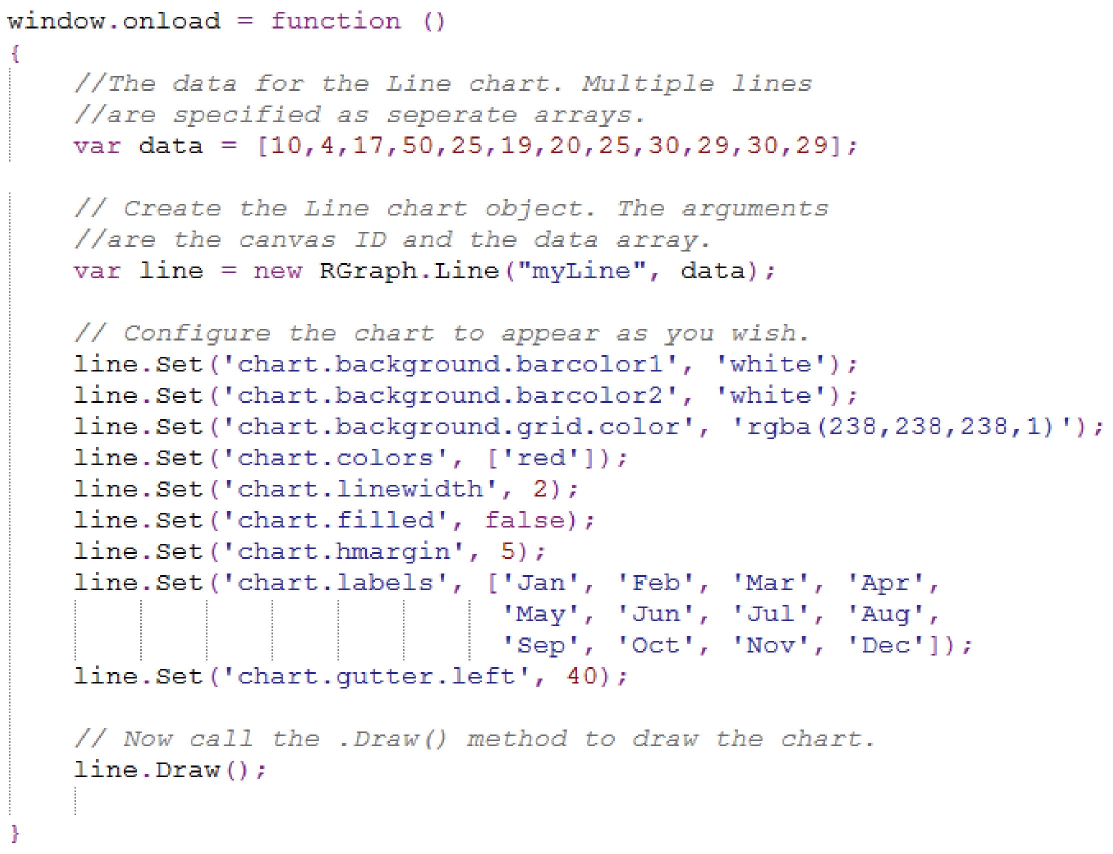


FIGURE 4. An example graph generated with RGraph

The code used to generate the above example graph can be seen in Figure 5.



```
window.onload = function ()
{
    //The data for the Line chart. Multiple lines
    //are specified as seperate arrays.
    var data = [10,4,17,50,25,19,20,25,30,29,30,29];

    // Create the Line chart object. The arguments
    //are the canvas ID and the data array.
    var line = new RGraph.Line("myLine", data);

    // Configure the chart to appear as you wish.
    line.Set('chart.background.barcolor1', 'white');
    line.Set('chart.background.barcolor2', 'white');
    line.Set('chart.background.grid.color', 'rgba(238,238,238,1)');
    line.Set('chart.colors', ['red']);
    line.Set('chart.linewidth', 2);
    line.Set('chart.filled', false);
    line.Set('chart.hmargin', 5);
    line.Set('chart.labels', ['Jan', 'Feb', 'Mar', 'Apr',
                              'May', 'Jun', 'Jul', 'Aug',
                              'Sep', 'Oct', 'Nov', 'Dec']);
    line.Set('chart.gutter.left', 40);

    // Now call the .Draw() method to draw the chart.
    line.Draw();

}
```

FIGURE 5. The JavaScript code of the RGraph example

The library supports a wide variety of different graphs and is simple and effective to use in the creation of graphs. The ability to draw multiple line plots on a single graph is supported by the library and thus it contains, in theory, all the features required by the current stable version of the ADEI for drawing the graphs.

3.1.4 Conclusions

Regarding the previously done research on the new features of HTML5, discussed in chapter 3.1.1, the new ideas suggested seem to be well thought out and worth implementing in the future.

In conclusion for the research conducted for the thesis, the canvas tag seems to be a valid way to replace the current way of creating graphs for the ADEI in future versions. Based on the tests conducted on Firefox 3.6, Firefox 4 and Opera 11, the RGraph library seems to be versatile and it seems to contain all the features needed to implement the same functionality as the current ADEI graphing classes do.

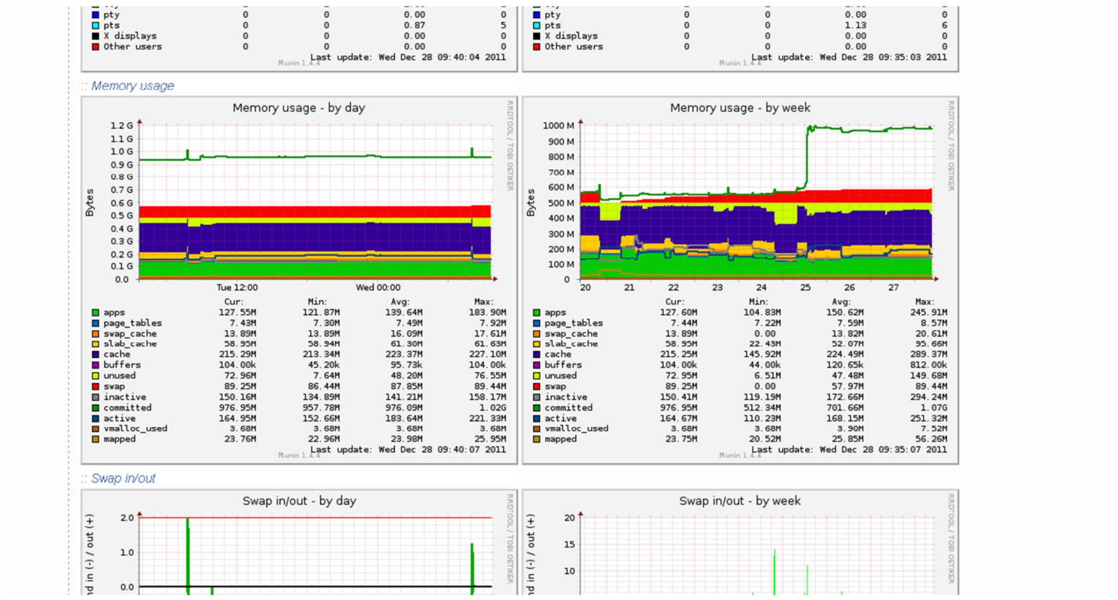
3.2 Performance Monitoring Tool for ADEI

3.2.1 Round Robin Database

Round Robin Databases (RRD) created by the RRDTool open source logging tool (Oetiker T. 2011) are traditionally used in the UNIX world to store time series data produced by resource monitoring tools. This storage format is also used by the resource monitoring tool Munin, which has been used to monitor the ADEI servers and their performance.

3.2.2 Munin

Munin is a tool created with the Perl programming language for monitoring system resources, storing the resource data in RRD-files and graphing the stored data so it can be viewed in the Munin interface using a web browser (The Munin Project 2012). Part of the interface can be seen in Figure 6.



🟦 pty	0	0	0.00	0
🟦 pts	0	0	0.87	5
⬛ X displays	0	0	0.00	0
🟥 Other users	0	0	0.00	0

Last update: Wed Dec 28 09:40:04 2011

🟦 pty	0	0	0.00	0
🟦 pts	0	0	1.13	6
⬛ X displays	0	0	0.00	0
🟥 Other users	0	0	0.00	0

Last update: Wed Dec 28 09:35:03 2011

	Cur:	Min:	Avg:	Max:
🟩 apps	127.55M	121.87M	139.64M	183.90M
🟦 page_tables	7.43M	7.30M	7.49M	7.92M
🟧 swap_cache	13.89M	13.89M	16.09M	17.61M
🟨 slab_cache	58.95M	58.94M	61.30M	61.63M
🟦 cache	215.29M	213.34M	223.37M	227.10M
🟪 buffers	104.00k	45.20k	95.73k	104.00k
🟨 unused	72.96M	7.64M	48.20M	76.55M
🟥 swap	89.25M	86.44M	87.85M	89.44M
⬜ inactive	150.16M	134.89M	141.21M	158.17M
🟩 committed	976.95M	957.78M	976.09M	1.02G
🟦 active	164.95M	152.66M	183.64M	221.33M
🟫 vmalloc_used	3.68M	3.68M	3.68M	3.68M
🟫 mapped	23.76M	22.96M	23.98M	25.95M

Last update: Wed Dec 28 09:40:07 2011

	Cur:	Min:	Avg:	Max:
🟩 apps	127.60M	104.83M	150.62M	245.91M
🟦 page_tables	7.44M	7.22M	7.59M	8.57M
🟧 swap_cache	13.89M	0.00	13.82M	20.61M
🟨 slab_cache	58.95M	22.43M	52.07M	95.66M
🟦 cache	215.25M	145.92M	224.49M	289.37M
🟪 buffers	104.00k	44.00k	120.65k	812.00k
🟨 unused	72.95M	6.51M	47.48M	149.68M
🟥 swap	89.25M	0.00	57.97M	89.44M
⬜ inactive	150.41M	119.19M	172.66M	294.24M
🟩 committed	976.95M	512.34M	701.66M	1.07G
🟦 active	164.67M	110.23M	168.15M	251.32M
🟫 vmalloc_used	3.68M	3.68M	3.90M	7.52M
🟫 mapped	23.75M	20.52M	25.85M	56.26M

Last update: Wed Dec 28 09:35:07 2011

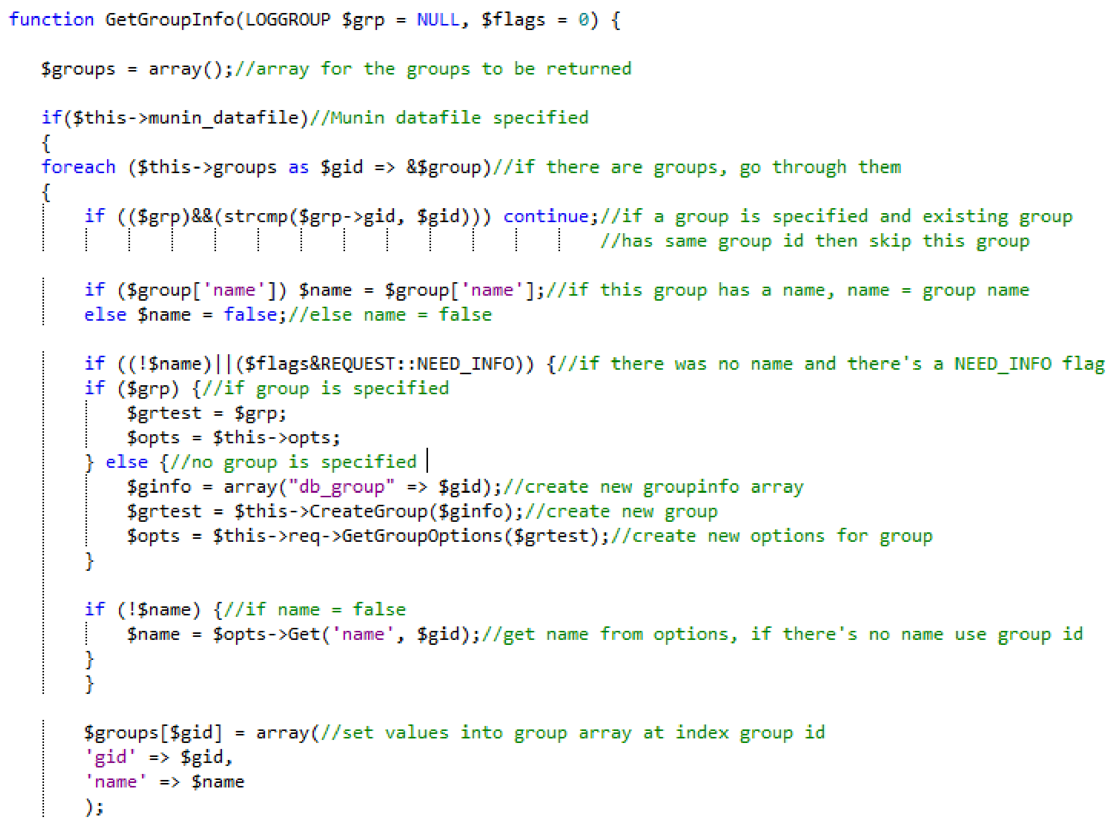
FIGURE 6. Munin web interface

3.2.3 Requirements

The RRDREADER and several other classes were created to provide a way to view the server performance data stored by Munin using the ADEI interface. The required features of the system were to be able to read the data stored in RRD files by Munin and to automatically create different log groups from the Munin created RRD files as specified in the Munin's configuration file so that for example, all the files storing CPU usage percentage data could be drawn in the same graph. Also, since some of the data stored by Munin assumes that the graph would be drawn as stacked filled line plots, this mode of drawing graphs also needed to be implemented to enable accurate graphing of the Munin data using the ADEI web interface. The reader was also required to be able to handle the reading of generic RRD files not associated with Munin to enable the use of other RRD files as data sources for the ADEI in the future.

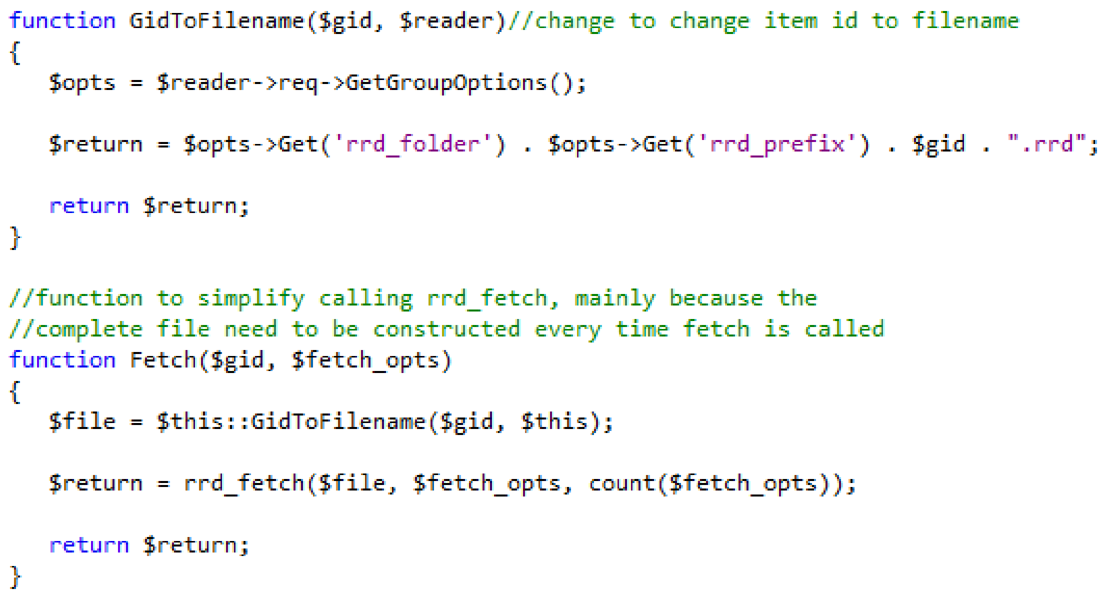
3.2.4 Implementation

The new RRDREADER class extends the ADEI's reader interface by implementing the basic methods GetGroupInfo, GetItemList, GetRawData and HaveData that are found in its parent class READER. As can be seen in Figure 7, the reader class utilizes its inherited variables and methods such as the CreateGroup method and the $flags variable among many others. The class also contains several methods used to acquire the information required by the base class methods to extract data out of the Munin created RRD files and their configuration file. Some of the class methods can be seen in Figure 8.



```
function GetGroupInfo(LOGGROUP $grp = NULL, $flags = 0) {

   $groups = array();//array for the groups to be returned

   if($this->munin_datafile)//Munin datafile specified
   {
   foreach ($this->groups as $gid => &$group)//if there are groups, go through them
   {
       if (($grp)&&(strcmp($grp->gid, $gid))) continue;//if a group is specified and existing group
                                                       //has same group id then skip this group

       if ($group['name']) $name = $group['name'];//if this group has a name, name = group name
       else $name = false;//else name = false

       if ((!$name)||($flags&REQUEST::NEED_INFO)) {//if there was no name and there's a NEED_INFO flag
       if ($grp) {//if group is specified
           $grtest = $grp;
           $opts = $this->opts;
       } else {//no group is specified
           $ginfo = array("db_group" => $gid);//create new groupinfo array
           $grtest = $this->CreateGroup($ginfo);//create new group
           $opts = $this->req->GetGroupOptions($grtest);//create new options for group
       }

       if (!$name) {//if name = false
           $name = $opts->Get('name', $gid);//get name from options, if there's no name use group id
       }
       }

       $groups[$gid] = array(//set values into group array at index group id
       'gid' => $gid,
       'name' => $name
       );
```

FIGURE 7. The beginning of the RRDREADER's GetGroupInfo function

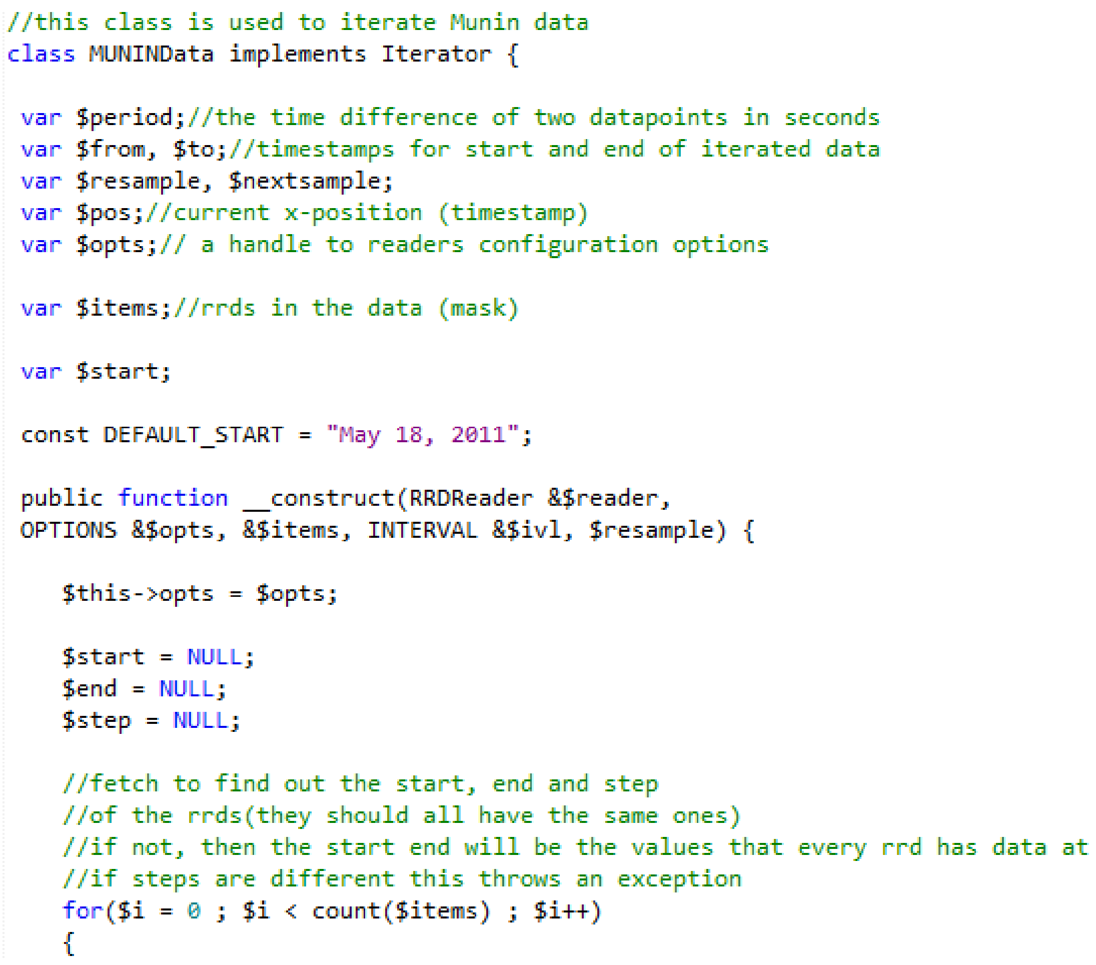


```
function GidToFilename($gid, $reader)//change to change item id to filename
{
   $opts = $reader->req->GetGroupOptions();

   $return = $opts->Get('rrd_folder') . $opts->Get('rrd_prefix') . $gid . ".rrd";

   return $return;
}

//function to simplify calling rrd_fetch, mainly because the
//complete file need to be constructed every time fetch is called
function Fetch($gid, $fetch_opts)
{
   $file = $this::GidToFilename($gid, $this);

   $return = rrd_fetch($file, $fetch_opts, count($fetch_opts));

   return $return;
}
```

FIGURE 8. A few methods of the RRDREADER class

In order to get the data from the Munin generated RRD files, the reader uses RRDTool and its PHP extension to read the files. Because the PHP extension did not implement all the functions required to get the necessary information from the files, in some cases, the RRDTool was used with the PHP shell_exec() method to get the information.

In addition to the RRDREADER class, several other additions and changes were made in the ADEI. For the iteration of the data returned by the reader classes, two new classes were created. One of the classes, RRDDATA is to be used when no Munin configuration file is specified in the ADEI server configuration. This class exists for the purposes of future development, enabling the reading of data stored in other systems that use RRD files as their data storage. The other class, MUNINDATA is used with the data that is read from Munin created RRD files. By using the additional information read from the Munin configuration file, this class is able to handle the data so that it can be presented exactly as Munin's own web interface does it. Both of these classes implement the PHP Iterator interface and are modeled after the iterating classes used by other ADEI readers. A part of the MUNINDATA class code can be seen in Figure 9.



```
//this class is used to iterate Munin data
class MUNINData implements Iterator {

 var $period;//the time difference of two datapoints in seconds
 var $from, $to;//timestamps for start and end of iterated data
 var $resample, $nextsample;
 var $pos;//current x-position (timestamp)
 var $opts;// a handle to readers configuration options

 var $items;//rrds in the data (mask)

 var $start;

 const DEFAULT_START = "May 18, 2011";

 public function __construct(RRDReader &$reader,
 OPTIONS &$opts, &$items, INTERVAL &$ivl, $resample) {

    $this->opts = $opts;

    $start = NULL;
    $end = NULL;
    $step = NULL;

    //fetch to find out the start, end and step
    //of the rrds(they should all have the same ones)
    //if not, then the start end will be the values that every rrd has data at
    //if steps are different this throws an exception
    for($i = 0 ; $i < count($items) ; $i++)
    {
```

FIGURE 9. The beginning of MUNINDATA class declaration

A new MUNINAXES class was created by extending the existing GRAPHAXES class, one of the several classes used by the ADEI to generate the complete graphical presentation of data, to handle the y-axis name information when dealing with Munin generated data.

The classes that were created and modified when implementing the RRDREADER functionality can be seen in Figure 10. The red colored boxes represent the new classes created specifically for the RRDREADER. The orange DRAW class was slightly modified to accommodate for the drawing of filled line plots when specified in the data read from Munin RRD files.

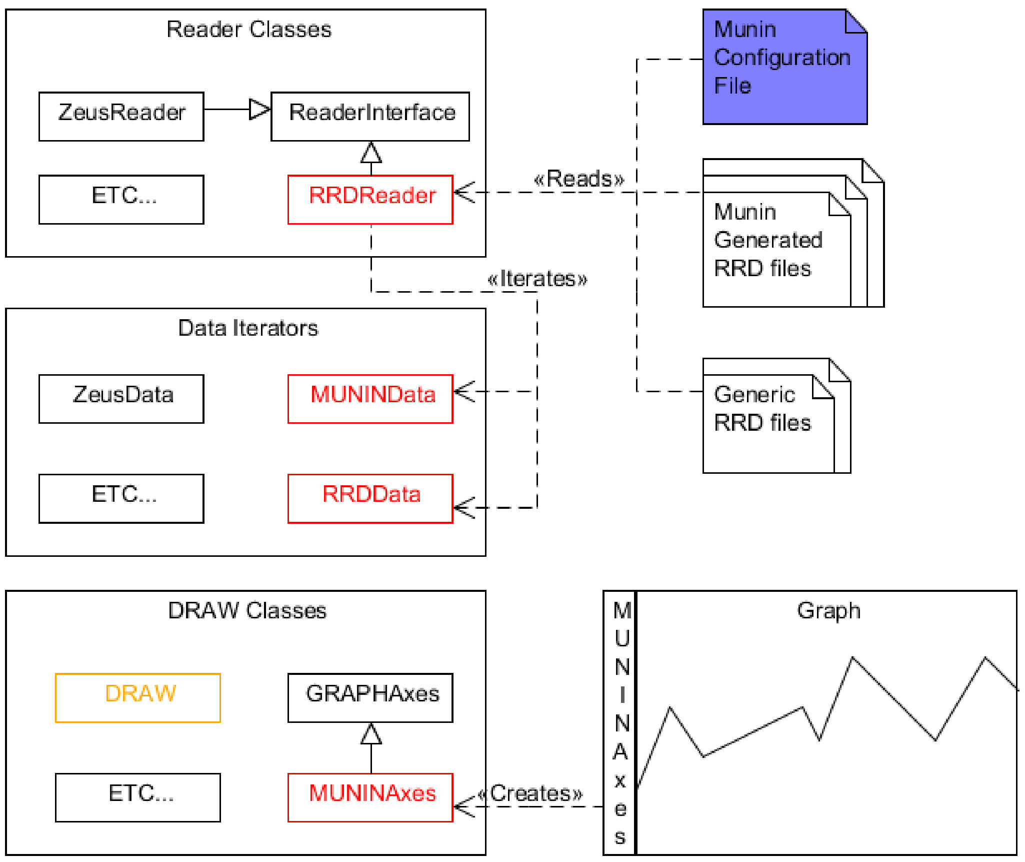


FIGURE 10. The classes created and modified while implementing the RRDReader

An addition to the user interface in the form of a new control module called Plot was also made to enable the switching between the standard graphing mode and the newly added Munin mode. This module could be easily expanded in the future if additional modifiers are needed for the drawing system.

For the purposes of drawing graphs from the Munin generated files, changes needed to be made to the main DRAW class of the ADEI to account for a new parameter from the user interface control module mentioned in the previous paragraph. This parameter changes the drawing mode so that it takes into account the type of the plot needed to be drawn for each data source. This also caused the removal of the vertical zooming option in the JavaScript code of the image cropping system used to zoom in on the presented data in the ADEI.

3.2.5 Conclusions

The extension of the ADEI reader interface to include RRD files was successful and the new extension is already in use on KIT's ADEI servers supporting the KATRIN and the KITCube experiments. A graph made using the data provided by the RRDREADER can be seen in Figure 11.

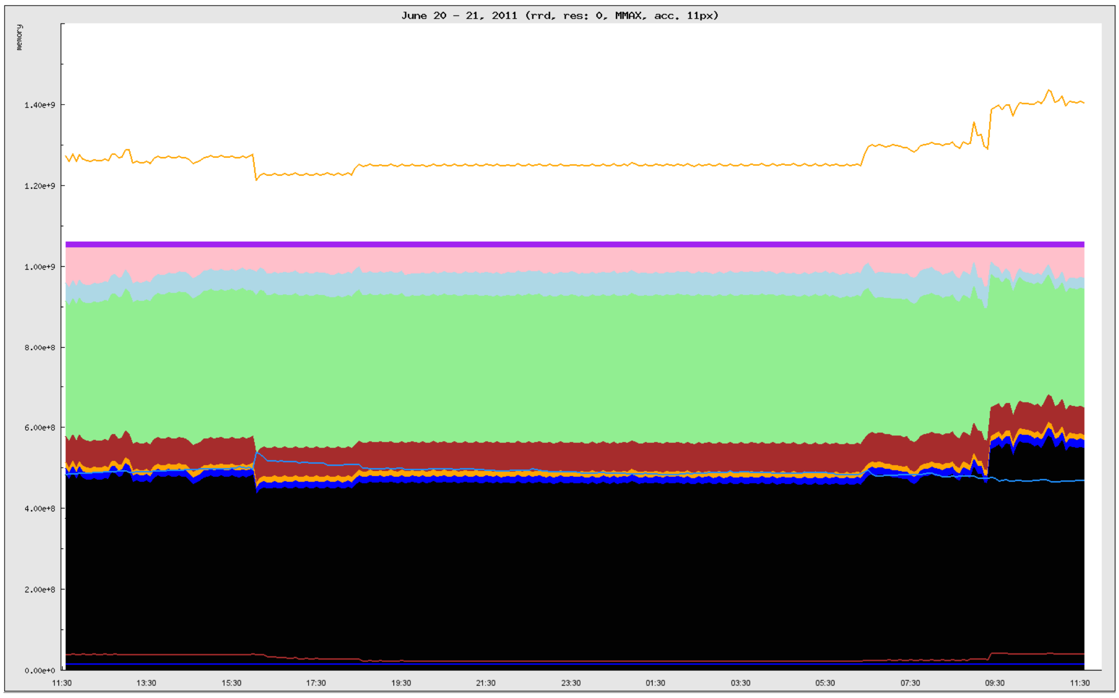


FIGURE 11. Munin generated data on system memory usage graphed using ADEI

3.3 Module for Visualizing Multidimensional Data

The presentation of multidimensional data is not possible in the current version of the ADEI since the drawing modes used to present the data only account for one variable on every specific moment in time. Many forms of data cannot be accurately presented with this type of graphing. The new module for visualizing multidimensional data was created to address this problem.

3.3.1 Requirements

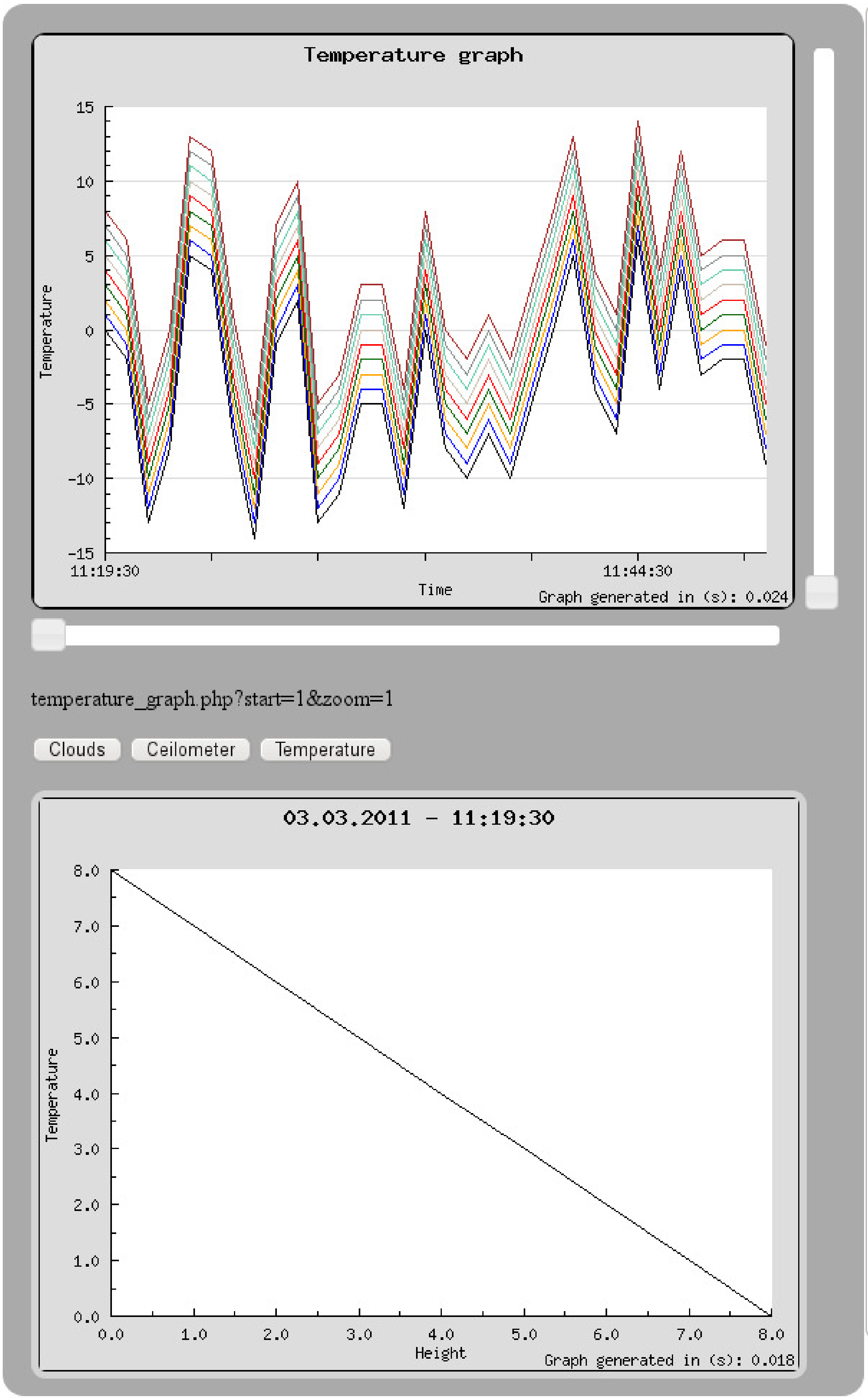
Since the data to be presented with the module may vary greatly, the aim of its development was to make a foundation for multidimensional data presentation that could be easily extended to support the handling of new data types that need a different approach than those already implemented in the services used by the module.

There were three different types of multidimensional data that needed to be considered when developing the module. The wide angle pictures taken by a cloud camera at the KIT weather tower, the two-dimensional array of data provided by a ceilometer used to determine the heights of cloud layers and a dataset from temperature sensors at different heights of the KIT weather tower.

3.3.2 Implementation

At first a prototype of the new module was created using HTML and JavaScript to test different approaches to visualize multidimensional data. This way the early development could focus on handling the data and how to present it efficiently without having to be concerned with the specific workings of the ADEI. The services that dealt with creating images for the prototype from the data provided were done using PHP. These services utilize the same JPGraph PHP graphing library as the ADEI's DRAW class. This library is used to set the axes that the generated image is then fitted to.

The prototype's aim was to create a template that had a place to generate a graph displaying the multidimensional data and provided the necessary user controls to move through the data. The layout of the prototype showing temperature data can be seen in Figure 12. The sliders on the side and below of the upper graph are used to adjust the zoom level of the graph and scroll through measurement times. The user can switch between different modes of visualization with the buttons found between the two graphs and the bottom graph changes represent the data selected by the user by mouse clicking the upper graph.



temperature_graph.php?start=1&zoom=1

FIGURE 12. The layout of the prototype module

The first type of multidimensional data to be visualized using the module were images taken by the KIT weather tower cloud camera. These wide angle shots display the whole sky above the tower, with the horizon displayed as a circle on the edge of the image. The approach taken to visualizing these images over time was to go through the images stored on the server by the camera and crop a small slice from the middle of each image and then generate an image from the slices to display the changes in cloud density and daylight over time.

While the primary image created this way provides the user with a general idea on the look of the sky on a given time, a secondary image displaying the complete image from a time chosen by the user is shown below the primary image.

Another type of data needed to be dealt with was a two-dimensional array of figures describing the amount of laser light backscatter from clouds as measured by the weather tower ceilometer. The array contains the time of a measurement as a UNIX timestamp as the key and an array of backscatter values as the value for each key. These arrays can be found stored in a MySQL database as BLOB data.

The approach taken to visualize this data was to create an image in which every pixel describes a time/height pair and the color of the pixel describes the thickness of the cloud layers based on the backscatter values. Using this approach to create the image enables the user to get the general idea on how high and how thick the cloud layers are on a specific time. This can be seen in Figure 13. Once again, the secondary image of the module is used to show a more accurate display of the backscatter values from a time selected by the user as seen in Figure 14.

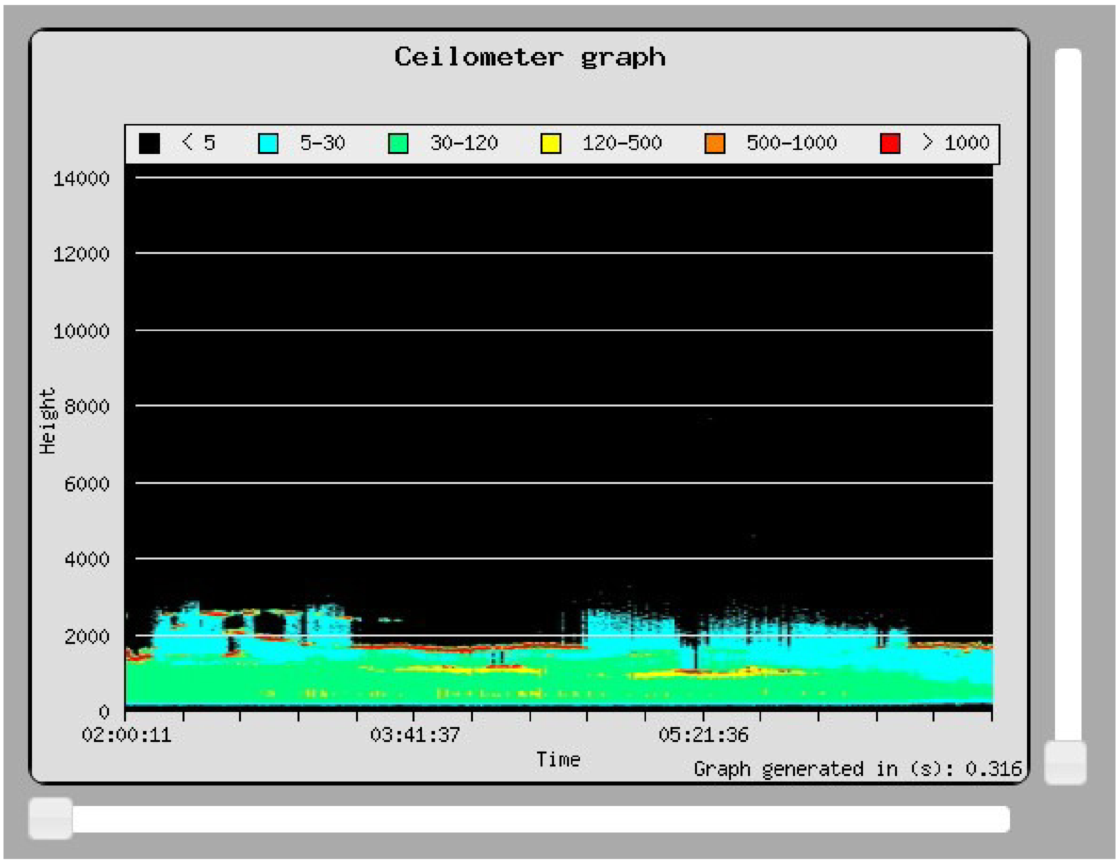


FIGURE 13. Ceilometer main graph

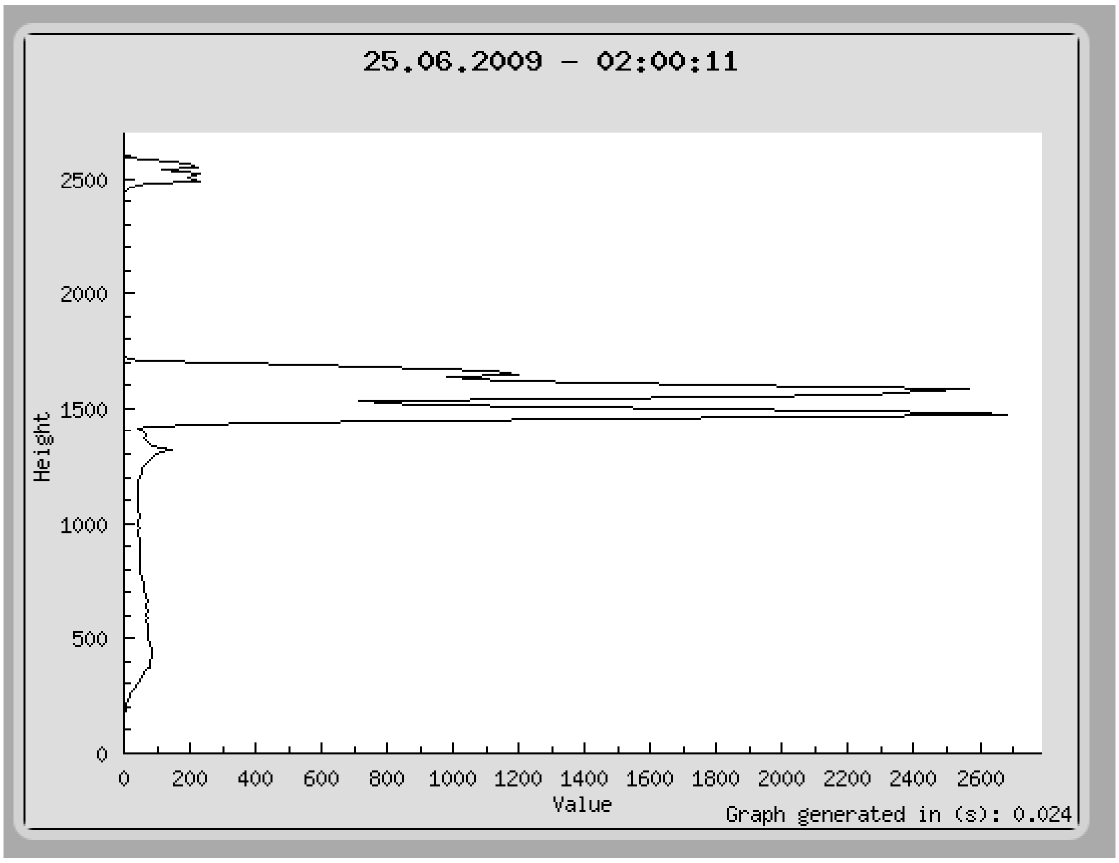


FIGURE 14. Ceilometer secondary graph

The third and final type of data required to be visualized by the new module was the air temperatures measured by sensors on different heights of the KIT weather tower. However, since the at the time, only one of the sensors was online and outputting real data, the modules feature used randomized data for the temperatures as can be seen on Figure 12. For simplicity the simulated values assume a linear relation between height and temperature.

The module was integrated into the ADEI by using the standard way of adding a module described in the ADEI Project's website (The ADEI Project 2012). This included the creation of an XSL template to create the base of the module page, the creation of the PHP services returning the XML for the template and the JavaScript code to handle user interaction with the module.

3.3.3 Conclusions

All three of the ways of displaying multidimensional data were successful and achieved a reasonable efficiency and representation accuracy. The display of cloud camera images was successfully integrated into the ADEI running on the test server and it offers a solid foundation to further develop the visualization of multidimensional data in the ADEI. A UML model describing the module and its current services can be seen in Figure 15.

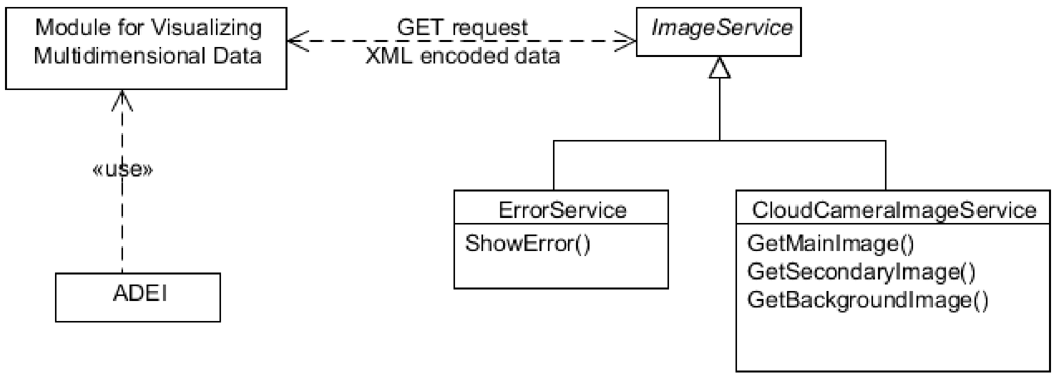


FIGURE 15. UML showing the module and the currently implemented services

However, the module is still a rough draft of what it could be and will likely require further development and polishing before it can be considered a stable part of the existing system.

4 CONCLUSIONS

While I was working on my thesis the ADEI was expanded to include features for resource monitoring and the ability to present multidimensional data. The usage of HTML5 in the future development of the ADEI was also researched.

The performance monitoring features implemented are already in use to track resources on KIT's Linux servers supporting the KATRIN and KITCube experiments. The work on integrating multidimensional data visualization on the KITCube ADEI is also in progress. The research on HTML5 has shown promise in upgrading the ADEI to HTML5 and it has been planned that this work will be continued.

Working on the thesis went smoothly after getting familiar with the system, which of course took its time considering the size and complexity of the ADEI system compared to what I had worked with before. Regarding this, some sort of a tutorial on adding new modules to the ADEI system would have been a good way to start examining the system as, for example, the XSLT approach of creating views was not familiar to me beforehand and so it took a rather long time to sink in.

Overall working on my thesis at the IPE was a pleasant experience that widened my own views of the world and developed both my professional and language skills. It would not be far off to say that my time in Germany was the most entertaining half a year of my life so far.

REFERENCES

The ADEI Project 2012 *A publically visible implementation of the ADEI supporting the KATRIN project at KIT* [online] [Accessed 21 March 2012] Available from: http://katrin.kit.edu/adei/

The ADEI Project 2012 *The Advanced Data Extraction Infrastructure project's Trac page* [online] [Accessed 20 March 2012] Available from: http://adei.info

Chilingaryan S., Beglarian A., Kopmann A. and Vöcking S. 2010 *Advanced data extraction infrastructure: Web based system for management of time series data* [online] [Accessed 21 March 2012] Available from: http://iopscience.iop.org/1742-6596/219/4/042034

Chilingaryan S., Kopmann A. 2011 *The future of the ADEI, PowerPoint Presentation*

The Munin Project 2012 *Documentation for the resource monitoring tool Munin* [online] [Accessed 21 March 2012] Available from: http://munin-monitoring.org/

Oetiker T. 2011 *Documentation for the RRDTool used to read data from RRD-files* [online] [Accessed 21 March 2012] Available from: http://oss.oetiker.ch/rrdtool/

RGraph 2012 *Documentation for the RGraph HTML5 JavaScript graphing library* [online] [Accessed 20 March 2012] Available from: http://www.rgraph.net

W3C 2012 *World Wide Web Consortium's specifications on HTML5* [online] [Accessed 21 March 2012] Available from: http://dev.w3.org/html5/spec/spec.html

APPENDICES

Appendix 1, Program code of the ADEI RRDREADER extension's iterator and reader classes

```
<?php

//this class is used to iterate Munin data
class MUNINData implements Iterator {

 var $period;//the time difference of two datapoints in seconds
 var $from, $to;//timestamps for start and end of iterated data
 var $resample, $nextsample;
 var $pos;//current x-position (timestamp)
 var $opts;// a handle to readers configuration options

 var $items;//rrds in the data (mask)

 var $start;

 const DEFAULT_START = "May 18, 2011";

 public function __construct(RRDReader &$reader, OPTIONS &$opts, &$items, INTERVAL
&$ivl, $resample) {

    $this->opts = $opts;

    $start = NULL;
    $end = NULL;
    $step = NULL;

    //fetch to find out the start, end and step of the rrds(they should all have the
same ones)
    //if not, then the start end will be the values that every rrd has data at
    //if steps are different this throws an exception
    for($i = 0 ; $i < count($items) ; $i++)
    {
             $file = $this->opts->Get('rrd_folder') . $this->opts->Get('rrd_prefix')
. $items[$i]['uid'] . ".rrd";

             $fetch_opts = array("AVERAGE");
             $fetch = rrd_fetch($file, $fetch_opts, count($fetch_opts));
             if($start === NULL) $start = $fetch['start'];
             if($end === NULL) $end = $fetch['end'];

             if($start < $fetch['start'])
             {
                 $start = $fetch['start'];
             }
             if($end > $fetch['end'])
             {
                 $end = $fetch['end'];
             }
             if($step === NULL)
             {
                 $step = $fetch['step'];
             }
             else if($step !== $fetch['step'])
             {
                 throw new ADEIException(translate("File %s has an irregular data
interval!", $file));
             }
    }

    $this->period = $step;

    $this->resample = $resample;
    $from =  $ivl->GetWindowStart();

    $ifrom = ceil($from);//get start point as integer
    $rem = $ifrom % $this->period;//checks whether start point as integer is a
multiple of period
    if ($rem) $this->from = $ifrom + ($this->period - $rem);//if not, add to start
period - the remnants of the division
    else $this->from = $ifrom;//if yes, from is ifrom



    $this->to = $ivl->GetWindowEnd();

    if($this->from < $start)
    {
             $this->from = $start;
    }

    $this->pos = $this->from;

    $limit = $ivl->GetItemLimit();//get interval's item limit, if any (defines how
many datapoints to show and adjusts to or from based on that?)

    if ($limit) {
             if ($limit > 0) {
                 $to = $this->from + $this->period * $limit;
                 if ($to < $this->to) $this->to = $to;
             } else {
                 $from = $this->to + $this->period * $limit;
                 if ($from > $this->from) $this->from = $from;
             }
    }

    $this->items = &$items;//items to show (rra:s in this case)

    $limit = $opts->GetDateLimit($this::DEFAULT_START, time());//gets the window
limits (as in, the first and last possible timestamp values)

    $this->start = $limit[0];
 }

 function doResample() {
    for ($next = $this->pos + $this->period;(($next < $this->to)&&($next < $this-
>nextsample));$next += $this->period)
             $this->pos = $next;

    $this->nextsample += $this->resample;
    if ($this->nextsample < $this->pos) {
             $add = ceil(($this->pos - $this->nextsample) / $this->resample);
             $this->nextsample += $add * $this->resample;
    }
 }

 function rewind() {
    $this->pos = $this->from;

    if ($this->resample) {
             $this->nextsample = $this->resample*ceil($this->pos / $this->resample);
             $this->doResample();
    }
 }

 function current() {

    $values = array();

    foreach($this->items as $item)
    {
             $file = $this->opts->Get('rrd_folder') . $this->opts->Get('rrd_prefix')
. $item['uid'] . ".rrd";//construct filename

             $fetch_opts = array("AVERAGE", "--start", ($this->pos - $this->period),
"--end", ($this->pos));

             $fetch = rrd_fetch($file, $fetch_opts, count($fetch_opts));//fetch data
from file

             foreach($fetch['data'] as $index => $data)
             {
                 if(($fetch['start']   +   ($index   +   1)   *   $fetch['step'])   ===
intval($this->pos))//check if the data is from current time
                 {
                          if(strval($data)  ==  "NAN")  $data  =  NULL;//if  the  data
value is NAN, convert it to NULL
                          array_push($values, $data);
                 }
             }
    }

    return $values;
 }

 function key() {//this returns a timestamp for the current position
    $res = $this->pos;
    if (is_float($res)) return sprintf("%.8f", $res);
    return $res;
 }

 function next() {//moves position to next timestamp
    $this->pos += $this->period;

    if ($this->resample) $this->doResample();
 }

 function valid() {//checks whether this datapoint is valid  ( timestamp is smaller
than the last one )
    return ($this->pos <= $this->to);
 }
}

//this class is used to iterate general RRD data (not Munin)
class RRDData implements Iterator {

 var $file; //filename of the rrd-file that is iterated
 var $period;//the time difference of two datapoints in seconds
 var $from, $to;//timestamps for start and end of iterated data
 var $resample, $nextsample;
 var $pos;//current x-position (timestamp)

 var $items;//items in the data (mask)

 var $start;//what's this for?
 var $info;//contains information about the rrd

 const DEFAULT_START = "May 18, 2011";

 public function __construct(RRDReader &$reader, OPTIONS &$opts, &$items, INTERVAL
&$ivl, $resample) {

    $data_start = $opts->Get('data_start');
    $data_start = date("F j, Y", $data_start);

    $this->period = $opts->Get('step');
    $this->file = $opts->Get('rrd_folder') . $opts->Get('rrd_prefix') . $opts-
>Get('file') . ".rrd";

    $this->info = $reader::GetRRDInfo($opts->Get('file'), $reader); //get
information on the rrd and its archives


    $this->resample = $resample;

    $from =  $ivl->GetWindowStart();

    $ifrom = ceil($from);//get start point as integer
    $rem = $ifrom % $this->period;//checks whether start point as integer is a
multiple of period
    if ($rem) $this->from = $ifrom + ($this->period - $rem);//if not, add to start
period - the remnants of the division
    else $this->from = $ifrom;//if yes, from is ifrom


    $this->pos = ($this->from);

    $this->to = $ivl->GetWindowEnd();

    $limit = $ivl->GetItemLimit();//get interval's item limit, if any (defines how
many datapoints to show and adjusts to or from based on that?)

    if ($limit) {
             if ($limit > 0) {
                 $to = $this->from + $this->period * $limit;
                 if ($to < $this->to) $this->to = $to;
             } else {
                 $from = $this->to + $this->period * $limit;
                 if ($from > $this->from) $this->from = $from;
             }
    }

    $this->items = &$items;//items to show (rra:s in this case)

    //this sets the caching starting point based on the longest rra:s datapoints
    $limit = $opts->GetDateLimit($data_start, time());
    //$limit = $opts->GetDateLimit($this::DEFAULT_START, time());//gets the window
limits (as in, the first and last possible timestamp values)

    $this->start = $limit[0];
 }

 function doResample() {
    for ($next = $this->pos + $this->period;(($next < $this->to)&&($next < $this-
>nextsample));$next += $this->period)
             $this->pos = $next;

    $this->nextsample += $this->resample;
    if ($this->nextsample < $this->pos) {
             $add = ceil(($this->pos - $this->nextsample) / $this->resample);
             $this->nextsample += $add * $this->resample;
    }
 }

 function rewind() {
    $this->pos = $this->from;

    if ($this->resample) {
             $this->nextsample = $this->resample*ceil($this->pos / $this->resample);
             $this->doResample();
    }
 }

 function current() {
    $current_timestamp = $this->pos;

    $info = $this->info;//info about the rrd-file
    $items = $this->items;//rra:s to show
    $from = $this->from;
    $to = $this->to;
    $file = $this->file;
    $period = $this->period;

    $values = array();//return array

    $item_keys = array();

    //get an array of item ids that should be in the return array
    foreach($items as $array)
    {
             foreach($array as $key => $value)
             {
                 if($key == "id")
                 {
                          array_push($item_keys, $value);
                 }
             }
    }

    foreach($info["rra"] as $key => $rra)//go through all rra:s
    {

             $datafound = false; //flag set to true if valid data is found on
current timestamp

             if(array_search($key, $item_keys) === false)//if current rra is not in
the iterators item list, it is skipped
             {
                 continue;
             }

             $step = $info["info"]["step"] * $rra["pdp_per_row"]; //time difference
of the current rra's datapoints in seconds
             $span = $step * $rra["rows"]; //time difference of first and last
datapoint of the current rra

             //fetch options set in a way that it only returns data close to the
current timestamp
             $fetch_opts = array($rra["cf"], "-r", $step, "--start", ($this->pos -
($step)), "--end", ($this->pos + ($step)));

             $fetch = rrd_fetch($file, $fetch_opts, count($fetch_opts));//fetch rra
data

             if($fetch["step"] > $step)//sometimes the fetch returns data at bigger
intervals than the current rra should, this fixes that
             {
                 $fetch_opts = array($rra["cf"], "-r", $step, "--end",  ("N - " .
($span - 2 * $step)), "--start", ("N - " . $span));
                 $fetch = rrd_fetch($file, $fetch_opts, count($fetch_opts));
             }

             $count = count($fetch["data"]);//number of datapoints returned by fetch

             //this keeps track of that no rra returns values from before its
beginning, needs a bit more work
             $first_timestamp = $to - ($info["info"]["step"] * $rra["pdp_per_row"] *
$rra["rows"]);

             if($first_timestamp > $current_timestamp)//if the current timestamp is
not valid for the current rra we skip it
```

```
            {
                array_push($values, NULL);
                continue;
            }
            //go through all the data returned by fetch
            foreach($fetch["data"] as $index => $data)
            {

                if(strval($data) == "NAN") $data = NULL; //if there's no data, data
is a float type with value NAN,
                //this makes no sense so we convert it to NULL

                //construct  timestamp  for  the  datapoint  based  on  the  datapoint
index and the end timestamp returned by fetch
                $timestamp   =   ($fetch["end"]   -   (($count-($index   +   2))   *
$fetch["step"]));//+2 because rrd

                //if the current datapoints timestamp is valid
                if($timestamp >= $first_timestamp && $timestamp <= $to)
                {
                        //if  the  current  datapoint  is  the  one  we  are  interested
in
                        if($timestamp == $current_timestamp)
                        {
                            $value = $data;
                            $datafound = true;//data found for current timestamp,
                            //this  makes  the  $value  stay  the  same  while  going
through the rest of the rra:s
                        }
                }
                else if(!$datafound)//if no data is found, $value is NULL
                {
                        $value = NULL;
                }

            }
            if(!$datafound && $value !== NULL)//in some cases if no data was found
the value still wasn't NULL, this fixes that
            {
                $value = NULL;
            }

            if($value !== NULL)//if there is a valid value, insert it into return
array
            {
                array_push($values, $value);
            }
            else
            {
                //if there was no valid value for this timestamp in the rra,
                //(eg. if the rra data interval is 30 mins it only has values on
every sixth period if the iterator period is 5 min)
                //this will fetch the previous valid value from the current rra
                $fetch_opts    =    array($rra["cf"],    "-r",    $step,    "--start",
((floor($this->pos  /  $step)  *  $step)  -  ($step)),  "--end",  ((floor($this->pos  /
$step) * $step) + ($step)));

                $fetch = rrd_fetch($file, $fetch_opts, count($fetch_opts));

                foreach($fetch["data"] as $index => $data)
                {
                        if(strval($data) == "NAN") $data = NULL;
                        $stamp   =   ($fetch["end"]   -   (($count-($index   +   2))   *
$fetch["step"]));

                        if($stamp == (floor($this->pos / $step)) * $step)
                        {
                            $value = $data;
                            }
                    }
                    array_push($values, $value);
                }
        }

        return $values;
 }

 function key() {//this returns a timestamp for the current position
        $res = $this->pos;
        if (is_float($res)) return sprintf("%.8f", $res);
        return $res;
 }

 function next() {//moves position to next timestamp
        $this->pos += $this->period;

        if ($this->resample) $this->doResample();
 }

 function valid() {//checks whether this datapoint is valid  ( timestamp is smaller
than the last one )
        return ($this->pos <= $this->to);
 }

}

class RRDReader extends READER {
 var $cache;

 var $items;//RRDs
 var $groups;//RRD-groups for Munin or RRDs for non Munin
 var $munin_datafile;
 var $axes;

 function __construct(&$props) {
        parent::__construct($props);

        if ($this->dbname) {
                $folder = $this->req->GetGroupOptions()->Get('rrd_folder');
                if($folder)//check if there's a specified folder for RRDs in config
                {
                    $this->items = $this::ReadRRDs();//items holds all the RRD files

                    if(!empty($this->items))
                    {
                            if($this->req->GetGroupOptions()-
>Get('munin_datafile'))//check if datafile is specified
                        {
                                $this->munin_datafile          =          $this->req-
>GetGroupOptions()->Get('munin_datafile');
                                $this->groups  =  $this::GetMuninGroups();//read  Munin
file groups from the specified Munin datafile
                                $this->axes = $this::CreateAxes();
                            }
                            else
                            {
                                $this->munin_datafile = false;

                                if($this->items)//if rrd-files were found
                                {
                                        $this->groups = array();

                                        foreach  ($this->items  as  $gid)//go  through
the files
                                        {
                                            if($names[$gid])//if  the  file  had  a
human readable name, give the loggroup a name
                                            {
                                                    $name = $names[$gid];
                                            }
                                            else//else  the  loggroups  name  will  be
it's group identifier (filename)
                                            {
                                                    $name = false;
                                            }
                                            $this->groups[$gid] = array(
                                                    'name' => $name
                                            );
                                        }
                                }
                            }
                    }
                    else
                    {
                            throw new ADEIException(translate("No files found in the
specified folder %s", $folder));
                    }
                }
                else
                {
                    throw  new  ADEIException(translate("No  RRD  folder  specified  in
config."));
                }
        }

 }

 function GidToFilename($gid, $reader)//change to change item id to filename
 {
    $opts = $reader->req->GetGroupOptions();

    $return = $opts->Get('rrd_folder') . $opts->Get('rrd_prefix') . $gid . ".rrd";

    return $return;
 }

//this function reads the RRD filenames from the folder specified in the rrd server
configuration
//and returns them in an array
 function ReadRRDs()
 {
    $opts = $this->req->GetGroupOptions();

    $folder = $opts->Get('rrd_folder');//get the RRD folder specified in the options
    $prefix = $opts->Get('rrd_prefix');//same for RRD files prefix
    if($opts->Get('rrd_file_mask'))//check if a regexp is defined for filtering the
RRD files
    {
            $filemask = $opts->Get('rrd_file_mask');
    }

    if($folder)
    {
            $rrd_array = array();//return array

            if(chdir($folder))//change dir to specified folder
            {
                if($filehandle = opendir($folder))//open folder
                {
                        while(($filename = readdir($filehandle)) !== false)//read
files in folder
                        {
                              if(preg_match("/\.rrd$/", $filename) != 0)//if file
is .rrd file
                              {
                                       $filename = str_replace(".rrd", "",
$filename);//strip the rrd suffix from the filename
                                       $filename = str_replace($prefix, "",
$filename);//strip the rrd prefix from the filename
                                       if($filemask)
                                       {
                                           if(preg_match($filemask, $filename) !=
0)//if regexp is defined in config, only matching RRDs will be shown
                                           {
                                                    array_push($rrd_array,
$filename);
                                           }
                                       }
                                       else
                                       {
                                           array_push($rrd_array, $filename);
                                       }
                              }
                          }
                          closedir($filehandle);//close folder
                 }
             }

             asort($rrd_array);//sort the array by filename

             return $rrd_array;
    }
    else//if there's no folder specified in config, throw an error
    {
             throw new ADEIException(translate("No RRD folder specified in
config."));
    }
 }

//this function gets information about an rrd-file
 function GetRRDInfo($gid, RRDReader $reader)//reader is passed here because
apparently the call to this function from RRDDATA's constructor prevents the use of
$this pointer here
 {
    $file = $reader::GidToFilename($gid, $reader);//construct the complete filename
out of the group id

    $info = shell_exec("rrdtool info ". $file);//get info of the rrd into a string

    $array = explode("\n", $info);//make the string into an array
    $ds_array = array();//array with info about the rrd:s data source
    $rra_array = array();//array with info about the rrd:s archives(rra)
    $info_array = array();//general info about the rrd
    $return_array = array();//an array that will contain the other arrays

    foreach($array as $data)//go through the data returned by rrdtool info and sort
the rows into correct arrays
    {
             $key = substr($data, 0, strpos($data , " = "));
             $value = str_replace("\"", "", (substr($data, strpos($data, " = ") +
3)));
             if(preg_match("/^ds\[/", $key))
             {
                 $ds_key = substr($key, (strpos($key, "[") + 1), (strpos($key, "]")
- 3));
                 $ds_array_key = substr($key, (strpos($key, ".") + 1));
                 $ds_array_value = $value;
                 $ds_array[$ds_key][$ds_array_key] = $ds_array_value;
             }
             else if(preg_match("/^rra\[/", $key))
             {
                 $rra_key = substr($key, (strpos($key, "[") + 1), (strpos($key, "]")
- 4));
                 $rra_array_key = substr($key, strpos($key, ".") + 1);
                 $rra_array_value = $value;
                 $rra_array[$rra_key][$rra_array_key] = $rra_array_value;
             }
             else if($key != "0")
             {
                 $info_array[$key] = $value;
             }
    }
    $info_array["start"] = "";//start and end are left empty here and filled in with
rrd_fetch in GetGroupInfo()
    $info_array["end"]   =   "";//they   are   only   filled   in   GetGroupInfo()   if
REQUEST::NEED_INFO flag is given in the function call


    $return_array["info"] = $info_array;
    $return_array["ds"] = $ds_array;
    $return_array["rra"] = $rra_array;

    return $return_array;
 }

 //this function reads the Munin groups from the  specified datafile and returns
them in an array containing their id and name
 function GetMuninGroups()
 {

    $opts = $this->req->GetGroupOptions();
    $datafile = $opts->Get('munin_datafile');//get the specified datafile name

    $info = file_get_contents($datafile);//read datafile content to string

    $info = explode("\n", $info);//make the string into an array of rows

    //this strips the setup dependant prefix from the datafile rows
    //(first row has version number and no prefix so we skip that)
    foreach($info as $key => &$line)
    {
             if($key !== 0)
             {
                 $line = str_replace(substr($line, 0, strpos($line, ":") + 1), "",
$line);
             }
    }

    //find all datafile rows that define a graph title and get the group ids and
names and axis info from those rows
    foreach($info as $row)
    {

             if(substr_count($row, ".graph_title") !== 0)
             {
                 $group_key = substr($row, 0, strpos($row, ".graph"));
                 $group_value =  str_replace(substr($row, 0, (strpos($row, " ") +
1)), "", $row);

                 $group_array[$group_key] = array(
                                                                         "gid"      =>
$group_key,
                                                                         "name"     =>
$group_value
                                                                     );

             }
             if(substr_count($row, ".graph_args") !== 0 && strstr($row, $group_key))
             {
                 $args = explode(" ", substr($row, strpos($row, " ")));

                 $log = NULL;
            $min = NULL;
                 $max = NULL;

                 if(array_search("--logarithmic", $args)) $log = true;
                 if(array_search("--lower_limit",       $args))       $min       =
$args[array_search("--lower_limit", $args) + 1];
                 if(array_search("--upper_limit",       $args))       $max       =
$args[array_search("--upper_limit", $args) + 1];
             }

             if(substr_count($row,   ".graph_vlabel")   !==   0   &&   strstr($row,
$group_key))
             {
                 $label = str_replace(substr($row, 0, (strpos($row, " ") + 1)), "",
$row);

                 $label = str_replace("\${graph_period}", "second", $label);

                 if($log)
                 {
                         $mode = "LOGARITHMIC";
                 }
                 else
                 {
                         $mode = "STANDARD";
                 }
                 if($min||$max)
                 {
                         $range = array($min, $max);
                 }
                 else
                 {
                         $range = false;
                 }

                 $opts->options['axes_table'][$group_key] = array(
                                                     "axis_units" => false,
                                                     "axis_name" => $label,
                                                     "axis_mode" => $mode,
                                                     "axis_range" => $range
                                                     );
             }
    }

    return $group_array;
 }

 function ParseSeconds($seconds)//parses seconds into more readable formats (because
no one knows how much eg. 3338000 seconds is)
 {
    if((($seconds % (3600 * 24)) == 0) && (($seconds / (3600 / 24) >= 1)))
    {
             $return = ($seconds / (3600 * 24)) . "d";
    }
    else if((($seconds % 3600) == 0) && (($seconds / 3600) >= 1))
    {
             $return = ($seconds / 3600) . "h";
    }
    else if((($seconds % 60) == 0) && (($seconds / 60) >= 1))
    {
             $return = ($seconds / 60) . "min";
    }
    else
    {
             $return = $seconds . "s";
    }
    return $return;
 }

 //function to simplify calling rrd_fetch, mainly because the complete file need to
be constructed every time fetch is called
 function Fetch($gid, $fetch_opts)
 {
    $file = $this::GidToFilename($gid, $this);

    $return = rrd_fetch($file, $fetch_opts, count($fetch_opts));

    return $return;
 }

 function CreateAxes($flags = 0) {
    if ($this->axes) return $this->axes;

    $axes_table = $this->req->GetGroupOptions()->options['axes_table'];

    if ($axes_table) {

             $this->axes  =  new  MUNINAxes($this->req,  $axes_table);//try  to  change
this to GRAPHAxes and then something...

             return $this->axes;
    } else {
             return parent::CreateAxes($flags);
    }
 }


 function GetGroupInfo(LOGGROUP $grp = NULL, $flags = 0) {

    $groups = array();//array for the groups to be returned

    if($this->munin_datafile)//Munin datafile specified
    {
             foreach  ($this->groups  as  $gid  =>  &$group)//if  there  are  groups,  go
through them
             {
                 if  (($grp)&&(strcmp($grp->gid,  $gid)))  continue;//if  a  group  is
specified and existing group


                          //has same group id then skip this group

                 if  ($group['name'])  $name  =  $group['name'];//if  this  group  has  a
name, name = group name
                 else $name = false;//else name = false

                 if  ((!$name)||($flags&REQUEST::NEED_INFO))  {//if  there  was  no  name
and there's a NEED_INFO flag
                          if ($grp) {//if group is specified
                              $grtest = $grp;
                              $opts = $this->opts;
                          }  else  {//no  group  is  specified  (easier  to  just  throw
error here since there can't really be a default .rrd-file)?
                              $ginfo   =   array("db_group"   =>   $gid);//create   new
groupinfo array
                              $grtest   =   $this->CreateGroup($ginfo);//create   new
group
                              $opts  =  $this->req->GetGroupOptions($grtest);//create
new options for group
                          }
```

```
                        if (!$name) {//if name = false
                            $name  =  $opts->Get('name',  $gid);//get  name  from
options, if there's no name use group id
                        }
                }

                $groups[$gid] = array(//set values into group array at index group
id
                        'gid' => $gid,
                        'name' => $name
                );

                if($flags&REQUEST::NEED_COUNT)//if there's a NEED_COUNT flag
                {
                        $items = $this::GetItemList($grp);//change to getitemlist
                        $rrd_fetch_opts = array("AVERAGE");
                        $fetch = $this::Fetch($items[0]['uid'], $rrd_fetch_opts);

                        $record_num  =  (  (  $fetch['end']  -  $fetch['start']  )  /
$fetch['step'] );

                        $groups[$gid]['records'] = $record_num;
                }

                if ($flags&REQUEST::NEED_INFO) {//if there's a NEED_INFO flag

                        $items = $this::GetItemList($grp);//change to getitemlist
                        $rrd_fetch_opts = array("AVERAGE");
                        $fetch = $this::Fetch($items[0]['uid'], $rrd_fetch_opts);
                        $groups[$gid]['first'] = $fetch['start'];
                        $groups[$gid]['last'] = $fetch['end'];

                        if    ($flags&REQUEST::NEED_COUNT)    {//if    there's    a
NEED_COUNT flag
                            $record_num = ( ( $fetch['end'] - $fetch['start'] ) /
$fetch['step'] );//calculate number of records

                            $groups[$gid]['records'] = $record_num;
                        }

                        if   ($flags&REQUEST::NEED_ITEMINFO)   {//if   there's   a
NEED_ITEMINFO flag
                            $groups[$gid]['items']            =            $this-
>GetItemList($grp);//get itemlist for specified group
                        }
                }
            }//went through existing groups
    }
    else//no Munin datafile
    {
            foreach  ($this->groups  as  $gid  =>  &$group)//if  there  are  groups,  go
through them
            {
                if  (($grp)&&(strcmp($grp->gid,  $gid)))  continue;//if  a  group  is
specified and existing group has same group id then skip this group

                if  ($group['name'])  $name  =  $group['name'];//if  this  group  has  a
name, name = group name
                else $name = false;//else name = false

                if  ((!$name)||($flags&REQUEST::NEED_INFO))  {//if there was no name
and there's a NEED_INFO flag
                        if ($grp) {//if group is specified
                            $grtest = $grp;
                            $opts = $this->opts;
                        }  else  {//no  group  is  specified  (easier  to  just  throw
error here since there can't really be a default .rrd-file)?
```

```
                              $ginfo  =  array("db_group"  =>  $gid);//create  new
groupinfo array
                              $grtest  =  $this->CreateGroup($ginfo);//create  new
group
                              $opts = $this->req->GetGroupOptions($grtest);//create
new options for group
                          }

                          if (!$name) {//if name = false
                              $name  =  $opts->Get('name',  $gid);//get  name  from
options, if there's no name use group id
                          }
                 }

                 $groups[$gid] = array(//set values into group array at index group
id
                          'gid' => $gid,
                          'name' => $name
                 );

                 if($flags&REQUEST::NEED_COUNT)//if there's a NEED_COUNT flag
                 {
                          $rrd_fetch_opts = array("AVERAGE");
                          $fetch = $this::Fetch($gid, $rrd_fetch_opts);

                          $record_num  =  (  (  $fetch['end']  -  $fetch['start']  )  /
$fetch['step'] );

                          $groups[$gid]['records'] = $record_num;
                 }

                 if ($flags&REQUEST::NEED_INFO) {//if there's a NEED_INFO flag

                          $rrd_fetch_opts = array("AVERAGE");
                          $fetch = $this::Fetch($gid, $rrd_fetch_opts);
                          $groups[$gid]['first'] = $fetch['start'];
                          $groups[$gid]['last'] = $fetch['end'];

                          if    ($flags&REQUEST::NEED_COUNT)    {//if    there's    a
NEED_COUNT flag
                              $record_num = ( ( $fetch['end'] - $fetch['start'] ) /
$fetch['step'] );//calculate number of records

                              $groups[$gid]['records'] = $record_num;
                          }

                          if   ($flags&REQUEST::NEED_ITEMINFO)   {//if   there's   a
NEED_ITEMINFO flag
                              $groups[$gid]['items']            =            $this-
>GetItemList($grtest);//get itemlist for specified group
                          }
                 }
             }//went through existing groups
    }
    if (($grp)&&(!$groups)) {//if group is specified and there are no defined groups
             throw  new  ADEIException(translate("Invalid  group  (%s)  is  specified",
$grp->gid));
    }

    return $grp?$groups[$grp->gid]:$groups;//if group was specified, return group in
groups table at index group id, else return groups table
 }

 function GetItemList(LOGGROUP $grp = NULL, MASK $mask = NULL, $flags = 0) {//gets
list of specified items in the specified group, returns an array containing item id,
uid and name(here maybe rra index,

             $grp = $this->CheckGroup($grp, $flags);//check if the group is valid
```

```
            if (!$mask) $mask = $this->CreateMask($grp, $info=NULL, $flags);//if 
there's no mask, create a new mask with default settings
            $opts = $this->req->GetGroupOptions();

            $gid = $grp->gid;

            $res = array();//return array

            if($this->munin_datafile)//if there's Munin datafile
            {
                $groups = $this->items;//contains all RRDs in the specified RRD 
folder

                $datafile = $this->munin_datafile;//get the specified datafile name

                $info = file_get_contents($datafile);//read datafile content to 
string

                $info = explode("\n", $info);//divide the string into an array of 
rows

                //this strips the setup dependant prefix from the datafile rows
                //(first row has version number and no prefix so we skip that)
                foreach($info as $key => &$line)
                {
                        if($key !== 0)
                        {
                            $line = str_replace(substr($line, 0, strpos($line, 
":") + 1), "", $line);
                        }
                }

                $items = array();//array for the items of this Munin group

                $item_ids = array();//array for identifiers of this groups items

                foreach($info as $row)
                {
                        if(strstr($row, ($gid . ".")) && strstr($row, ($gid . 
".graph_order")))//find the datafile row that defines the RRDs in this group
                        {
                            $array = explode(" ", substr($row, (strpos($row, " ") 
+ 1)));

                            foreach($array as $index => $value)//go through the 
RRDs of this group
                            {
                                    if($mask)
                                    {
                                        if(!$mask->Check($index))
                                        {
                                                continue;
                                        }
                                    }//if item key wasn't in the mask, skip it
                                    if(!empty($value))
                                    {
                                        array_push($item_ids, $value);
                                    }
                            }
                        }
                        $tmp = substr($row, strpos($row, ".")+1);

                        $string = substr($tmp, 0, strpos($tmp, "."));

                        if(strstr($row, ($gid . ".")) && !strstr($row , 
"graph_"))
                        {
                            if(array_search($string, $item_ids) === false)
```

```
                                {
                                        array_push($item_ids, $string);
                                }
                        }
                }

                foreach($item_ids as $index => $value)
                {
                        $string = (str_replace(".", "-", $gid) . "-" . $value);
//create a string to find the correct RRD for this item

                        foreach($groups as $rrd)
                    {
                                if(substr_count($rrd, $string) !== 0)
                                {
                                        $items[$index]['id'] = $index;
                                        $items[$index]['name']   =   $rrd;//this   is
replaced by info from datafile if info is found
                                        $items[$index]['uid']      =      $rrd;//the
identifier used to contruct the filename of the RRD
                                        $items[$index]['axis'] = $gid;
                                }
                        }
                }

                //go through all the items of the group and set their names as the
info row of the datafile if it's available
                foreach($item_ids as $index => $value)
                {
                        $draw_mode = NULL;
                        $string = "";
                        foreach($info as $row)
                        {
                                //for groups that list values per device we need to
separate the devices info values by using the devices id
                                if(strstr($row,  ($gid  .  "."))  &&  strstr($row,  "per
device"))
                                {
                                        $string = " ( " . $value . " ) ";
                                }
                                //if  info  is  found,  use  it  as  the  items  name,
otherwise the RRD identifier is used as the name
                                if(strstr($row,  ($gid  .  "."  .  $value  .  "."))  &&
substr_count($row, ".info") !== 0)
                                {
                                        $items[$index]['name']     =     substr($row,
(strpos($row, " ") + 1)) . $string;//if a name was found, it replaces the previously
set RRD identifier value
                                }
                                if(strstr($row,  ($gid  .  "."  .  $value  .  "."))  &&
substr_count($row, ".draw") !== 0)
                                {
                                        $draw_mode = substr($row, (strpos($row, " ")
+ 1));
                                }

                        }

                        if($draw_mode && array_key_exists('id', $items[$index]))
                        {
                                if(strstr($draw_mode, "LINE"))
                                {
                                        $draw_mode = "LINE";
                                }
                                $items[$index]['draw_mode'] = $draw_mode;
                        }
                        else if(array_key_exists('id', $items[$index]))
                        {
```

```
                        $items[$index]['draw_mode'] = "LINE";
                    }

                }
                $res = $items;
            }
            else//no Munin file
            {
                $info = $this::GetRRDInfo($gid, $this);

                foreach($info["rra"]  as  $key  =>  $rra)//go  through  all  af  the
archive info on current rrd-file and construct names for the archives
                {
                    if($mask)
                    {
                        if(!$mask->Check($key)) continue;
                    }//if rra key wasn't in the mask, skip it

                    $step = $info["info"]["step"];
                    $filename = $info["info"]["filename"] . "_";
                    $string = "RRA" . $key;
                    $string .= "_" . $rra["cf"];
                    $string .= "_SPAN";
                    $span = (( $step ) * $rra["pdp_per_row"] * $rra["rows"]);
                    $string .= $this::ParseSeconds($span);
                    $string .= "_INTERVAL";
                    $interval = ( $step ) * $rra["pdp_per_row"];
                    $string .= $this::ParseSeconds($interval);
                    $uid = str_replace("\"", "", ($filename . $string));
                    $res[$key]["id"] = $key;
                    $res[$key]["name"] = str_replace("\"", "", $string);
                    $res[$key]["uid"] = $uid;

                    $string = "";
                }
            }

    return $res;//returns an array containing uid, id and name of the item
 }

 function GetRawData(LOGGROUP $grp = NULL, $from = 0, $to = 0, DATAFilter $filter =
NULL, &$filter_data = NULL) {//get iterator for raw data of rrd:s archives

            $grp = $this->CheckGroup($grp);//checks if the group is valid

            $ivl  =  $this->CreateInterval($grp);//creates  a  data  interval  for  the
group

            if($filter)//if there is a filter, set filter options for the current
data
            {
                $mask = $filter->GetItemMask();
                $resample = $filter->GetSamplingRate();
                $limit = $filter->GetVectorsLimit();
                if   ($limit)   $ivl->SetItemLimit($limit);//if   item   limit   was
specified for the filter, then set item limit for interval

                if (isset($filter_data)) {
                    if ($mask) $filter_data['masked'] = true;
                    if ($resample) $filter_data['resampled'] = true;
                    if ($limit) $filter_data['limited'] = true;
                }
            }
        else//if there's no filter
            {
                $mask = NULL;
                $resample = 0;
                $limit = 0;
```

```
             }

             $ivl->Limit($from, $to);//set interval timelimit

             $opts = $this->req->GetGroupOptions($grp);//get options for group

             $items = $this::GetItemList($grp);//get list of items for current group

             if (($mask)&&(is_array($ids = $mask->GetIDs())))//if there's a mask and
you can get an array of ids from that mask
             {
                 $tmp = array();
                 foreach ($ids as $id)//go through all ids of the mask
                 {
                          if ($id >= sizeof($items))//if id goes out of bounds of
items array, throw an error
                     {
                              throw new ADEIException(translate("Invalid item mask
is supplied. The ID:%d refers non-existing item.", $id));
                          }
                          array_push($tmp, $items[$id]);// create a new items array
that contains just the items specified by the mask
                 }
                 $items = $tmp;

             }

    if($this->munin_datafile)//if Munin datafile is specified
    {
             return new MUNINData($this, $opts, $items, $ivl, $resample);// return
raw data
    }
    else//if no Munin datafile is specified
    {
             $gid = $grp->gid;

             $info = $this::GetRRDInfo($gid, $this);//get rrd info for the current
rrd-file

             $step = $info["info"]["step"];


             //get the minimum and maximum interval between two archive datapoints
in seconds
             $min_itv = 0;
             $max_itv = 0;
             $max_itv_index = -1;
             foreach($info["rra"] as $key => $rra)
             {
                 if($mask)
                 {
             if(!$mask->Check($key)) continue;
                 }

                 $itv = $info["info"]["step"] * $rra["pdp_per_row"];

                 if($itv > $max_itv)
                 {
                          $max_itv = $itv;
                     $max_itv_index = $key;
                 }
                 if($min_itv === 0)
                 {
                     $min_itv = $itv;
                 }
                 else if($itv < $min_itv)
                 {
                     $min_itv = $itv;
```

```
                    }
                }

                //passing a few parameters in options to RRDData
                $opts->options[count($opts->options)]['step'] = $min_itv;
        $opts->options[count($opts->options)]['file'] = $gid;
        $opts->options[count($opts->options)]['data_start'] = (time() - ($max_itv *
$info["rra"][$max_itv_index]["rows"]));

                return new RRDData($this, $opts, $items, $ivl, $resample);// return raw
data
    }
 }

 function HaveData(LOGGROUP $grp = NULL, $from = 0, $to = 0) {//checks whether file
has data or not
    $grp = $this->CheckGroup($grp);//checks if group is valid

    $ivl = $this->CreateInterval($grp);//creates interval
    $ivl->Limit($from, $to);//set limits for interval

    $period  =  $this->req->GetGroupOption('period',  $grp);//gets  period  of  the
records

    $from = $ivl->GetWindowStart();
    $to = $ivl->GetWindowEnd();

    if (($from - $to) > 2 * $period)
    {
             $return = true;
    }
    else
    {
             $return = false;
    }
    return $return;
 }
}

?>
```