Lappeenranta University of Technology
Faculty of Technology Management
Intelligent Computing Major

Master's Thesis

Joni Herttuainen

DESIGN AND IMPLEMENTATION OF A HIGH PERFORMANCE REAL-TIME DATA BROWSER FOR POWER CONVERTERS

Examiners: Prof. Lasse Lensu
Assoc. Prof. Arto Kaarna

Supervisors: Quentin King
Prof. Lasse Lensu

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Intelligent Computing Major

Joni Herttuainen

Design and implementation of a high performance real-time data browser for power converters

Master's Thesis

2017

68 pages, 13 figures, 5 tables, and 3 appendices.

Examiners: Prof. Lasse Lensu
Assoc. Prof. Arto Kaarna

Keywords: power converter, particle collider, Fourier transform, JavaScript, CERN, RBAC

The experts at the European Organization for Nuclear Research are constantly developing more accurate controllers for power converters to achieve the required magnetic fields in the particle accelerators. To aid in the development of these controllers, there is a need for tools that visualize the data of the converters. The earlier tools used in the organization are inadequate for the current requirements. The biggest issue has been the performance. A new tool, PowerSpy, is currently under development. This thesis studies the development of PowerSpy and the solutions made in the development process from the performance viewpoint. The performance-wise inefficient solutions are improved by designing better implementations to replace them and the improvement is measured. The results show that the performance of the application can be significantly improved by making correct decisions in the design and implementation of the software.

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Älykkään Laskennan Pääaine

Joni Herttuainen

Sähkövirran muuttajien reaaliaikaisen dataselaimen suunnittelu ja toteutus

Diplomityö

2017

68 sivua, 13 kuvaa, 5 taulukkoa ja 3 liitettä.

Tarkastajat: Prof. Lasse Lensu
Assoc. Prof. Arto Kaarna

Hakusanat: virtamuuttaja, hiukkaskiihdytin, Fourier-muunnos, JavaScript, CERN, RBAC
Keywords: power converter, particle collider, Fourier transform, JavaScript, CERN, RBAC

Euroopan hiukkastutkimuskeskuksen asiantuntijat kehittävät jatkuvasti tarkempia sähkövirran muuttajien ohjaimia halutunlaisten magneettikenttien aikaansaamiseksi hiukkaskiihdyttimissä. Ohjainten kehitystyön tueksi tarvitaan muuttajien datan visualisointityökaluja. Tutkimuskeskuksen aikaisemmat työkalut ovat riittämättömiä nykyisiin vaatimuksiin verrattuna. Suurimpana ongelmana on ollut työkalujen suorituskyky. Uusi työkalu - PowerSpy - on kuitenkin kehitteillä. Tämä diplomityö tutkii PowerSpyn kehitystä ja siinä tehtyjä ratkaisuja suorituskyvyn kannalta. Ongelmakohtiin suunnitellaan parempia ratkaisuja, jotka toteutetaan, ja joiden tuoma parannus mitataan. Tulokset osoittavat, että ohjelmiston suorituskykyä voidaan parantaa tekemällä oikeita ratkaisuita sekä ohjelmiston suunnittelussa että toteutuksessa.

PREFACE

First and foremost, I wish to thank my beloved wife for the tremendous support I have received throughout the past six years I have spent studying and building my career. It has meant the world to me. Trust me, I will have more spare time from now on.

Also, thanks to the rest of my family, especially to my mother and father, for never judging me or the choices I have made in life. Most of them have taken me to the point I currently am in. Special thanks to Aki for the excessive amount of treats brought to me during my studies. I am looking forward to have one of those NHL evenings, again.

For all my friends, I wish to say that I am grateful for your never ending understanding of me putting my studies first. Thank you for letting me to have my space and I am duly sorry for getting estranged from you all.

Most special thanks to Lassi Riihelä for not only constantly supporting me and being the best friend one could ever hope for but also for making me apply for the technical student position at CERN.

Thanks to CERN and the TE-EPC-CCS section for an unforgettable year and thanks for a unique thesis' subject.

Last, but definitely not least, I want to thank my supervisor, Dr. Lasse Lensu for being my tireless mentor throughout my studies.

As put by Douglas Adams:
"So long, and thanks for all the fish!"

Saint-Genis-Pouilly, May 14th, 2017

Joni Herttuainen

CONTENTS

1 INTRODUCTION **8**
1.1 Power Converters and Data Browsers 8
1.2 Objectives 9
1.3 Limitations 10
1.4 Structure 11

2 BACKGROUND **12**
2.1 History of CERN Power Converter Browsers 12
2.1.1 Spy 12
2.1.2 FGCSpy 12
2.1.3 LabVIEW Data Viewer 13
2.2 PowerSpy 13
2.2.1 Front-End 13
2.2.2 Back-End 16
2.2.3 Desktop application vs. Web application 16
2.3 PowerSpy's Issues 17
2.3.1 Fast Fourier Transform 17
2.3.2 Security 19
2.3.3 Back-End Optimization 21
2.3.4 JavaScript performance 21
2.3.5 Data Formats 23
2.3.6 Communication Format 24

3 IMPLEMENTATION **27**
3.1 Fast Fourier Transform API 27
3.1.1 Restrictions 27
3.1.2 Candidate Selection 27
3.1.3 Candidates 28
3.2 Role Based Access Control 29
3.3 Back-End Optimization 30
3.4 JavaScript performance 31
3.5 Data Formats 32
3.5.1 Parsing time 32
3.6 Communication Format 32

4 TESTING AND PERFORMANCE **33**
4.1 Test Machine 33

4.2 Fast Fourier Transform API 33
4.3 Role Based Access Control 35
4.4 Back-End Optimization 36
4.5 JavaScript Performance 39
4.6 Data Formats 40
4.6.1 Size 40
4.6.2 Parsing Time 40
4.6.3 Addition of JSON format 42
4.6.4 New CSV Format 45
4.7 Communication Format 47

5 DISCUSSION **48**
5.1 Fast Fourier Transform API Selection 48
5.2 Role Based Access Contral 48
5.3 Back-End Optimization 49
5.4 JavaScript performance 49
5.5 Data Formats 49
5.6 Communication Format 50

6 CONCLUSION **51**

REFERENCES **52**

APPENDICES
Appendix 1: FGCSpy CSV data format (old format)
Appendix 2: PowerSpy JSON data format
Appendix 3: PowerSpy CSV data format (new format)

ABBREVIATIONS AND SYMBOLS

AC	Alternating Current
ACW	Accsoft Commons Web
AJAX	Asyncronous JavaScript and XML
API	Application Programming Interface
CERN	The European Organization for Nuclear Research
DC	Direct Current
DFT	Discrete Fourier Transform
FEC	Front-End Computer
FFT	Fast Fourier Transform
FGC	Function Generator/Controller
IdP	Identity Provider
JSON	JavaScript Object Notation
LHC	Large Hadron Collider
RBAC	Role Based Access Control
SAML	Security Assertion Markup Language
SP	Service Provider
SSE	Server Sent Events
SSO	Single Sign-On
TE-EPC	CERN's Electronic Power Converters section

1 INTRODUCTION

1.1 Power Converters and Data Browsers

The European Organization for Nuclear Research (CERN) is an international research organisation lately known for its pursuit and success of finding evidence for the existence of Higgs' boson. However, it is probably best known for having the largest and most powerful particle collider on the planet, the Large Hadron Collider (LHC), the latest addition to the CERN collider complex.

LHC, as its name states, is large indeed. It has the diameter of roughly nine kilometers, a circumference of about 27 kilometers and it consist of hundreds of superconducting electromagnets that are used to control the particles. It is worth noting that LHC is not an independent particle accelerator but the last link in a chain of accelerators, each of which boosts the energy of the particle beam before injecting it into the next accelerator.

Controlling the accelerated beam of particles is done with superconducting electromagnets. These electromagnets are further controlled by power converters which convert the alternating current (AC) from the electrical grid to direct current (DC) and supply it to the magnets. Correspondingly, each power converter is controlled by a Function Generator/Controller (FGC). The FGC provides the power converter with a correct amount of current to cause the electromagnets to provide a desired amount of deflection.

The Electronic Power Converters division of CERN (TE-EPC) is responsible for designing and implementing both the hardware and the software for these FGCs. The Power Converters of LHC are all digital ones, which then again means a huge number of digital signals. These signals are then logged and analysed in order to produce a better regulation of the beam.

For analysing, an interactive software to browse the control signals is needed. This includes good visualization of the signal with a possibility to do noise analysis and zoom in on a selected part of the visualized signal. Of course, it would be possible to analyze the results with non-interactive software tools that allows charting, such as Excel, but then an excessive amount of working hours per year

would be used to cope with the clumsiness of the software.

From this arises the need for proper tools to aid in the development of the FGCs. CERN has had tools in the past to partially answer these needs, but with the growing data sizes and feature requirements, these tools have, however, become less adequate. Even the improved tools are failing to meet the increasing performance requirements. Thus, a new tool on a different platform had to be developed.

PowerSpy is a project that begun in 2014. It is a web application that is designed to overcome all the shortcomings of the previous tools and platforms. It consists of an Apache server with a Perl CGI back-end and a front-end written with a combination of JavaScript, HTML and CSS. Of course, not everything has been written from scratch, but a few open source Application Programming Interfaces (API) and libraries are used in the application to provide required functionalities.

This thesis studies further development of the PowerSpy application from multiple viewpoints. These include the motivation for selected APIs, choices made to improve the efficiency of the application, security viewpoints as well as restrictions set by the platform or implied by the given requirements and specification for the application. The development also comprises of adding new features, such as Fast Fourier Transform (FFT) and new data formats to the existing application.

1.2 Objectives

Since earlier tools fail performance-wise, the main goal of the thesis is to study and improve the performance of the PowerSpy application. In practice, this means studying the current state of the application's front-end, back-end, communication protocol as well as data formats and trying to find better solutions to improve the overall performance of the application. Different solutions are studied and their performance is estimated by measuring the time spent on different tasks. The best solution performance-wise is then utilized in PowerSpy. There may, however, be certain restrictions, which prevent using the performance-wise optimal solution, in which case, the best solution that fits the given criteria is selected.

In the case of the front-end, for example, JavaScript is known to have the possibility of do certain tasks in a number of different ways. To keep the application

as responsive as possible, performance measurements and analyses are carried out to try to find out which methods are the most efficient for each task. Also, some questionable decisions made in the back-end are analyzed performance-wise and optimized.

Also, one of the main goals is to find a suitable Fast Fourier Transform (FFT) API for the application. To achieve this, a JavaScript FFT API benchmark and feature comparison is carried out to find an API that is not only efficient bet also meet the other qiven requirements.

Also, since the main use case of the application is to read and visualize data, the used data formats and communication protocols are analysed and redesigned. The goal of the data format study is to not only gain improvements on the overall performance of the application, but also, to ensure the portability of the data.

Since PowerSpy is a web application, which communicates with the FGCs over a network, security is a non-trivial issue that cannot be omitted. Thus, one goal is to study the application security-wise and find adequate solutions to ensure the security of the users, the application and the devices it communicates with as well as the internal CERN network. The security solutions are based either on common practices or consulted experts' opinions.

1.3 Limitations

The possible solutions are limited by the specifications and requirements given for PowerSpy that must be met as well as by the limits of the tools that are used in the development of the application. Also, generally in software projects, it is also possible for the specifications and requirements to change.

In the optimal case, a real-time data browser would indeed get real-time updates from the server constantly without the need for any user action. Not only are some of the systems in LHC connected to such a slow fieldbus, that this would not be even theoretically possible, but this could also become a problem due to the number of buffer data requests sent to the devices. The excessive overhead could end up in a failure of the device (cf., denial of service attack). Also, the Front-End Computers (FEC) that handle the requests have a limited amount of resources.

On top of handling requests, the FECs must deliver a reliable operation of the beam at all times. Therefore, the application is not real-time in a sense that it would show the current data stored in the memory of a FGC, but it can make requests to get a snapshot of the current data.

The thesis primarily focuses on observing, solving and presenting encountered problems as well as comparing and justifying the choices made. Encountered anomalies and their causes are only studied superficially as opposed to making detailed analytical research on them.

Not all the choices can be based on purely quantitative measurement, but also qualitative one have to be used . In these cases, the choice is made on the basis of the given requirements and specifications. Also, in some cases, experts are consulted (e.g., in case of computer security solutions) and the choises are made based on their professional opinions.

1.4 Structure

This thesis is structured so that in Section 2, background for power converter browsers is presented. This includes also a detailed presentation of PowerSpy. In Section 3, the different aspects of the performance optimizaton and studies to be carried out in this research are presented. The results of the studies are then presented in Section 4 and are further analyzed and discussed in Section 5. Finally, the research is concluded in Section 6.

2 BACKGROUND

2.1 History of CERN Power Converter Browsers

2.1.1 Spy

Spy was the very first of the digital power converter data browsers in CERN. It was developed in early 2000 in MATLAB, but was shortly abandoned due to the fact that MATLAB requires a software licence for development. Also, there was not enough competence in the software section of TE-EPC division to do MATLAB development. Back then, the vizualization elements of LabVIEW were more competitive which lead towards developing the browser in the LabVIEW environment.

The "Spy" in the name refers to the action of acquiring the signals from the FGCs. The signals are recorded in circular buffers during runtime of the converter and the data is available postmortem in case the converter trips. It is also possible to acquire real-time data from the buffer, in other words, "spy" on the data in the buffers, without stopping the recording.

2.1.2 FGCSpy

FGCSpy was CERNs second attempt to create a power converter data browser. It was such a success that it is still in use. FGCSpy was created in LabVIEW, although the platform selection made the software suffer from some of the same shortcomings as its predecessor. Namely, the lack of competence in the development group. A screenshot of the software can be seen in the Figure 1.

FGCSpy managed to stay in active use for over a decade, even though it had some major deficiencies. First of all, the software were only able to show eight signals at a time. It had the possibility acquire up to 40 signals, out of which the signals to be visualized were then selected. What is worse, is that FGCSpy could only show signals that shared the same time base. Thus, it could not even show a pair of signals at the same time no matter how crucial it would have been for

the development of a device.

Fixing all the issues in FGCSpy would have required massive structural changes in the software up to the level of a complete remake. However, there were not enough man power or competent members in the development group to do the job and, thus, the TE-EPC division managed to cope with what they had.

2.1.3 LabVIEW Data Viewer

LabVIEW Data Viewer was an unsuccessful attempt at tackling the shortcomings in the FGCSpy software. Instead of developing the existing FGCSpy that was in active use, a new software was developed from scratch. It included most of the features that were lacking in the previous softwares.

The development of the work was terminated after the first prototypes were tested due to massive performance issues. There was an attempt to optimize the software, but LabVIEW was simply not efficient enough to handle the required amount of data. At this point, it was clear that the next software should be built on a different platform.

2.2 PowerSpy

CERN's PowerSpy is the first browser-based application to access the FGC buffer data. The development of the application began in 2014 and it is still under development. Even though the software is not complete yet, it is in active daily use.

2.2.1 Front-End

The front-end of the PowerSpy is written in JavaScript, HTML and CSS. The front-end uses a multitude of open source libraries that are used to accomplish the main functionalities of the application. A screenshot of the client application can be seen in Figure 2.

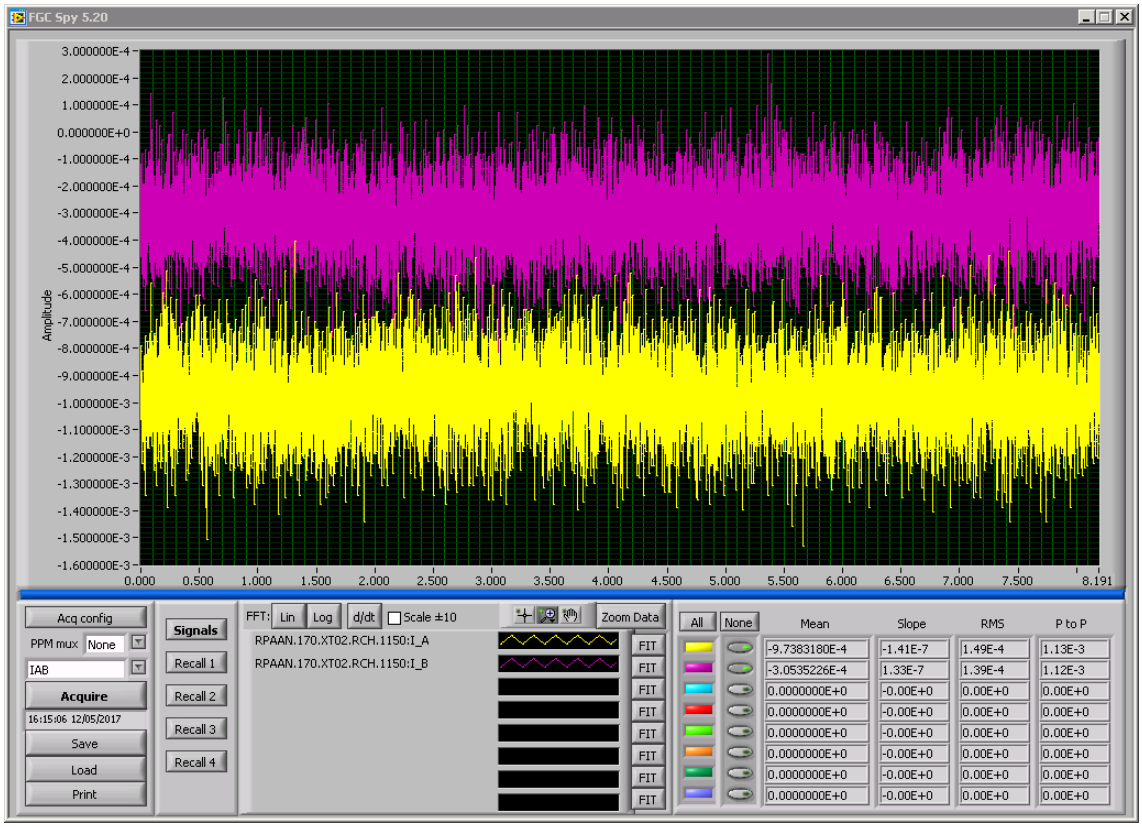


Figure 1. Screenshot of FGCSpy software.

Flot and its plugins

Flot is an open source plotting library for jQuery originally started by Ole Larsen with the support of a Danish software agency called IOLA [1] and currently maintained by David Schnur. It has an active community of users that contribute in the project, for example, by developing the plugins available for Flot. [2]

Flot provides PowerSpy with the ability to plot FGC signal data. Also, a number of plugins have been used to provide more features, such as the tool tip developed by Kris Urbas [3]. However, the most important plugin performance-wise is the downsampling plugin which is based on Largest-Triangle-Three-Buckets algorithm and is created by Sveinn Steinarsson [4]. The downsampling plugin is crucial for PowerSpy since it allows the application to provide charts with signals consisting of millions of datapoints without making the browser freeze. Initially, Highcharts [5] were considered as an alternative charting library, but Flot was finally selected due to its superior performance.

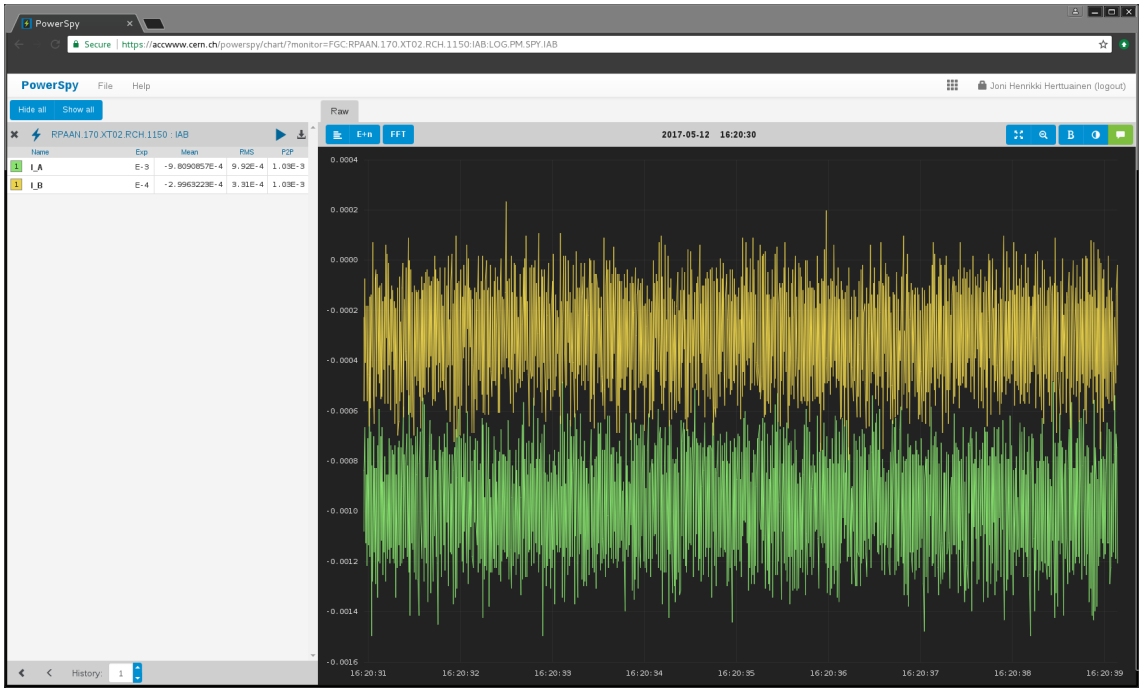


Figure 2. Screenshot of PowerSpy application.

jQuery

jQuery is a JavaScript library designed to simplify a multitude of tasks in JavaScript such as DOM manipulation and event handling. The project was started in 2005 by John Resig and was inspired by Ben Nolan's project called Behaviour [6]. One of the earliest goals was to write more understandable code with less characters. The strive for this has remained to this date and has actually influenced the jQuery's slogan: "Write less, do more.". [7, 8] Since PowerSpy is designed to have a minimal number of dependencies, jQuery is present only due to the fact that Flot depends on it.

Bootstrap

The development team defines bootstrap as "the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web". It is an open source project started in 2010 and was originally known as Twitter Blueprint. It is built with Less [9] preprocessor and is designed to ease and speed up front-end development with multi-platform support. [10]

Mousetrap

Mousetrap is an open source JavaScript library designed to provide keyboard

shortcuts and key sequence handling in web applications. The project started in 2012 and it is created and primarily maintained by Craig Campbell. It has no external dependencies, works with international keyboard layouts and is supported by all the major browsers. [11]

2.2.2 Back-End

The back-end consists of CGI scripts written in Perl. The scripts are run on an Apache server. To communicate with the FGCs, the back-end also uses CERN's APIs written in Perl which was the main reason for selecting Perl as the language of implementation. Login is provided by the CERN Single Sign-On (SSO) and session handling is done using an Apache module called mod_auth_mellon [12].

2.2.3 Desktop application vs. Web application

Even though CERN has some Java-based data viewer tools in operation, the plan was to go towards browser-based web application right from the start. There were multiple reasons to choose a web application instead of a desktop application. The software section of TE-EPC division consists of mostly programmers used to embedded, low-level programming and there was no desire to maintain either Java code nor competency to do Java programming in the section. There were, of course, more to it than that.

First of all, selecting a browser-based application ensures multi-platform compatibility without any extra work from a developer. Also, browsers and their JavaScript engines are actively developed to provide better performance. On the top of that, one of the main advantages is that there is no need for the users to ever install or update the software. What is more, browsers have advanced, easy-to-use developer tools to aid developing and debugging software.

2.3 PowerSpy's Issues

Many requirements were given by the TE-EPC division concerning PowerSpy. The application was lacking certain features that needed to be implemented. Also, certain features were implemented, but the alternatives for the implementation needed to be further studied to find the most efficient way to achieve desired functionalities.

2.3.1 Fast Fourier Transform

One of the FGCSpy's most important functionalities is the Fast Fourier Transform (FFT). It is used for harmonic noise detection in the FGC output signals. In circular accelerators, if there is harmonic noise and the oscillation of the magnetic field is linked to tune (i.e., the frequency of the transverse oscillation [13]) of the beam of the particles, the magnetic field oscillation can cause the orbit oscillation of particles to be amplified. This amplification can in turn cause the beam to be lost. With FFT, these harmonic frequencies can immediately be detected.

Performance
Since the performance of PowerSpy is of such high priority, a literature review of different FFT algorithms was done to have an overall picture of the order of superiority of the implementations.

The term Fast Fourier Transform is generally used for algorithms that compute the Discrete Fourier Transform (DFT) with a computational complexity of less than or equal to $O(N \log N)$, where N is the number of samples in the data . The distinction between the terms FFT and DFT is that the DFT means the mathematical transformation while FFT is often used to refer to the algorithm or to the algorithmic process of computing the transformation. There is a multitude of ways to compute the transformation and finding the most efficient way to do it for different purposes has been a subject of study for decades.

In [14], the so-called Cooley-Tukey method was introduced. The method is based on a so-called divide and conquer principle. In the method, the input sequence is divided into smaller sets, size of which depends on the implementation. Perhaps

the most well-known implementation is radix-2, which recursively splits the input sequence into two parts. Also, other radices (such as radix-4) are used.

In [15], the authors present an algorithm, which does not require complex multiplication in computation of the FFT. Thus, it improves the FFT computation cost compared to the Cooley-Tukey algorithm. The difference is notable in small computers.

In [16], the authors present an algorithm with similar computational cost as the algorithm presented in [15]. The presented algorithm has lower memory requirements with big input sequences than the algorithms used in comparison.

In [17], the authors present the split-radix algorithm that is as flexible as the radix-2 algorithm while requiring significantly less computation steps than any of the algorithms used for the comparison. The split-radix method was further optimized in [18].

In [19], the author presents a Fast Hartley Transformation [20] algorithm, which is then further developed in [21]. The algorithm is not a FFT algorithm per se, but it can be applied in digital signal processing tasks instead of the Fourier transform. [19] The implementation does not require complex arithmetic computations and the algorithm clearly outperforms the Cooley-Tukey based radix-2 algorithm [21, 22].

In [23], the author present the Decimation-in-time-frequency algorithm, which combines the decimation-in-time and decimation-in-frequency algorithms. The resulting algorithm results in a smaller number of computations than the radix-2 algorithm.

In [24], the Quick Fourier Transform is presented. The algorithm is capable of computing the Fourier transform on arbitrary length data and is computationally more efficient than Cooley-Tukey based FFT algorithms, but is less efficient than the split-radix algorithm.

The abovementioned algorithms with their points of interest performance-wise can be seen in Table 1. While FFT algorithms is a widely researched subject even nowadays, these are widely used and benchmarks exist at least for a subset of them. The general consensus of superiority is that the fast Hartley transform

outperforms all the FFT methods. [22, 25, 26] Also, the split-radix algorithm was found to be superior to other Fourier transform based algorithms in [22] as well as in [25]. However, according to the results presented in [26], the radix-4 algorithm seems to outperform the split-radix algorithm.

2.3.2 Security

Single Sign-On

SSO is a centralized login service provided by CERN. It authenticates the user the username and password combination and creates a session cookie for the user. This cookie can then be used in every service or application in CERN the network, provided that they support the SSO login protocol. PowerSpy is designed to use the SSO.

Role Based Access Control

Role Based Access Control (RBAC) is an approach to perform access control based on the rights granted for the given user group (i.e., a role) [27]. Assigning a specific role to a user gives the user a permission to perform one or more tasks on a system or, for example, access specific files on a server. The roles may be related to a user's actual role in a system or in an organization (e.g., system administrator) or they may be artificial roles based on the granted rights. For example, a role 'serverX-access-allowed' set to a user could mean that the user in question has access to server X. Also, the roles can be applied not only to human users, but also to systems or processes.

In CERN, RBAC is used in power converter control system. Whenever a user sends a request for the control system, the authority of the user is validated. The authorization is done by a centralized RBAC server which authenticates the users, provides them with an authentication token and also verifies the authentication tokens. These so-called RBAC tokens are then sent with each request to the power converter control system which will then send the token to RBAC server for verification. If the RBAC token is valid, it is verified and the initial request is then fulfilled or declined based on the roles assigned to the user. The RBAC token is digitally signed, so it cannot be modified, for example, to add roles to a user without making the token invalid in the process. [28, 29]

Table 1. The points of interest (performance-wise) of the presented FFT Algorithms.

Algorithm	Notions
Cooley-Tukey [14]	Divide and conquer principle. Most commonly known FFT algorithm. According to [26], the radix-4 variation outperforms split-radix.
Rader-Brenner [15]	Does not require complex arithmetics. Outperforms Cooley-Tukey.
Preuss [16]	Low memory requirements with big input sequences.
Split-Radix [17]	Outperforms all other FFT-based algorithms.
Fast Hartley Transformation [19]	Not FFT-based. Does not require complex arithmetics. Outperforms all the FFT algorithms.
Decimation-in-Time-Frequency [23]	Less computations than radix-2 (Cooley-Tukey).
Quick Fourier Transform [24]	Arbitrary length data possible. Outperforms Cooley-Tukey.

Originally, each data acquisition for the FGCs was done with RBAC authentication based on the location of the server. That is, the RBAC server authorizes requests coming from the IP address of the PowerSpy server instead of authorizing the actual users making the requests. Not only was there no easy way to backtrack possibly malicious actions, there were no possibility to restrict the access to the converters, if they were in the software. That is, if a developer would make an error, an unwanted (even malicious) user could theoretically get access to the LHC control system.

In Figure 3 and Figure 4, there are simplified diagrams depicting the login process and the data acquisition process as they were originally performed in PowerSpy. As it can be seen in Figure 3, the login process was very staightforward: the client makes a HTTP request for PowerSpy web page, server detects that the client is not logged in and makes a redirect to the SSO server, client signs in and gets a session cookie with the session ID.

The data acquisition process was more complicated than the login process as can be seen in Figure 4. The client first makes a data request, PowerSpy server requests for a RBAC token by location, adds the acquired token to the request and forwards it to the FEC, and then forwards the acquired data back to the client. Not only was the data acquisition complicated, but it was also inefficient. That is, since the server was stateless, it discarded the acquired RBAC token after use

and fetched it again every time it made a request for data even though the token would have been valid for the duration of eight hours.

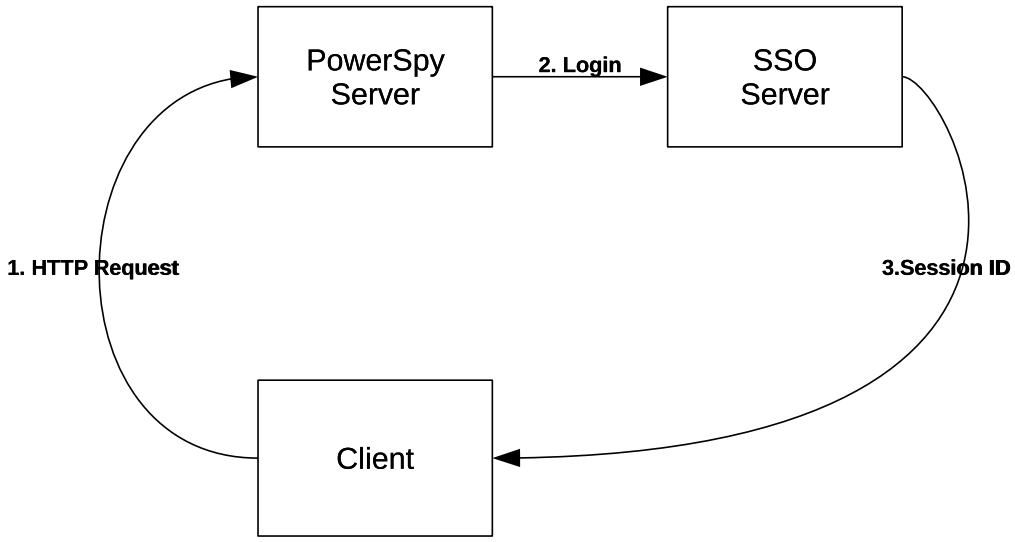


Figure 3. Simplified diagram of the original login protocol. The client makes a HTTP request to PowerSpy Server (1), which redirects the user to the SSO server to login (2). The SSO server provides the client with a session cookie (3).

2.3.3 Back-End Optimization

The API used by the back-end to communicate with the FECs was designed following the principles of Representational state transfer (REST) [30] API design. That has many advantages, including ease of implementation, but it also has its disadvantages. In the case of PowerSpy, one of the major disadvantages is not having a possibility to cache the device information in the back-end. This information (e.g., device names, gateways) is needed when the back-end requests data from the devices. The device information was originally parsed from a so-called name file for every request. Parsing this huge file is a time-consuming operation that can be quite easily speeded-up.

2.3.4 JavaScript performance

There exists a handful of ways to implement a task in JavaScript. For example, finding a correct DOM element can be achieved at least by using findElements-

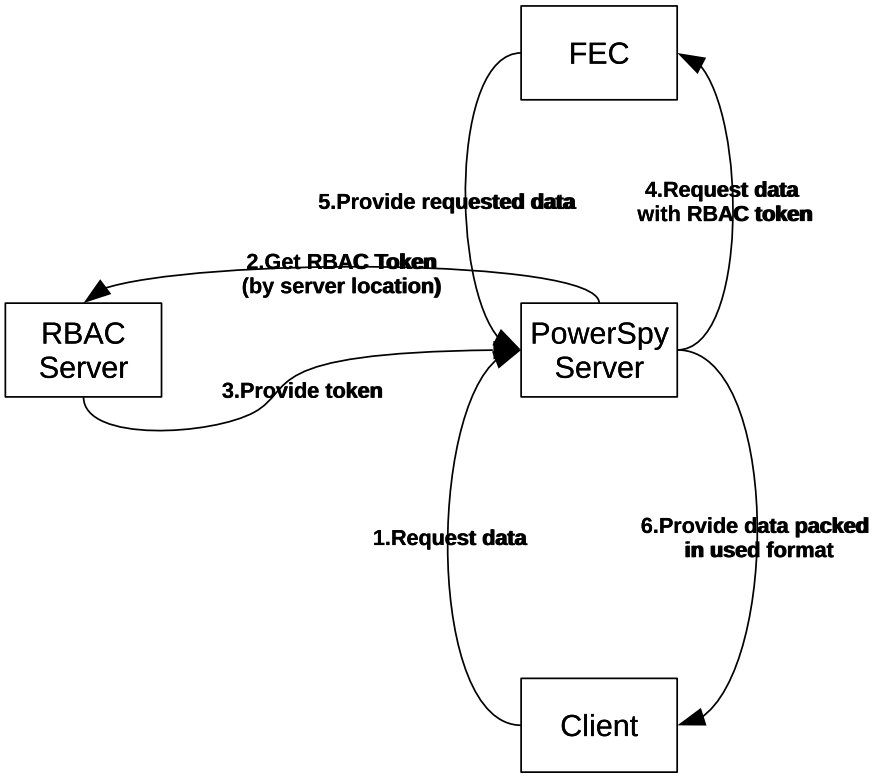


Figure 4. Simplified diagram of the original data acquisition protocol. The client makes a request for data (1), the server gets a valid RBAC token based on location (2,3), the server forwards the client's request with the RBAC token (4), the FEC provides the requested data (5) and the server forwards the data to the client (6). Note that the RBAC token is discarded after use and needs to be reacquired for every request for data.

ByClassName, findElementById or querySelector. On top of these, since PowerSpy uses jQuery and jQuery has its own functions to achieve these functionalities, there are even more possibilities to select from. For this reason, different methods to achieve the often used functionalities in PowerSpy are analysed and benchmarked. The best method performance-wise is selected and this method is then used throughout the code to achieve as instant feedback from the front-end as possible.

2.3.5 Data Formats

Originally, PowerSpy used a Comma-Separated-Values (CSV) as the only data format. On top of its use as the data exchange format, it was also used to export data from the application. This exported data could be imported to a spreadsheet software such as Excel. However, the timestamps included in the PowerSpy CSV file format are not needed in the application. They can be be computed if the time of the first sample and the sampling period are known. Therefore, the data format could be improved and more compact solution, JavaScript Object Notation (JSON) format, was suggested as an alternative solution. What is more, parsing the JSON format in JavaScript is a faster operation than parsing the CSV format, which would also support using JSON.

CSV is a format for storing and transferring data. According to the de facto standard, it is structured so that each line contains one record of data consisting of data fields that are in the same order in each of the records. These fields are typically separated by commas (hence the name). The format can also have a header row. [31]

JSON is a lightweight format for data exchange. The data is structured in a human-readable, easy to understand fashion. Even though it is derived from the syntax of JavaScript object notation, the format itself is independent of the language. The main advantage of the format in web applications is the ability of the JavaScript to convert data from JSON format to native JavaScript objects. [32]

2.3.6 Communication Format

The main use case of PowerSpy is to acquire data from a FGC buffer. Generally, accessing new data can be implemented in two ways: either by making static pages for each data set or by using a dynamic page, to which all data is loaded. In the case of FGC development, the user would most likely wish to access and compare multiple datasets at once and, thus, making the application dynamic was the only option. There are, however, more than one method for a web application to communicate with the server.

PowerSpy was originally designed to have a possibility to subscribe to a device buffer. When subscribing, the client would send a request to the device's FEC to acquire data of the wanted buffer. The FEC would then again fetch the data from the buffer, send it to the connecting client and keep the connection open to send new data dumps to the client periodically. To achieve this functionality, the communication for data acquisition between PowerSpy server and clients was implemented using Server Sent Events (SSE) [33].

It was to be further evaluated whether or not there was an actual need for the data subscription and, thus, the SSE. The objective was to keep the SSE if there was a need for it or, otherwise, to replace it with a simpler, more commonly used and more maintainable communication technology, such as Asyncronous JavaScript and XML (AJAX) [34].

Server Sent Events

Server sent events is a unidirectional communication technology between a client and a server. The technology was designed by Opera in 2006 [35] and is supported by all the modern browsers with the exception of Internet Explorer and Edge. The interfaces that implement server sent events carry the name EventSource [36]. The basic principle of server sent events is that a client establishes a persistent HTML connection to a server, which provides the client with updates of data sent as event streams. On the client side, these event streams are handled the same way as any event (e.g., click, key press).

Figure 5 depicts the basic operation of server sent events. In this figure, it can be clearly seen how the client initiates the communication to the server (blue arrow).

After the initiation, the server provides the client with new data (event streams).

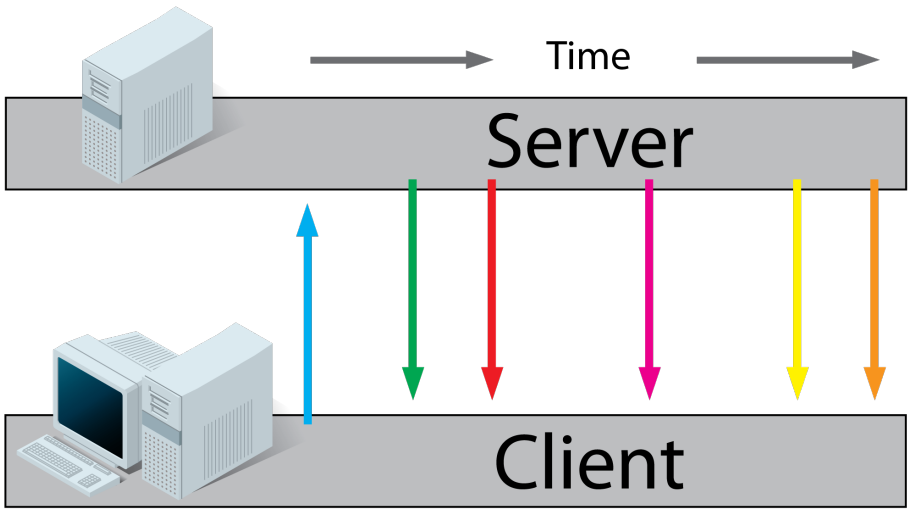


Figure 5. Server Sent Events. Image published under Creative Commons Share Alike [37] license in [38].

AJAX Request

AJAX is an approach originally defined to exchange XML data between the client and the server. It is not, or ever was, a single invented technology, but a combination of existing technologies. It was originally designed to overcome the problem of user needing to wait for the whole page to load before being able to interact with the page [34]. Even though it was initially designed to be used with XML, it is more commonly used with JSON nowadays.

There are two main advantages of using SSE over AJAX. The first advantage is the significantly lower HTTP overhead. In the AJAX method, a client establishes the connection to and sends a request to the server which then again responds to the request after which the connection is closed. Thus, each data request consists of request and response message as well as opening and closing the connection. With the SSE method, the connection is opened only once, and the data updates need not to be requested. The second advantage is the quick access to new data since it can be sent to the client immediately when it becomes available.

Figure 6 depicts multiple AJAX requests. In the figure it can easily be seen how the client needs to keep constantly requesting for new data. By comparing this

figure to Figure 5, one can instantly see how the number of requests made is bigger in the case of AJAX protocol.

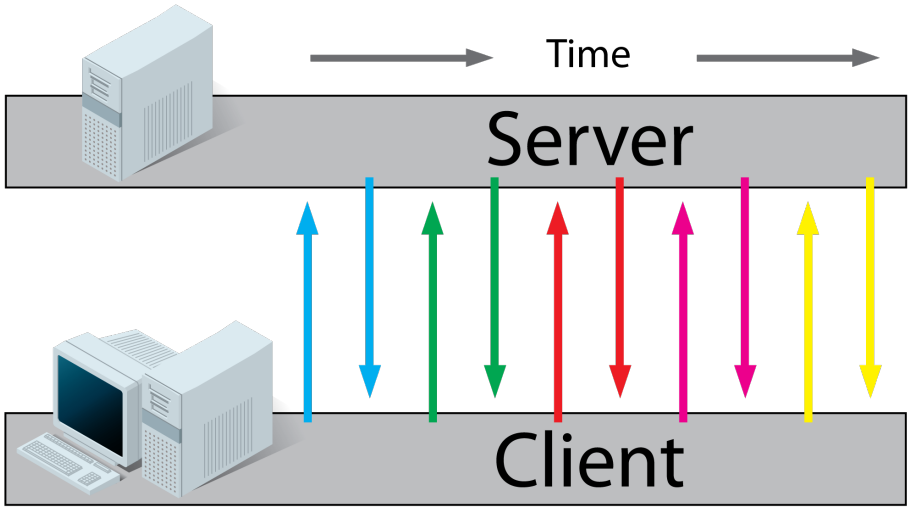


Figure 6. AJAX request. Image published under Creative Commons Share Alike [37] license in [38].

3 IMPLEMENTATION

3.1 Fast Fourier Transform API

3.1.1 Restrictions

In PowerSpy, the FFT feature was not yet implemented nor was there any API suggested for it. There were certain requirements given for choosing the FFT implementation: It had to be quick, have a free software license, it had to have an adequate support and it had to add no unnecessary dependencies. If the API also had windowing functions, it was considered as a positive feature. The objective was to find a ready-to-use solution on the Internet. In a case that suitable solution was not found, the solution would be to implement the FFT API from scratch. Since some of the given restrictions were vaguely specified or otherwise open for subjective interpretation, they were further specified.

Perhaps the most vague restriction was the adequate support. Adequate support can be a person responding to an e-mail within specified time frame or it may be immediate online support. In the case of PowerSpy, it was specified to mean that the API has an active community of users or the API is actively developed. Again, active is a highly subjective term, so it was further specified to mean that there has been some activity within last 12 months.

The requirement of the API not having any unnecessary dependencies was interpreted that the API would need to get by itself. This means that the API would not need any additional libraries or, for example, translators. Thus, the FFT had to be written in pure JavaScript without dependencies to other libraries. The requirement was also specified to be a restriction on the ease of use: no dependencies to expertise. That is, the user can use the API with little knowledge of FFT itself.

3.1.2 Candidate Selection

There are surprisingly many JavaScript-based open-source APIs available on the Internet. Many APIs were found while researching the available options, some of

them were even targeted for servers [39]. Most of the APIs were implemented using the Cooley-Tukey algorithm, whilst more efficient solutions were also available.

In an article written in The Breakfast Post [40], there is an exhaustive list of openly available FFT APIs as well as some APIs that are compiled from C to asm.js [41] using Emscripten [42]. In the article, it was found that the APIs translated to asm.js perform better than native JavaScript APIs. All the APIs considered for PowerSpy can be found in this list. In the article, there are also the results of a benchmark done with a subset of the APIs. Since not every API considered for the PowerSpy were included in the selection, a benchmark was done to find out the algorithmic efficiency of the candidates, although other restrictions would weigh more in the selection.

All in all, five different APIs and a modification to one of those were considered as being potentially used in the PowerSpy application. The selection for testing was quite straightforward: take any pure JavaScript solution that can be used to implement the FFT in PowerSpy. All the JavaScript-based APIs presented in [40] were selected, except for the Timbre [43] due to the fact that it is a JavaScript library for objective sound programming and the FFT implementation is merely a minor part of it which would complicate maintaining and updating the PowerSpy in the future.

3.1.3 Candidates

Nayuki [44] provides the same FFT API available in six different languges. The API is compact, light, easy to use and easy to add to PowerSpy. The computation of the FFT is based on the Cooley-Tukey algorithm. However, the API is not being actively developed nor does it have window functions. Also, it is difficult to find any community support in the case it is needed. However, the efficiency is one of the best amongst the candidates.

The aforementioned Nayuki's FFT API was slightly modified by the author of [40] to achieve even more efficient computing. The made modification creates an object for the FFT, precalculates the used sine and cosine values and stores them into the object. This seems to decrease the computation time, but in the

case of PowerSpy, it does not decrease the overall time due to the fact that there is rarely a need to calculate the FFT for a signal more than once.

The fft.js [45] seems to be the slowest candidate. The API is no longer maintained or developed. It also has no window functions nor any kind of support. Even though the API itself would bring no extra dependencies, it is a highly uncertain option as there has been no activity on the API for years.

The jsfft [46] API is the most recently updated FFT API of the candidate. However, during the time of the FFT benchmark, it was not updated for over a year. The API has no window functions, but is a pure JavaScript solution.

The most popular API of the candidates is the dsp.js [47]. It is not only a FFT API, but a digital signal processing one. Hence the name, dsp.js. Whilst it does not have a commununity support per se, search engines tend to find forums (e.g., Stack Overflow) discussing about this API more than the other candidates. This API has window functions, and it adds no extra dependencies. What is more, it has multiple different FFT algorithms implemented, including the split-radix algorithm. The presence of the split-radix implementation lead to the hypothesis that this would probably be the fastest API to be tested due to its efficiency noted in Section 2.3.1.

The KissFFT [48] is the only candidate that is not written in JavaScript but in C. It was translated to asm.js with Emscripten [42] by the author of [40]. This is the clumsiest solution, since each time the API is modified, it would need to be recompiled, which adds an extra dependency on Emscripten and on the know-how to use it. Also, this particular API does not contain window functions.

3.2 Role Based Access Control

The CERN Accsoft Commons Web (ACW) team has developed an application template together with RBAC team, which implements a so-called SSO-to-RBAC bridge. The application template can be used as a basis when one begins to develop new software without having to add all the libraries needed by the application for it to run in the CERN network. The SSO-to-RBAC bridge is a method of acquiring a RBAC token during login. Basicly, when the user signs in using

SSO, the login is wrapped in a Security Assertion Markup Language (SAML) response [49], which is kind of a sealed envelope. This SAML response is signed by the SSO service and, thus, cannot be forgered. The SAML response containing the assertion, can then be sent to the RBAC server, which then verifies the login, and responds with a RBAC token of the user. In Figure 7, one can see how the login is implemented in the ACW application template.

The ACW application template, however, is Java-based and it uses the Spring framework [50]. This means that the SSO-to-RBAC bridge implemented in it cannot be directly applied to PowerSpy. Therefore, a similar solution was implemented in Perl.

The SAML response from the SSO is valid for only an hour after the login. After the SAML response expires, it can no longer be used to get a RBAC token from the RBAC server. Due to this, it was obvious that the RBAC token would need to be stored somewhere to be reused later on. Also, this would mean that the back-end does not have to retrieve a new RBAC token in each request which would, in principle, speed up the back-end.

The previous client software, such as FGCSpy stores the RBAC token in client side since the applications have no server between the client and the FGCs. Hence, the initial thought was to save the token on the client side instead of adding adding a database on PowerSpy and, thus, complicating the server. The main reason was, that it would not matter whether a possibly malicious user got access to the RBAC token or the SAML respons, since the outcome would be the same. However, this was to be decided together with CERN security experts. Also, the amount of time saved due to not needing to fetch the token in every request would be measured out of curiosity.

3.3 Back-End Optimization

Two alternative strategies were designed for getting the needed information without parsing the whole name file on the server. The first was to use regular expressions to find the correct line for the device in the file and extract only the needed information. This would theoretically speed up the process by a great deal since the back-end can omit hundreds of lines of data. The second option would be

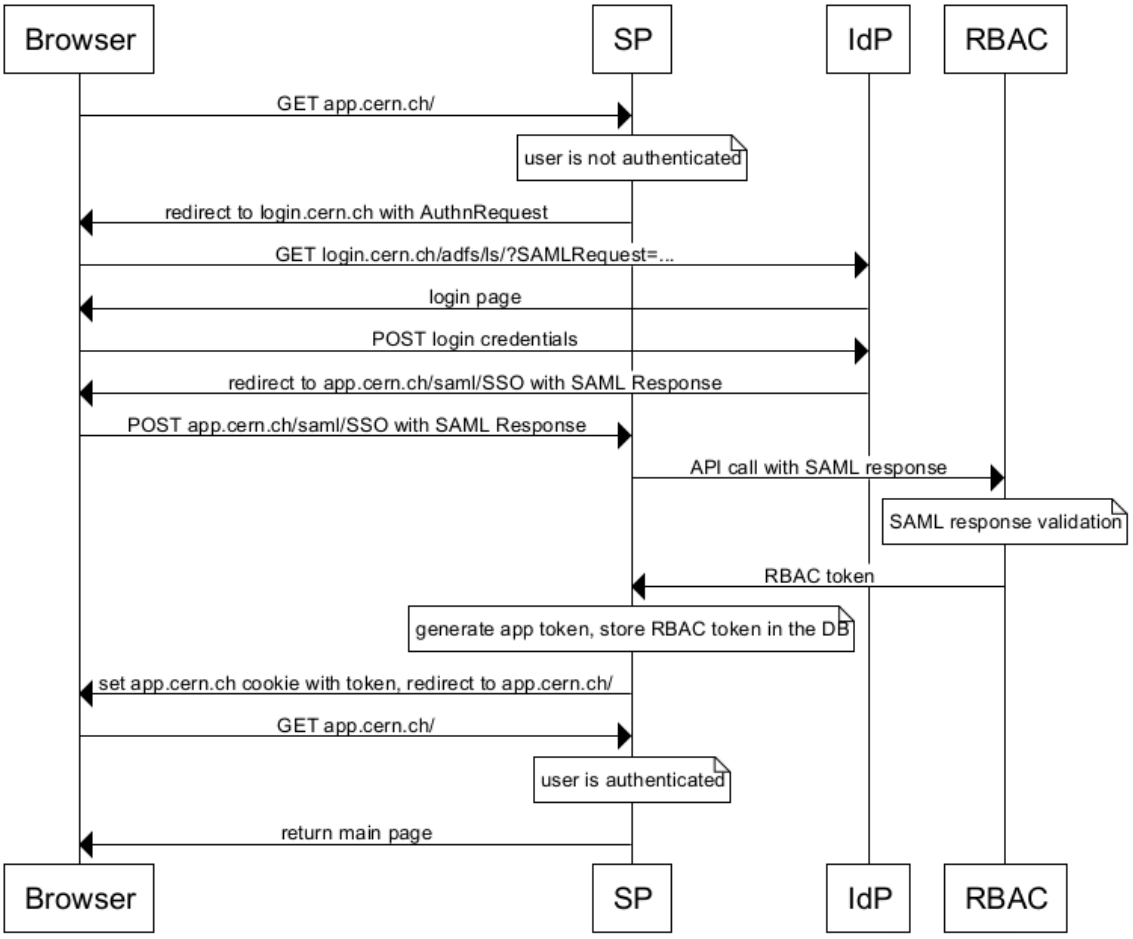


Figure 7. The login protocol as implemented in ACW project template. In the case of PowerSpy, service provider (SP) is the PowerSpy server and identity provider (IdP) is the CERN SSO server. Note that the RBAC token acquisition is initiated in the server (API call with SAML response). The original diagram is presented in [51].

to include the needed information to the request so the back-end can skip the whole process of handling the name file. These methods were to be tested and measured.

3.4 JavaScript performance

The JavaScript performance tests are run on an online application called jsPerf [52]. jsPerf is an application where users may create and share test cases to compare the performance between JavaScript functions or snippets. Test cases were restricted to those functionalities that occur often in PowerSpy.

3.5 Data Formats

Originally, PowerSpy used the CSV format both in the communication between the client and the server and as the means of data output to and input from files. The objective of the data format study was to determine if JSON format was a better substitute for CSV in terms of network overhead (size of the data) and data parsing time. The size of the formats was to be measured along with the parsing time of the formats.

3.5.1 Parsing time

The main objective in a possible JSON format implementation was to reduce the time to visualize the data after it is received. The decision, whether or not to change the existing CSV implementation to JSON was subject to the given requirements. In the requirements, it was specified that, on average, the time it takes to parse the data in the CSV format should be within a factor of two times the time it takes to parse the JSON for it to stay as the communication format.

The study was carried out by first creating synthetic data samples in both CSV and JSON and computing the ratio of CSV parsing time to that of JSON parsing time for each data sample. The generated data was exactly the same in both formats. That is, after parsing the data, the structure parsed from CSV was identical to the structure parsed from JSON. Every data sample was then parsed and the parsing times were then recorded for the both formats.

3.6 Communication Format

It was to be further evaluated, whether there was a true need for the SSE and if there was a reason to maintain it. If there was none, AJAX was to be implemented to replace the existing SSE solution. The actual evaluation was done by consulting the experts of TE-EPC to find out is there or will there be a use case that would require the SSE to be maintained.

4 TESTING AND PERFORMANCE

4.1 Test Machine

All the tests were run in 64-bit CentOS Linux 7. The used browser was 64-bit Mozilla Firefox ESR (version 52.0). The computer had Intel's 8-core i7-4790 processor (3.60 GHz, 8 MB cache) and 16 GiB of 1600 MHz DDR3 memory.

4.2 Fast Fourier Transform API

The considered candidates to be added in PowerSpy can be seen in Table 2. The table rows consist of per-API information (presented in Section 3.1.3) and columns in the table consist of candidate names, their efficiency, license information, support for the API, window function availability and whether or not the API adds extra dependencies. The table is colour-coded so that the green cells denote a positive outcome for a feature, red cells denote a negative outcome and yellow ones denote neither a positive nor a negative outcome.

The compution speed of a FFT API was measured by performing 50 Fourier transforms on a 2^{19} sample data and taking the average out of those. This result was then compared between the all the FFT APIs. The speed was measured in two parts: the time it takes to compute the FFT and the overall time. Overall time here is the sum of the time to allocate the memory for the resulting data structure and the time to compute the FFT. This was done just to further understand how the time is spent in the computation of the transform. In Table 2, the overall time is shown first and the mere computation time is shown in parenthesis.

Out of the candidates, the Split-Radix implementation of dsp.js clearly outperformed all the other candidates. It also met all the other requirements specified for the FFT API: it is the only candidate that has window functions, has open source software licence, active community and it does not add any extra dependencies. Thus, it was selected to be integrated in PowerSpy.

Table 2. Candidate FFT APIs. Each row represents an API and its features selected for the comparison. Each column represents a feature. Red, yellow and green color represent that the features are considered as negative, neutral and positive aspects, respectively. The time has two values: overall time and time to compute the FFT only. The time values are averages of 50 measurements of computing the FFT for a 2^{19}-sample signal.

API Properties					
Name	Time (ms)	License	Community and Support	Window functions	Extra dependencies
Nayuki [44]	71 (60)	MIT [53]	Last update two years ago, no recent activity.	No	No
Nayuki-Obj [40]	73 (48)	MIT [53]	Not supported in any way.	No	No
fft.js [45]	191 (92)	BSD (2 clause) [54]	The author is not currently maintaining the API. Last update years ago.	No	No
jsfft [46]	190 (190)	MIT [53]	Last update in February 2017, very little other activity	No	No
dsp.js [47]	65 (57)	MIT [53]	Latest update in May 2016, active and quite popular.	Yes	No
KissFFT [48] (C>asm.js)	90 (30)	BSD (3 clause) [55]	Latest update years ago but it seems that the API is popular and the support forum is active.	No	Yes

4.3 Role Based Access Control

After discussing the options with numerous CERN experts, which include computer security, RBAC and SSO experts as well as system administrators, it was clear that the SAML response did not pose as much of a threat as was initially thought. First of all, it is valid only for an hour. Secondly, it is only sent once for the server, after which, as long as the client has a valid session cookie, it is never sent again. If the session cookie becomes invalid, a new login is needed. Therefore, it was imminent that the RBAC token would need to be stored on the server side instead of storing it in the browser. Storing the token in the browser would increase the security risk by requiring the browser to constantly send the token to the server.

In Figure 8 and Figure 9, there are simplified diagrams for the improved login and data acquisition, respectively. The original login process (see Figure 3) was augmented with the additional step of requesting the RBAC token and storing it in a database on the server as can be seen in Figure 8. Note that even though the RBAC token is stored on the server, the request to acquire it is initiated in the PowerSpy client. That is, when the client is logged in, a JavaScript code launches an AJAX request for the server to acquire the token.

Now that the RBAC token was stored on the server side, the data acquisition process was significantly simplified. The process is depicted in Figure 9. When the client makes a request for data, the server can now use the SessionID included in the session cookie to retrieve the RBAC token from a database and attach it to the request without acquiring a token for each request which used to be the case (see Figure 4). The process was also significantly speeded up, since removing the RBAC token acquisition caused the back-end to use 228 milliseconds less time for each request.

The acquisition of the RBAC token could have also been implemented so that the server would automatically request the token on login. This would have required altering the mod_auth_mellon module used for session handling. That is, when mellon receives the SAML response, it would have to make a request to the RBAC server with the SAML response, store the token as a session variable and continue on its normal routine. However, changing the mellon would have caused a maintainability issue in the case that mellon was updated and, thus,

this modification gained no support and was not implemented.

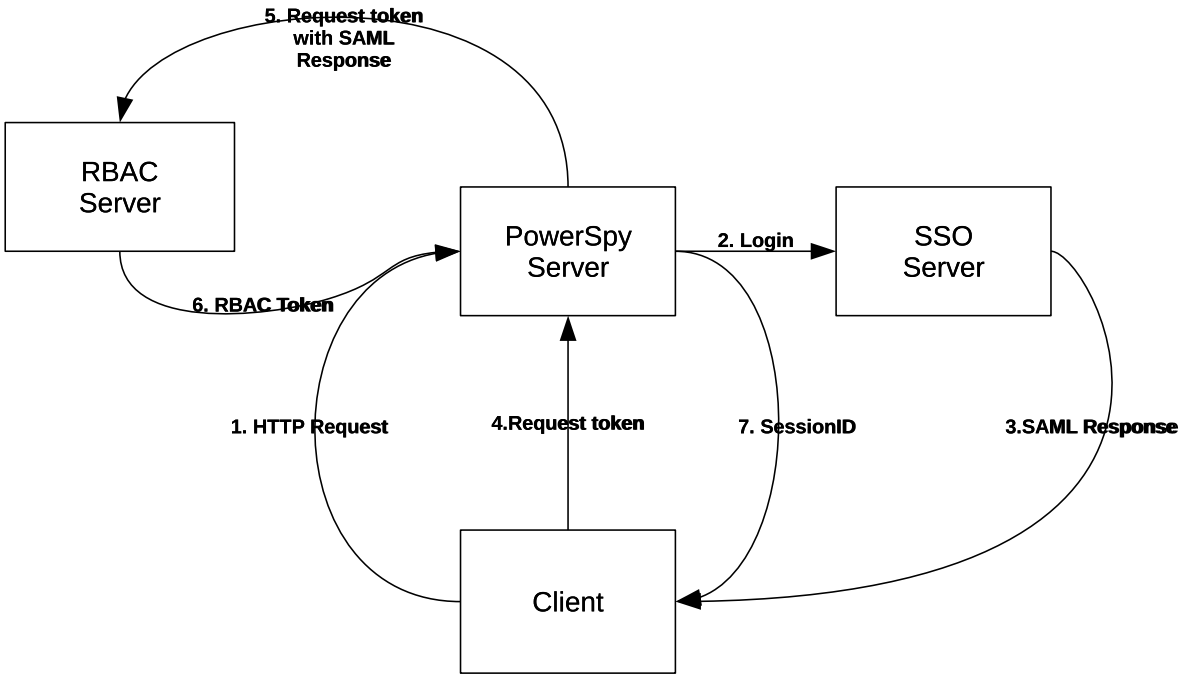


Figure 8. Simplified diagram of the new login protocol. The Client makes a HTTP request to the PowerSpy Server (1), which redirects the user to the SSO server to login (2). The SSO server provides the client with a SAML response (3). The client makes a request for the RBAC token for the server (4), which authenticates the user with the SAML response (5) and gets the RBAC token as a response (6) and provides the client with a PowerSpy cookie (7). Note that in reality, between steps 3-4, the SAML Response is actually sent to the server which provides the user with an Apache session cookie.

4.4 Back-End Optimization

There were three strategies to consider: the original, only finding the needed line with regular expression and sending the needed data with the request. A total of 522 measurements were done and the time for each strategy.

In Figure 10, there is a box plot depicting the results for different strategies to get the correct gateway in the back-end. The leftmost plot (ParseFile) represents the original strategy to parse the whole file. The median time for parsing the structure was 282 milliseconds.

The middle plot (RegexLine) represents the strategy of using a regular expression to find the correct line. The median time for extracting the needed information was

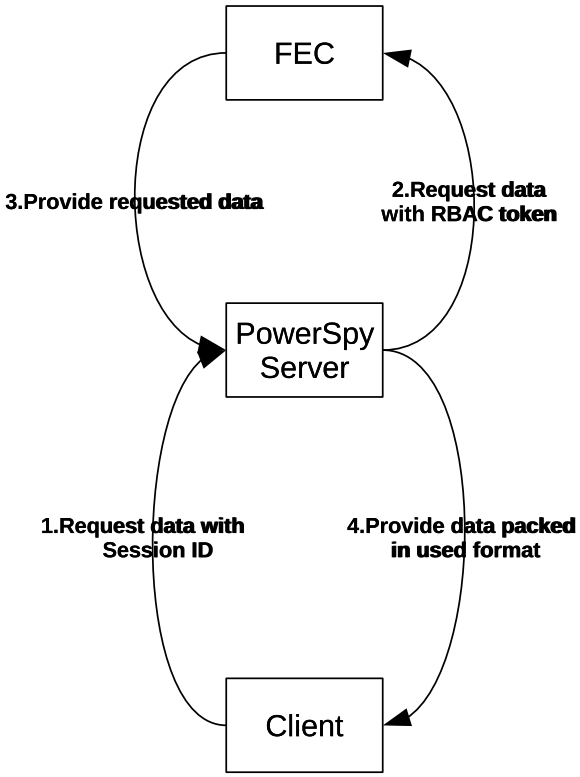


Figure 9. Simplified diagram of the new data acquisition protocol. The client makes a request for data (1) for PowerSpy server, which finds the RBAC token mathing client's session ID and forwards the request to the FEC (2), which provides the PowerSpy server with the required data (3) which packs the data and sends it to the client (4).

24 milliseconds which means this strategy uses about 91% less time or, in other words, is about 11 faster than the original strategy.

As expected, the third strategy of providing the needed information was the fastest by having a median time of 9 microseconds. This short time consists of storing a parameter from the request to a local variable in the back-end. This is represented by the rightmost plot (InsertData) that is reduced to only a red line that represents the median. Compared to the original strategy, providing the information is about 30,000 times faster strategy.

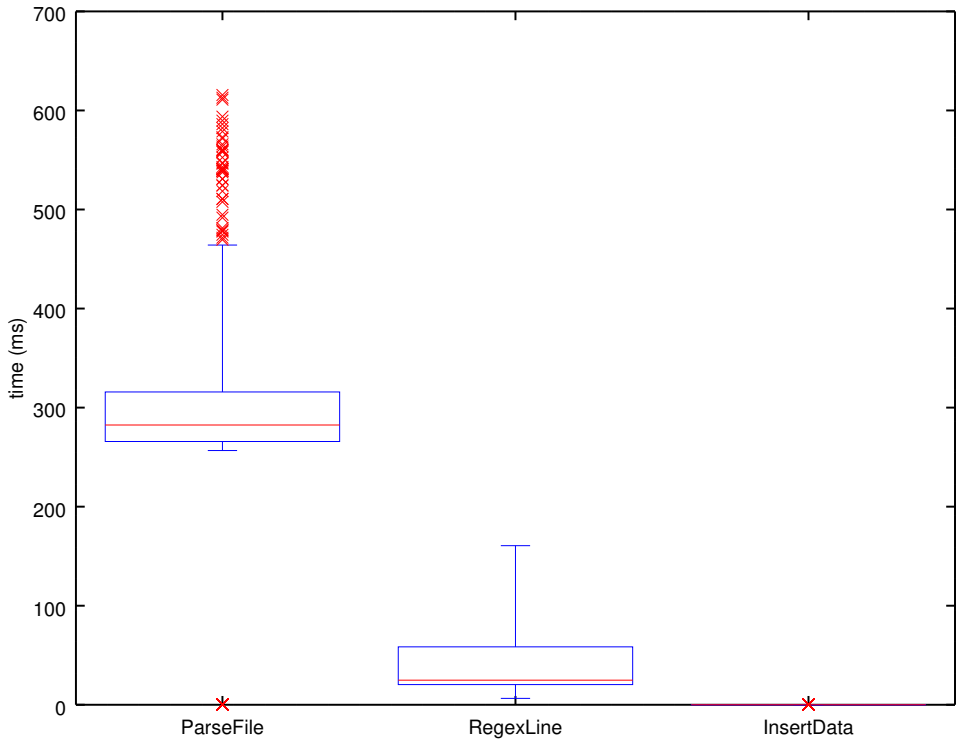


Figure 10. Namefile parsing alternatives. In this figure, one can see the average time for the original solution (on the left), finding the correct line from the file with regex (in the middle) and sending the needed information (on the right). In the plot, red lines are the median values of the averages, box boundaries represent the 25th and 75th percentile, whiskers extend to the most extreme values not considered as outliers (within 3 standard deviations from the mean) and red 'x' is used to plot the individual outliers.

4.5 JavaScript Performance

JavaScript performance tests were done on certain functionalities that occur relatively often in PowerSpy. These include adding new DOM elements to the web page, finding a correct element on a web page, adding event listeners to the elements and using methods versus using prototypes on objects.

In Table 3, the results for the JavaScript performance test can be seen. Each of the test cases are introduced and explained in The test cases are further explained in the web page of each test case [56–59]. The values in te table represent the number of times the functionality in question can be performed in a second (i.e., operations per second). In the table it can be seen, that the best solutions are native JavaScript solutions in all of the cases but handling event listeners, in which jQuery's solution was significantly faster. In comparison between solutions to implement objects' methods, prototype clearly outperformed private function.

Table 3. The JavaScript performance comparison. Each value represents how many times the functionality can be performed in a second.

Adding DOM elements [56]		
innerHTML	appendChild	(jQuery) .append()
79,060	56,837	4,678
Finding a DOM element [57]		
querySelector	getElementsByClass-Name	(jQuery) select
73,710	1,072,283	55,868
Event listeners [58]		
addEventListener	(jQuery) .on()	
362	184,737	
Object methods [59]		
prototype	function	
8,087,486	63,970,115	

4.6 Data Formats

4.6.1 Size

The size of a file in PowerSpy JSON format is notably smaller than the size of a file in PowerSpy CSV format. This is due to the fact that the JSON format does not have the same limiting factors as the CSV file. PowerSpy only needs the time stamp for the first sample and the sampling period and, thus, the time stamps can be left out of the JSON file format.

Since the CSV format needs to be portable to a spreadsheet software, it has to have the time stamps included. The time stamp format is specified to be epoch time with microsecond accuracy. Therefore, the size of a time stamp is larger (in characters) than the average value field for a signal. Hence, if a CSV and a JSON file represent the same data, the size of the JSON file is definitely smaller than the size of the CSV file, and the less there are signals, the greater is the difference. The original PowerSpy CSV format can be seen in Appendix 1.

In Figure 11, there is a plot depicting the ratio of the PowerSpy JSON format to the PowerSpy CSV format by the number of signals. From this figure, it can be easily seen how significantly the JSON format is smaller than the CSV. It is worth noting that the number of signals (the columns in CSV) is the only variable affecting the ratio.

4.6.2 Parsing Time

A total number 100 data samples were generated for the both formats by varying both the number of signals from 10 to 100 with 10 signal intervals and samples per signal from 10,000 to 100,000 with 10,000 sample intervals. In the foreseeable future, PowerSpy would only need to handle data with fewer signals than 30 most of the time. However, the upper boundary was set notably higher to investigate the correlation between the number of signals and the CSV to JSON ratio. The hypothesis was that when the number of signals increases, the JSON parser has to perform more memory allocations due to the specified format and, thus, the ratio would decrease. The selected boundaries for the samples per signal

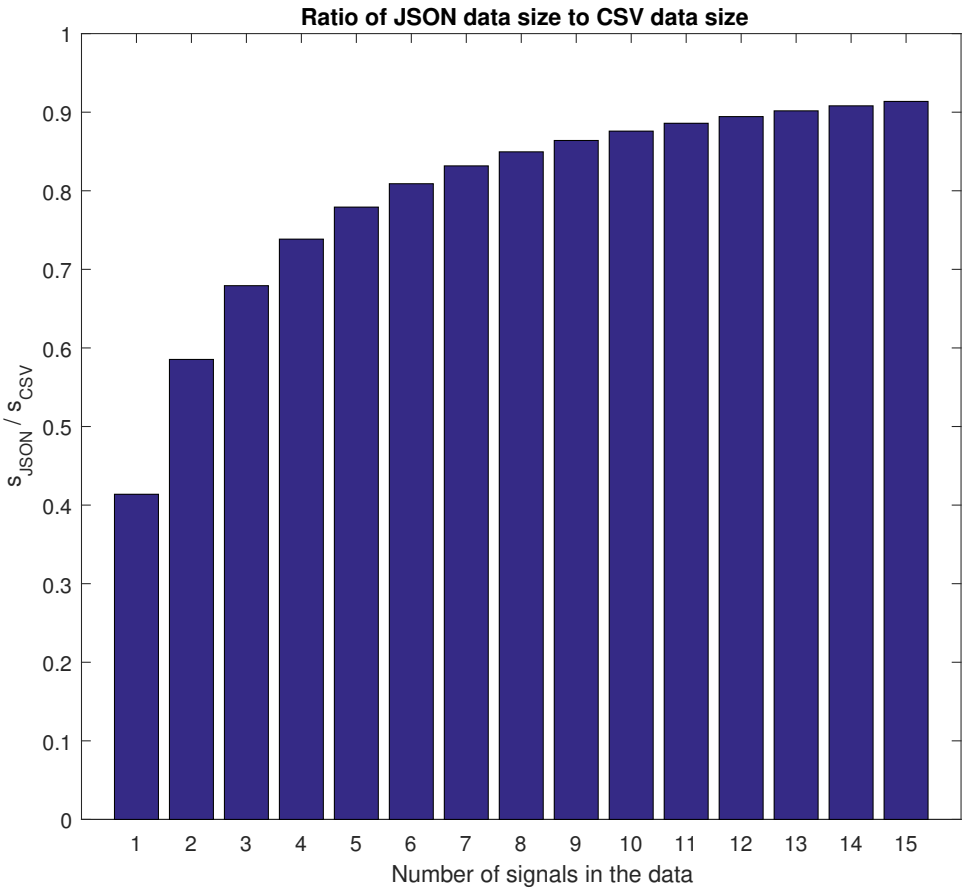


Figure 11. The size of a PowerSpy JSON format with respect to the CSV format. It can be seen how JSON format is significantly smaller than the CSV format. The difference is more obvious if the number of signals is small.

represent the range of likely use cases.

Figure 12 and Figure 13 show the results of the parsing time study. In the figures, the vertical axis shows the CSV to JSON ratio. One can clearly see that there is significant fluctuation in the ratio which seemed to stabilize in the region where $70 \leq N_{signals} \leq 100$ and $60{,}000 \leq N_{samples} \leq 100{,}000$. Within these ranges, the ratio had a value of ${}^{CSV}/_{JSON} \approx 4.20$. The overall average CSV to JSON ratio was ${}^{CSV}/_{JSON} \approx 5.50$.

In Table 4, one can see the previously plotted results per signal. The values acquired with 100 signals and 10,000 samples as well as 70 signals and 20,000 samples were both considered as outliers since they were over two standard deviations apart from the mean. These values are marked with red color in the table. Studying the reason for the anomalies is, however, beyond the scope of this study.

There was no clear correlation between the ratio and number of signals or between the ratio and number of samples per signal. It is, however, difficult to be certain since there seemed to be minor lags in CSV parsing which may have affected the results. The lag appeared in cycles after the test script had parsed a certain amount of data, which was noticed after randomizing the order of parsing of CSV data. However, researching the subject further is beyond the scope of this study.

4.6.3 Addition of JSON format

Since the average CSV to JSON value was significantly bigger than 2, which was used as a threshold value, the JSON format was to be implemented in the communication between the client and the server. During the development of the application, the need for metadata increased and new data fields were added in to the format. Previously, trying to obey the CSV de facto standard restricted adding metadata for the FGC devices and their signals. Now, with the possibility to use JSON, this was no longer an issue. In Table 5, one can see which data fields the data formats contain and the differences between the old and new data formats. The complete PowerSpy JSON format specification can be seen in Appendix 2.

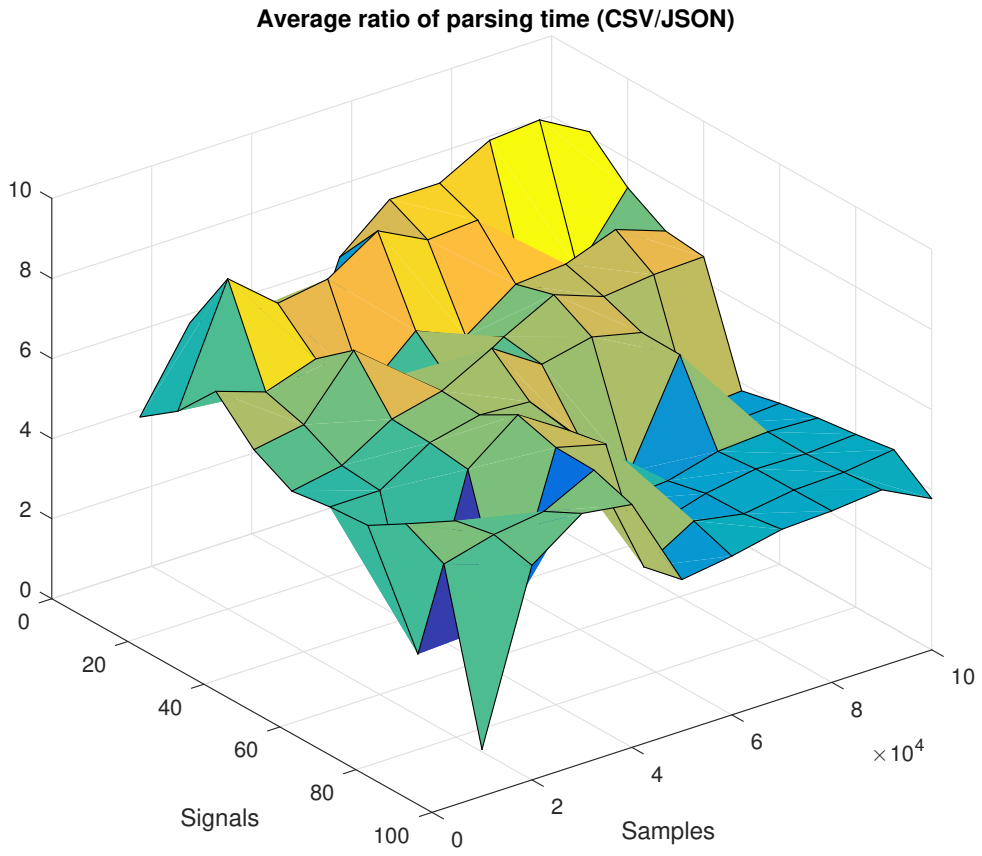
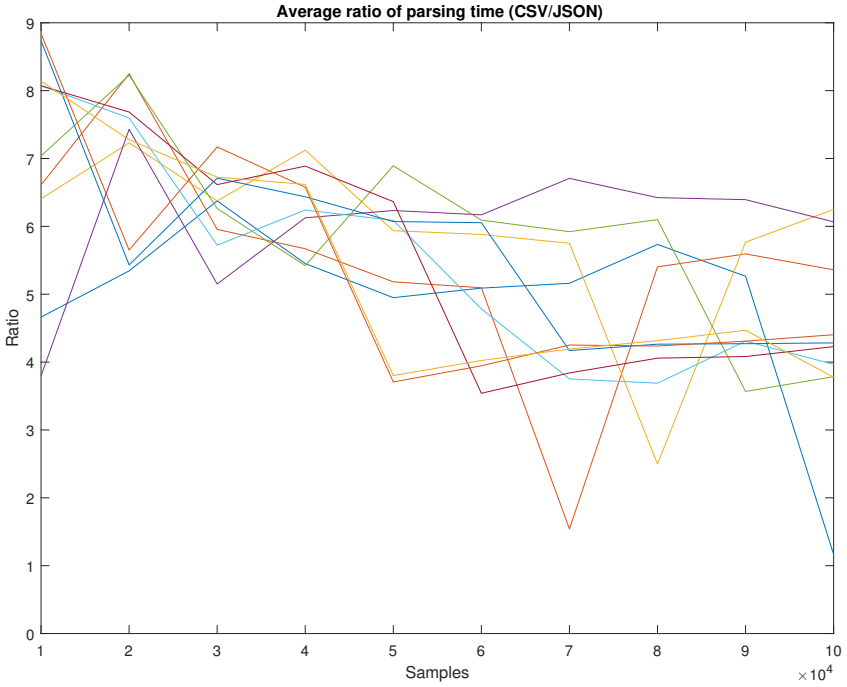
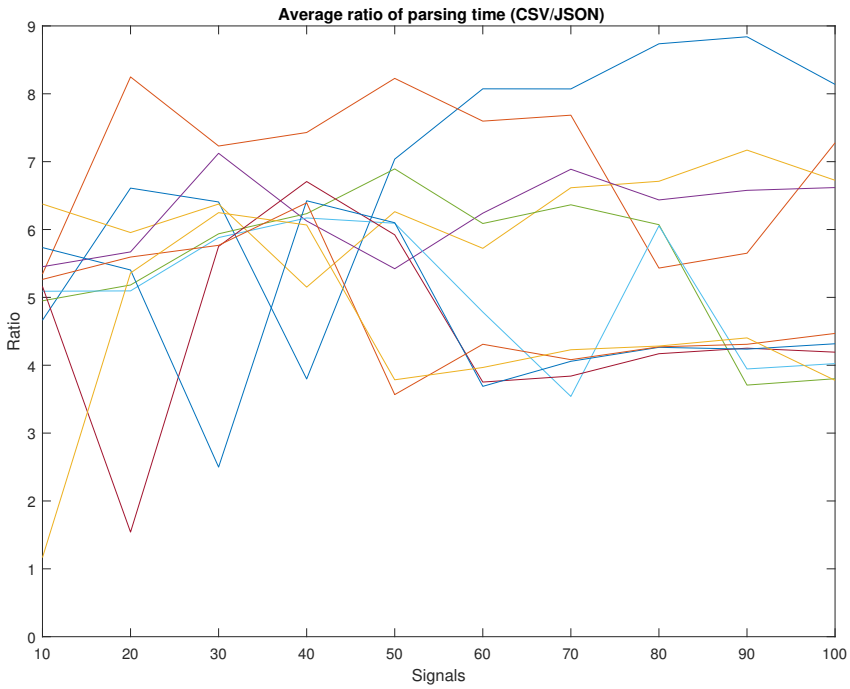


Figure 12. The ratio of CSV parsing time against JSON parsing time. From this figure, it can clearly be seen that parsing data from CSV format takes on average at least four times the time it takes to parse similar data from JSON data.



(a)



(b)

Figure 13. The ratio of CSV parsing time against JSON parsing time. From these figures, it can be seen how the number of samples (a) and the number of signals (b) affects the CSV parsing time to JSON parsing time ratio.

Table 4. The ratio of CSV parsing against JSON parsing time. The values considered as outliers (over 2 standard deviations apart from the mean) are marked in red.

		Samples (in thousands)									
		10	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Signals	**10**	4.7	6.6	6.4	3.8	7.0	8.1	8.1	8.7	8.8	8.1
	20	5.3	8.2	7.2	7.4	8.2	7.6	7.7	5.4	5.7	7.3
	30	6.4	6.0	6.4	5.2	6.3	5.7	6.6	6.7	7.2	6.7
	40	5.5	5.7	7.1	6.1	5.4	6.2	6.9	6.4	6.6	6.6
	50	4.9	5.2	5.9	6.2	6.9	6.1	6.4	6.1	3.7	3.8
	60	5.1	5.1	5.9	6.2	6.1	4.8	3.5	6.1	3.9	4.0
	70	5.2	1.5	5.8	6.7	5.9	3.8	3.8	4.2	4.3	4.2
	80	5.7	5.4	2.5	6.4	6.1	3.7	4.1	4.3	4.2	4.3
	90	5.3	5.6	5.8	6.4	3.6	4.3	4.1	4.3	4.3	4.5
	100	1.2	5.4	6.2	6.1	3.8	4.0	4.2	4.3	4.4	3.8

4.6.4 New CSV Format

The addition of the JSON format to PowerSpy came with new requirements. One of the requirements for the format was that whatever data or metadata was to be stored in the JSON format, it should also be possible to store it in the CSV format. One of the consequences led to the need to reinvent the CSV format in PowerSpy and to deviate from the CSV format standard due to the newly added data fields. Also, the format had to be able to be easily imported to spreadsheet software such as Microsoft Excel.

Initially, two options were considered for the new CSV format. In both of them, the original would change only a little. The most obvious was to use the very first line of the file to store all the so-called buffer parameter values. Another solution would be to keep using the very first cell of the first line and bloat it with all the parameters. Using the whole first line would be much more human-readable solution, but removing contents from a single cell would be more effortless than removing a whole line in the spreadsheet software. This would intuitively favor the single cell solution.

Another problem was, how to tell which value represents which parameter. Originally, there were only one parameter, which was either the text 'time' or a number.

Table 5. Comparison of data types between old and new data formats.

	Old CSV	New CSV/JSON
Buffer metadata		
Buffer Name	x	x
First sample time	x	x
Period	x	x
Time Origin	x	x
Data Source		x
Data Type		x
Table type		x
Classes		x
Per-signal metadata		
Signal name	x	x
Step interpolation	x	x
Time offset		x
Data types		
Analog signal	x	x
Digital signal		x
Table		x

Again, intuitive solution would be to keep them always in the same order. With a few parameters, this would be a good solution, but with a large number of parameters of which some are conditional, it would became too complex to manage. Also, this way, maintaining backwards compatibility would become a burden if some parameters became irrelevant.

Therefore, there had to be another way to recognize the values from each other. Since the CSV format had to be designed to be able to contain exactly the same data JSON format can contain, it was only natural to look for the solution in the JSON format itself. This problem was resolved by implementing similar key-value pair scheme used in JSON to the CSV format.

Despite of all the additional info added to the CSV format, the format did not change by much. In the end, the only change was made to the very first cell. While in the old format, the cell could contain the word 'time' or a number, in the new format it could contain the word 'time' or a whole bunch of buffer metadata given in key-value pairs. The keys and values were be separated by colon and the pairs with a blank space from each other. The complete specification for the new CSV format can be seen in Appendix 3.

4.7 Communication Format

Eventually, the need for data subscription became unnecessary or at least there were no plausible use cases for it. It was decided that PowerSpy would not have data subscription and the data acquisition part of the communication between the server and client was redesigned and rebuilt to use AJAX requests instead. One use case could have been combining the requests for multiple buffers at the same time in which case the buffers would have been sent to the client one by one once they arrive. This was, however, not a enough of a reason to maintain more complicated communication format.

5 DISCUSSION

5.1 Fast Fourier Transform API Selection

After examining the features of the selected APIs and after making a comparison against the given restrictions, the selection process became a choice between two satisfactory options: the Nayuki's API and the dsp.js. Both were computationally effective and easy to use. Neither of them would add extra dependencies and in fact adding them to PowerSpy would mean that the developer would only be dealing with one file in the future in both cases.

However, the superior performance of the dsp.js along with the window functions, active user base and more active development lead to it being selected to be integrated in PowerSpy. Also, it was the only algorithm that met all the given requirements.

5.2 Role Based Access Contral

Making the changes in the RBAC authentication process made the application more secure by not allowing everybody who have access to the application to have also access to the data of the devices. While acquiring data is not itself that dangerous, there is always a risk of malicious user managing to infiltrate the system and have access to a valid RBAC token which can then be used to control the FGCs. If one can control the FGCs, one can control the LHC.

Not only did changing the RBAC scheme make the application more secure, it also simplified the data transfer and significantly shortened the average request handling time. The simplicity and speed resulted from the fact that the server could now directly forward the request without the need to acquire a RBAC token. The speed up with a mean of 228 ms was very noticeable especially when device state queries were made. The response from the server came practically instantly.

5.3 Back-End Optimization

PowerSpy needs to send requests to the devices frequently and originally the name file had to be parsed again for each request. According to the study results, by far the best method is to include the needed information in the request. This saves up to about 300 ms of time with a median of 282 ms. This corresponds to from 0.5% in the worst case to over 50% in best case performance improvement. The difference is best noticed when only a small amount of data is transferred (e.g., requesting the state of a device). Depending on the type of the request, the range of the speed-up is from 0.5% in the worst case to over 50% in the best case.

5.4 JavaScript performance

There were no particular surprises in the results of JavaScript study. It is a common consensus that libraries tend to be slower than native code due to the fact that the libraries usually run more code while operating. Nevertheless, event handling in jQuery turned out to perform better than the JavaScript native function, although the difference is only a few milliseconds.

Also, according to the results, using object prototypes is a faster solution compared to using private functions. This was expected, since, for example, by creating thousand objects, one would create only one function in the case of a prototype function and a thousand functions in the case of private functions. That is to say, objects having the same class share common prototype functions, but each would still have their own private functions. Creating and operating a high number of private functions uses computing and memory resources and would, therefore, use more time.

5.5 Data Formats

JSON was adopted as the data format for communication. There were no differences in the transfer time, but the difference in parsing time is significant. JSON is approximately 4 times faster than CSV in parsing. A small amount of the differ-

ence is due to the fact that the data size is so much smaller in the case of JSON. The bigger part of the difference is explained by the fact that the JSON format were originally designed to be able quickly read into JavaScript.

The parsing time was not correlating with the number of signals or with the number of samples. However, there was a repeating lag in the tests which appeared in cycles after certain amount of data had been processed. This lead to the hypothesis that the lag might have something to do with the memory memory of the JavaScript engine of FireFox but the issue was not further studied.

Also, a new specification for the CSV format was requested. The definition was that it should be able to accomplish whatever JSON can do and it should look similar in PowerSpy. Also, the data format should be exportable to a spreadsheet software. This was accomplished by overloading the very first field in the CSV file. Therefore, if the data were exported to a spreadsheet software, clearing one cell would get rid of the unneeded metadata.

5.6 Communication Format

Eventually, the already implemented SSE protocol was changed to AJAX Request. This was expected since the SSE is only useful if there is a need for a subscription to data. However, the PowerSpy was never going to be an actual real-time data viewer due to the reasons explained in Section 1.3. Typically AJAX would add network overhead compared to SSE, but if SSE was going to be used in a similar fashion (get data once, close connection), the difference is negligible. Also, SSE is less known than AJAX which could be a minor maintainability issue.

6 CONCLUSION

In this thesis, a new high-performance real-time power converter data browser, PowerSpy, was introduced. The thesis addresses the issues with the predecessors of the software as well as the current solution. The software was analyzed considering the used data formats, communication formats, security measures and programmatic solutions and the performance-wise weak spots were identified and new solutions were introduced. The solutions were benchmarked and compared against the original. The development solutions were then made based on the results of the experiments. The suggested solutions significantly improved the overall performance of the software and resulted in a faster application.

Since FFT is a crucial tool in development of FGCs, an JavaScript FFT API benchmark was also done. A bunch of APIs were compared against given requirements for the FFT API. The criterion included computational performance, ease of use, community and support, whether or not the API adds extra dependencies, and whether it had a free licence or not. The best option was dsp.js and it was implemented in PowerSpy.

REFERENCES

[1] IOLA.
http://www.iola.dk/
(accessed:13/05/2017).

[2] Attractive JavaScript Plotting for jQuery.
http://www.flotcharts.org/
(accessed:13/05/2017).

[3] GitHub - Flot Tooltip.
https://github.com/krzysu/flot.tooltip
(accessed:13/05/2017).

[4] S. Steinarsson. Downsampling Time Series for Visual Representation. *Master's Thesis, University of Iceland*, 2013.

[5] Interactive JavaScript Charts for Your Webpage.
https://www.highcharts.com/
(accessed:14/05/2017).

[6] B. Nolan. Behaviour.
http://www.bennolan.com/behaviour/
(accessed:13/05/2017).

[7] The jQuery Foundation. jQuery.
https://jquery.com/
(accessed:13/05/2017).

[8] D. K. Taft. jQuery Eases JavaScript, AJAX Development, 2006.
http://www.eweek.com/development/jquery-eases-javascript-ajax-development
(accessed:13/05/2017).

[9] Less.js - Getting Started.
http://lesscss.org/
(accessed:13/05/2017).

[10] Bootstrap - The World's Most Popular Mobile-First and Responsive Front-End Framework.
http://getbootstrap.com/
(accessed:13/05/2017).

[11] C. Campbell. Mousetrap - Keyboard Shortcuts in Javascript.
`https://craig.is/killing/mice`
(accessed:13/05/2017).

[12] UNINETT. An Apache Module With a Simple SAML 2.0 Service Provider.
`https://github.com/UNINETT/mod_auth_mellon`
(accessed:28/04/2017).

[13] R. J. Steinhagen. Tune and Chromaticity Diagnostics. 2009.
`https://cds.cern.ch/record/1213281/files/p317.pdf`
(accessed:21/05/2017).

[14] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965.

[15] C. Rader and N. Brenner. A New Principle for Fast Fourier Transformation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 24(3):264–266, Jun 1976.

[16] R. Preuss. Very Fast Computation of the Radix-2 Discrete Fourier Transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 30(4):595–607, Aug 1982.

[17] P. Duhamel and H. Hollmann. Split Radix FFT Algorithm. *Electronics letters*, 20(1):14–16, 1984.

[18] S. G. Johnson and M. Frigo. A Modified Split-Radix FFT With Fewer Arithmetic Operations. *IEEE Transactions on Signal Processing*, 55(1):111–119, Jan 2007.

[19] R. N. Bracewell. The Fast Hartley Transform. *Proceedings of the IEEE*, 72(8):1010–1018, Aug 1984.

[20] R. V. L. Hartley. A More Symmetrical Fourier Analysis Applied to Transmission Problems. *Proceedings of the IRE*, 30(3):144–150, March 1942.

[21] H. S. Hou. The Fast Hartley Transform Algorithm. *IEEE Transactions on Computers*, C-36(2):147–156, Feb 1987.

[22] A. Ganapathiraju et al. Contemporary View of FFT Algorithms. In *Proc. of the IASTED*, pages 130–133, 1998.

[23] A. Saidi. Decimation-In-Time-Ffrequency FFT Algorithm. In *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, volume iii, pages III/453–III/456 vol.3, Apr 1994.

[24] G. Haitao et al. The Quick Fourier Transform: an FFT Based on Symmetries. *IEEE Transactions on Signal Processing*, 46(2):335–341, Feb 1998.

[25] M. Soni and P. Kunthe. A General Comparison of FFT Algorithms. *Pioneer Journal Of IT & Management*, 2011.

[26] M. Balducci et al. Benchmarking of FFT Algorithms. In *Southeastcon '97. Engineering new New Century., Proceedings. IEEE*, pages 328–330, Apr 1997.

[27] R. S. Sandhu et al. Role-Based Access Control Models. *Computer*, 29(2):38–47, Feb 1996.

[28] A. D. Petrov et al. User Authentication for Role-Based Access Control. *ICALEPCS'07 proceedings*, 2007.

[29] S. Gysin et al. Role-Based Access Control for the Accelerator Control System at CERN. *K. Kostro et al., Role-Based Authorization in Equipment Access at CERN*, 2007.

[30] R. T. Fielding. Architectural Styles and the Design of Network-Based Software Architectures. *PhD Thesis, University of California, Irvine*, 2000.

[31] The Internet Engineering Task Force. Common Format and MIME Type for Comma-Separated Values (CSV) Files, October 2005.
`https://tools.ietf.org/html/rfc4180`
(accessed:14/05/2017).

[32] Ecma International. ECMA-404 The JSON Data Interchange Standard, Oct 2013.
`http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`
(accessed: 02/04/2017).

[33] I. Hickson. Server-Sent Events. Candidate Recommendation, W3C, Feb 2015.
`https://www.w3.org/TR/eventsource/`
(accessed: 15/02/2017).

[34] J.J. Garrett. Ajax: A New Approach to Web Applications, 2005.
https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php
(accessed:08/05/2017).

[35] A. Bersvendsen. Event Streaming to Web Browsers.
https://dev.opera.com/blog/event-streaming-to-web-browsers/
(accessed: 15/02/2017).

[36] Mozilla Developer Network. EventSource.
https://developer.mozilla.org/en-US/docs/Web/API/EventSource
(accessed:28/04/2017).

[37] Creative Commons. Creative Commons Attribution Share Alike License.
https://creativecommons.org/licenses/by-sa/3.0/
(accessed: 02/04/2017).

[38] T. Van Veen. What Are Long-Polling, Websockets, Server-Sent Events (SSE) and Comet?
http://stackoverflow.com/questions/11077857/what-are-long-polling-websockets-server-sent-events-sse-and-comet
(accessed: 02/04/2017).

[39] Vail Systems. Pure Node.js Implementation of the Fast Fourier Transform (Cooley-Tukey method).
https://github.com/vail-systems/node-fft
(accessed: 14/03/2017).

[40] C. Cannam. FFTs in Javascript.
https://thebreakfastpost.com/2015/10/18/ffts-in-javascript/
(accessed: 14/03/2017).

[41] D. Herman et al. asm.js: Working Draft, 18 August 2014.
http://asmjs.org/spec/latest/
(accessed: 15/03/2017).

[42] A. Zakai. Emscripten: an LLVM-to-JavaScript Compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 301–312. ACM, 2011.

[43] Mohayonao. Timbre.js.
http://mohayonao.github.io/timbre.js/
(accessed: 14/03/2017).

[44] Nayuki. Free Small FFT in Multiple Languages.
https://www.nayuki.io/page/
free-small-fft-in-multiple-languages
(accessed: 14/03/2017).

[45] J. Nockert. Discrete Fourier Transform in Javascript.
https://github.com/JensNockert/fft.js
(accessed: 14/03/2017).

[46] N. Jones. Small, Efficient Javascript FFT Implementation.
https://github.com/dntj/jsfft
(accessed: 14/03/2017).

[47] C. Brook. Digital Signal Processing for Javascript.
https://github.com/corbanbrook/dsp.js/
(accessed: 14/03/2017).

[48] M. Borderding. Kiss FFT.
http://kissfft.sourceforge.net/
(accessed: 14/03/2017).

[49] Security Assertion Markup Language (SAML) V2.0 Technical Overview, March 25, 2008.
http://docs.oasis-open.org/security/saml/Post2.0/sstc-
saml-tech-overview-2.0.html
(accessed:14/05/2017).

[50] R. Johnson et al. The Spring Framework–Reference Documentation.
https://docs.spring.io/spring/docs/current/
spring-framework-reference/htmlsingle/
(accessed:03/04/2017).

[51] Integrate SSO / RBAC on a Client.
https://wikis.cern.ch/pages/viewpage.action?pageId=
85363362
accessed:21/05/2017.

[52] jsPerf - JavaScript Performance Playground.
https://jsperf.com/
accessed:14/05/2017.

[53] Open Source Initiative. The MIT License.
https://opensource.org/licenses/MIT
(accessed: 02/04/2017).

[54] Open Source Initiative. The 2-Clause BSD License.
https://opensource.org/licenses/BSD-2-Clause
(accessed: 02/04/2017).

[55] Open Source Initiative. The 3-Clause BSD License.
https://opensource.org/licenses/BSD-3-Clause
(accessed: 02/04/2017).

[56] Realistic innerHTML vs. appendChild vs jQuery.append().
https://jsperf.com/realistic-innerhtml-vs-appendchild/15
(accessed:21/05/2017).

[57] getElementsByClassName vs jQuery selector vs querySelectorAll.
https://jsperf.com/getelements-vs-jquery-vs-queryselector/9
(accessed:21/05/2017).

[58] Bind vs addEventListener.
https://jsperf.com/bind-vs-addeventlistener
(accessed:21/05/2017).

[59] Prototype vs Function Performance.
https://jsperf.com/prototype-vs-factory-performance/2
(accessed:21/05/2017).

Appendix 1. FGCSpy CSV data format (old format)

These specifications here are originally presented in the CERN internal wiki.

FGCSPY data format is a CSV format used in FGCSpy. It was also initially used in PowerSpy, although it was quickly discovered to be insufficient and was replaced by the new format. The format is specified as follows:

```
TIME,...,SIGNALM_PARS
TIME1,SIGNAL1_VALUE1,SIGNAL2_VALUE1,...,SIGNALM_VALUE1
TIME2,SIGNAL1_VALUE2,SIGNAL2_VALUE2,...,SIGNALM_VALUE2
TIME3,SIGNAL1_VALUE3,SIGNAL2_VALUE3,...,SIGNALM_VALUE3
...
TIMEN,SIGNAL1_VALUEN,SIGNAL2_VALUEN,...,SIGNALM_VALUEN
```

in which `TIME` is the word "time" or an empty string, `SIGNALM_PARS` is signal parameter (for parameters, see Table A1.1) field for signal number M, `TIMEN` is time stamp for the Nth row and `SIGNALM_VALUEN` is the value of signal M on the row N.

Example: Analog Buffer

```
TIME,SIGNAL1 STEP, SIGNAL2
1458137212.000000, 10.1,    -5, 1
1458137212.000100, 11.5E+3, -3, 2
1458137212.000200,-12.2E-2,  1, 3
```

Table A1.1. Explanation for the old CSV signal parameters.

Signal parameters	Mandatory	Notes
name	Yes	The name of the signal.
step	No	If string "step" is found in the signal name field, trailing step interpolation is applied for this signal. Case-insensitive.

Appendix 2. PowerSpy JSON data format

These specifications here are originally presented in the CERN internal wiki.

The PowerSpy JSON format is mainly used as the only communication format between the server and the clients. PowerSpy, however, also supports import from a JSON file. The buffer parameters and signal parameters contained in the format are presented in Table A2.2 and Table A2.3 respectively.

Example: Analog Signal

```
{
  "version":"1.1",
  "type":"analog",
  "source":"fgc",
  "device":"SYSTEM_NAME",
  "name":"BUFFER_NAME",
  "cycleSelector":4,
  "timeOrigin":1458137212.000000,
  "firstSampleTime":1458137212.000000,
  "period":1.00000E-04,
  "signals":[
    {
      "name":"SIGNAL1",
      "samples":[10.1,11.5E+3,-12.2E-2]
    },
    {
      "name":"SIGNAL2",
      "step":true,
      "timeOffset":0.500000,
      "samples":[-5,-3,1]
    },
    {
      "name":"SIGNAL3",
      "timeOffset":-2.500000,
      "samples":[1,2,3]
    }
  ]
}
```

(continues)

Appendix 2. (continued)

Example: Digital Signal


```
{
  "version":"1.1",
  "type":"digital",
  "source":"ccrt",
  "device":"SYSTEM_NAME",
  "name":"BUFFER_NAME",
  "cycleSelector":21,
  "timeOrigin":1458137212.000000,
  "firstSampleTime":1458137212.000000,
  "period":1.00000E-04,
  "signals":[
    {
      "name":"SIGNAL1",
      "timeOffset":0.000100,
      "samples":[0,0,1]
    },
    {
      "name":"SIGNAL2",
      "samples":[1,0,1]
    },
    {
      "name":"SIGNAL3",
      "samples":[0,1,1]
    }
  ]
}
```


(continues)

Appendix 2. (continued)

Example: Event Log


```
{
  "version":"1.1",
  "type":"table",
  "class":"eventlog",
  "source":"FGC",
  "device":"SYSTEM_NAME",
  "name":"BUFFER_NAME",
  "table":  {
    "rows":[
      {
        "timestamp":1464722837.172000,
        "cells":["STATUS.ST_UNLATCHED","START_EVENT","SET_BIT","+"]
      },
      {
        "timestamp":1464722837.272000,
        "cells":["STATUS.ST_UNLATCHED","START_EVENT","CLR_BIT","+"]
      },
      {
        "timestamp":1464722837.242000,
        "cells":["STATE.PC","RUNNING","SET"," "]
      }
    ]
  }
}
```


(continues)

Appendix 2. (continued)

Table A2.2. Explanation for JSON buffer parameters.

Buffer parameters	Mandatory	Default value	Notes
version	Yes	-	PowerSpy JSON format version number.
type	No	"analog"	Buffer type: "analog", "digital", "table".
source	No	empty string	Data source (e.g., "FGC'). Web server always defines the source.
device	No	file name	If the data is coming from a local file, the default value will be the file name without the .csv suffix. Web server always defines the device.
name	No	empty string	The buffer name will appear in the monitor title. Web server always defines the buffer name.
cycleSelector	No	NONE	Number of the cycle, for cyclic buffers.
Signal buffer parameters	Mandatory	Default value	Notes
signals	Yes	-	Array of objects containing the signal data.
timeOrigin	No	firstSampleTime	Time origin for all the signals in the buffer in UTC Unix time.
firstSampleTime	No	0 (1/1/1970)	Time of first sample in the buffer in UTC Unix time.
period	No	1	The time between time values in seconds.
Table buffer parameters	Mandatory	Default value	Notes
class	Yes	-	Specifies the type of table (only "eventlog" currently supported).

(continues)

Appendix 2. (continued)

Table A2.3. Explanation for the JSON signal parameters.

Signal parameters	Mandatory	Notes
name	Yes	The name of the signal.
step	No	If string "step" is found in the signal name field, trailing step interpolation is applied for this signal. Case-insensitive.
timeOffset	No	If a numeric value is found in the field, it will be used as the time offset for the signals in seconds. A positive time offset will move the point later and a negative time offset will move it earlier.
samples	Yes	Array of signal samples in time order.

Appendix 3. PowerSpy CSV data format (new format)

These specifications here are originally presented in the CERN internal wiki.

Signal Data Format

PowerSpy CSV format is an extension to FGCspy CSV format, so PowerSpy can read FGCspy files. With PowerSpy, the first column of the header row is a space-separated list of key:value pairs matching the equivalent buffer parameter described for the designed JSON. The order is not significant. If the field is empty or contains the case-insensitive word "TIME", then all the parameters are set to their defaults (such as in FGCSpy CSV format). The buffer parameters and their default values are presented in Table A3.1.

In addition to the buffer parameters, the first line contains the signal names as well as the other possible parameters for each signal. The signal parameters are presented and explained in Table A3.2. Otherwise, the new format is exactly the same as the old format.

The format is specified as follows:

```
BUF_PARS, SIGNAL1_PARS, SIGNAL2_PARS, ..., SIGNALM_PARS
TIME1,SIGNAL1_VALUE1, SIGNAL2_VALUE1, ..., SIGNALM_VALUE1
TIME2,SIGNAL1_VALUE2, SIGNAL2_VALUE2, ..., SIGNALM_VALUE2
TIME3,SIGNAL1_VALUE3, SIGNAL2_VALUE3, ..., SIGNALM_VALUE3
...
TIMEN, SIGNAL1_VALUEN, SIGNAL2_VALUEN, ..., SIGNALM_VALUEN
```

in which `BUF_PARS` is the buffer parameter field, `SIGNALM_PARS` is the signal parameter field for signal number M, `TIMEN` is the time stamp for the Nth row and `SIGNALM_VALUEN` is the value of the signal M on the row N.

(continues)

Appendix 3. (continued)

Example: Analog Buffer

```
source:fgc device:SYSTEM_NAME name:BUFFER_NAME, SIGNAL1 +0.002
STEP, SIGNAL2 STEP
1458137212.000000, 10.1,    -5, 1
1458137212.000100, 11.5E+3, -3, 2
1458137212.000200,-12.2E-2,  1, 3
```

Example: Digital buffer

```
source:ccrt device:SYSTEM_NAME name:BUFFER_NAME type:digital,
SIGNAL1, SIGNAL2 -0.5, SIGNAL3 +1.5
1458137212.000000, 0, 1, 0
1458137212.000100, 0, 0, 1
1458137212.000200, 1, 1, 1
```

Table Data Format

The new format has also a possibility to store table data in files and present it in PowerSpy. The motivation is to present so-called buffer event logs in PowerSpy, but also regular tables and text files are possible to be presented in the application. The event log format is used by FGCs so only it will be presented here.

Eventlog has some extra table-related buffer parameters (namely, class:eventlog) and it does not have some parameters (e.g., period, time origin, first sample time), but otherwise it is similar to the signal data format. The format is specified as follows:

```
BUF_PARS, Property, Action, Value, Status
TIME1, PROPERTY1, ACTION1, VALUE1, STATUS1
TIME2, PROPERTY2, ACTION2, VALUE2, STATUS2
TIME3, PROPERTY3, ACTION3, VALUE3, STATUS3
...
TIMEN, PROPERTYN, ACTIONN, VALUEN, STATUSN
```

in which `BUF_PARS` is the buffer parameter field, rest of the first row are the

(continues)

Appendix 3. (continued)

headings for event log, `TIMEN` is time stamp for the Nth row and `PROPERTYN`, `ACTIONN`, `VALUEN` and `STATUSN` are the event log property, action, value and status data values for the Nth row, respectively.

Example: Event Log

```
type:table class:eventlog..., Property, Action, Value, Status
1464722837.172000,STATUS.ST_UNLATCHED,START_EVENT,SET_BIT,+
1464722837.272000,STATUS.ST_UNLATCHED,START_EVENT,CLR_BIT,+
1464722837.242000,STATE.PC,RUNNING,SET,
```

(continues)

Appendix 3. (continued)

Buffer and Signal Parameters

All buffer parameters are optional and the default values will be used in the absence of them. In the case of the signal parameters, only the signal name is a mandatory parameter.

Table A3.1. Explanation for the new CSV buffer parameters.

Buffer parameters	Default value	Notes
type	analog	Buffer type: "analog", "digital", "table".
source	"FILE"	Data source (e.g., "FGC').
device	file name	If the data is coming from a local file, the default value will be the file name without the .csv suffix.
name	empty string	The buffer name will appear in the monitor title.
cycleSelector	NONE	Number of the cycle, for cyclic buffers.
Signal buffer parameters	Default value	Notes
timeOrigin	TIME1	Time origin for all the signals in the buffer in UTC Unix time. TIME1 is the first time value in the series.
firstSampleTime	TIME1	TIME1 is the first time value in the series.
period	TIME2-TIME1	TIME1 and TIME2 are the first and second time values in the series.
Table buffer parameters	Default value	Notes
class	-	Specifies the type of table ("text" / "eventlog") if not a normal table.
columns	'l'	The styling for the columns and (valid only if class is not specified). L, C and R are for left, center and right alignment respectively. If the letter is uppercase, the column is bolded.

(continues)

Appendix 3. (continued)

Table A3.2. Explanation for the new CSV signal parameters.

Signal parameters	Mandatory	Notes
name	Yes	The name of the signal.
step	No	If string "step" is found in the signal name field, trailing step interpolation is applied for this signal. Case-insensitive.
timeOffset	No	If a numeric value is found in the field, it will be used as the time offset for the signals in seconds. A positive time offset will move the point later and a negative time offset will move it earlier.