ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPR008

WEB EXTENSIBLE DISPLAY MANAGER 2*

R. J. Slominski†, T. L. Larrieu, JLab, Newport News, USA

Abstract

The Web Extensible Display Manager (WEDM) was first deployed at Jefferson Lab (JLab) in 2016 with the goal of rendering Extensible Display Manager (EDM) control screens on the web for the benefit of accessibility, and with version 2 our aim is to provide a more general purpose display toolkit by freeing ourselves from the constraints of the EDM dependency. Over the last few years WEDM has been extensively used at JLab for 24/7 information kiosks, on-call monitoring, and by remote users and staff. The software has also been deployed to Oak Ridge National Laboratory, and has become more robust as many bug fixes and contributions have been added. However, adoption and utility of the software as a general purpose control system display manager is limited by EDM, which is no longer actively maintained. A new toolkit can be built on modern frameworks, fully embrace web conventions and standards, and support multiple control system data sources. This new version is a result of a technology review and selection, and introduces a web inspired display file format, a web based display builder, new widgets, and a data interface intended to support pluggable data.

INTRODUCTION

There has been an explosion of interest in control system displays on the web in the last few years. It has increasingly become a user expectation for displays to be available on a variety of devices including smart phones, and often the web is the best method to deliver this experience. Our initial version of Web Extensible Display Manager (WEDM) relied on the ageing Extensible Display Manager (EDM) to provide a screen builder tool and editable displays while we added a read-only web runtime [1]. The primary objective of version 2 is to break the dependency on EDM and provide an independent web based display builder. The second version of WEDM is named Puddysticks, and is shown in Fig. 1. During development we studied and embraced modern software frameworks, adopted web paradigms, and implemented a data source agnostic interface to widgets. We report our status as we work towards a web based display manager that serves all devices, whether they be in the control room or in your pocket.

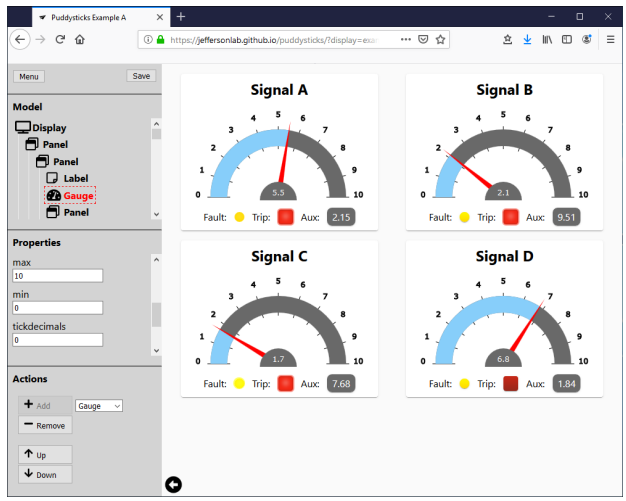

Figure 1: "Puddysticks" prototype.

A Case for Zero Code

Control systems have a long history of display managers: Graphical User Interface (GUI) builder tools that allow non-programmers to create control screens without writing any code. A display manager provides a consistent and familiar workflow to users, and the reusable framework saves software developers from having to spend time on each new project requiring displays.

A Case for Mobile

According to the Pew Research Center 76% of people in advanced economies had a smart phone in 2018 [2]. Using mobile devices with control systems is an opportunity for improved user experience. Remote monitoring of control systems from desktops may not even be an option for many as users drop home broadband in favor of cellular Internet and discontinue personal ownership of desktops. Besides ensuring staff are always connected to the control system wherever they are, mobile devices also provide cheap and readily available access to numerous additional user interaction mechanisms such as touch gestures (haptics) and voice commands. Readily available built-in cameras and GPS may also provide new opportunities. The best user experience might be for operators to adjust control system settings via touch or voice, instead of mouse and keyboard.

A Case for the Web

The web is a well-established standardized cross-platform way to provide interactive content on nearly all devices, including mobile devices. Web applications can work well on devices of varying screen sizes, but do not do so implicitly, and those that do are often referred to as having a responsive design. Going a step further, Progressive Web Applications (PWAs) are web applications that follow a set of best practices making them

* Authored by Jefferson Science Associates, LLC under US DOE Contract № DE-AC05-06OR23177
† ryans@jlab.org

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPR008

behave like cross-platform native applications [3]. Features of PWAs include fast loading times, user notifications, device home screen integration, and offline capabilities to ensure functionality even with network disconnection.

Alternative methods for deploying software to heterogeneous devices is to create a separate native application for each device type, or use a portable user interface (UI) framework. Creating a native application for each device type is costly to build and maintain, but provides the most flexibility. Portable UI frameworks are less commonly used than the web and require users to install software. Google supports the web approach to cross-platform applications with its PWA initiative, but Google also is supporting portable UI framework solutions with its product Flutter. Other portable UI solutions include Facebook's React Native, Ionic, and Xamarin. If a native environment is required, web technologies can actually be packaged as native applications, using various tools like Electron (desktop only), Cordova, and NW.js. Packaging web apps as native apps is one way to access a native-only Application Programming Interface (API). However, the main advantage of the web solution is that it does not require users to install an application on their device and it is more mature and familiar than proprietary portable UI frameworks.

RELATED WORK

Custom coding web applications from scratch is time consuming, but there are strategies for speeding up control system web development. One strategy for leveraging the web without the cost of developing a new display manager is to simply capture and store controls data into a format already understood by off-the-shelf web visualization tools. This is the approach taken by Belle2 [4], which stored EPICS data into a popular time series database InfluxDB, and then displayed the data using the popular Grafana web dashboard interface. This approach is reported as Control System Studio's main visualization strategy going forward. However, it may not be the most storage efficient way in the general case as it may result in data being duplicated in a control system archiver, plus an extra time series database instance. Further, the Grafana interface is limiting as a general purpose controls display. A similar project exists at SuperKEKB [5], but instead of duplicating data in a time series database a custom pvAccess gateway is used. However, the gateway used is not general purpose and is narrowly focused targeting a specific client (Grafana), and specific data (Control System Studio alarms). Another strategy for speeding up web development is to leverage a web application framework, of which there are many to choose from.

The web is pervasive, and it is clear that users expect to be able to interact with control screens on the web, just like with most everything else today. The web and mobile have obvious appeal and huge demonstrated interest as shown in recent projects from the previous ICALEPCS such as PWMA [6], DashBuilder [7], BNL HTTP Services [8], NICA Web Access [9], Tango Web Report [10], MXCuBE3 [11], and MAX IV Web Services [12]. More recently the Phoebus Display Builder Runtime [13] was created, and provides a web runtime to Control System Studio, similar to how WEDM version 1 provides a web runtime for EDM.

TOWARDS A BETTER DISPLAY MANAGER

Challenges on the Web

Web technologies provide an established and powerful application environment, yet unique challenges arise. Security is a top priority on the web unlike in traditional Supervisory Control and Data Acquisition (SCADA) environments, and robust solutions already exist for securing web applications. However, neither portable UI frameworks nor web applications are as flexible as a native application because of sandboxing. Sandboxing provides security: the trade-off is increased security for limited device access. Unlike a native application, a web application has neither arbitrary access to a device file system nor unlimited privilege to create network connections to remote servers. This is generally a good thing. The actions web applications can do are carefully curated, but still include powerful features such as obtaining geographical location (GPS, cellular, and by IP), using device camera and microphone, playing sounds and videos, storing files in a sandboxed area of a device file system, executing hardware accelerated graphics, reading local files explicitly allowed by users, and connecting to other servers explicitly requested by users. One ramification of this sandboxed environment with respect to display managers is that arbitrary local scripts cannot easily be executed in the same way previous display managers have historically allowed. A better approach which works with the web, is a client-server model, in which clients make requests of servers to execute actions on their behalf. This avoids the issue of relying on the existence of a script stored locally often requiring a shared file system.

User vs Developer Ergonomics

There are numerous modern display managers such as PyDM [14], Control System Studio [15], Inspector [16], Radar [17], and Taurus [18], and these are just the ones discussed at the previous ICALEPCS (two of which are from CERN). However, the existing display managers are not designed specifically to be portable to various devices. Further, many display managers differentiate themselves based prominently on which programming language they are implemented in. Since the idea with "zero code" display managers is that users do not have to write any code, the coding language is less important to end user experience. However, developer ergonomic considerations are not without value, and the ease with which software is maintained and enhanced is important. Ideally, technology choices satisfy all parties.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPR008

Selected Technology

There is an overwhelming amount of choice when it comes to ways to implement a web display manager: many frameworks and many technologies to choose between. It is a considerable amount of research to determine the state of web technology. We considered technologies with respect to the basic requirements of user and developer ergonomics matching that of existing display managers plus the ability to build and access displays portably. The ideas and technologies we chose include a compiled reactive framework, the concept of responsive design, and the idea of pluggable structured data sources.

Reactive programming? Not so fast! Reactive programming is popular for user interface development because it allows developers to use concise declarative statements instead of verbose and potentially more error prone imperative programming. The reactive programming paradigm is convenient because developers do not need to think as much about explicit state changes or control flow and instead can focus on program logic. For example an imperative implementation of a text input widget would require registering a change event listener and implementing a call-back method to store the updated input state into a variable. With reactive programming the programmer could instead simply indicate that a text input is bound to a variable and the variable state would then be automatically updated. The advantage is simpler and smaller code. As another example if that same variable content was then to be displayed on a page imperatively a developer would need to interact with the Document Object Model (DOM) API to splice the value into the document as changes occur. Reactively the developer could instead provide a page template which indicates where the variable is inserted and the page would be automatically updated as the variable changes.

Many popular frameworks like React and Vue leverage a Virtual DOM and runtime to provide reactivity, but at a cost. The Virtual DOM provides a declarative interface to work with, abstracting away the real DOM in which changes are made imperatively. The problem is web clients must download large runtimes and must execute DOM reconciliation routines plus other runtime code resulting in unnecessary work and slow page load times. The solution: compiled code. Compile time reactive frameworks such as Svelte and Solid do not use a Virtual DOM and instead compile developer friendly reactive code to optimized imperative statements that provide all of the developer benefit with minimal runtime cost and therefore a user friendly experience as well.

Responsive Web Design The idea with responsive web design is to create a single web page that works on any device, instead of creating a separate display per device. Display managers historically have used an absolute layout in which the display size is fixed and all widgets on the display have unchanging size and position. With responsive layout the display size varies resulting in changes in component size and position. From a display builder perspective you no longer specify x and y coordinates and instead think semantically about what you want to display and let the display manager software deal with layout.

A responsive web application leverages techniques like text wrapping, scaling, dynamic grid layouts, media queries, and flowing/hiding content off screen via scrollbars, carousels, cards, or tabs. Web browsers contain powerful layout engines that provide complex responsive layouts using Cascading Style Sheet (CSS) rules such as grid and flexbox. Dynamic grid layouts change the number of columns based on available screen space and flow content off screen. For example a mobile device will generally use a single column, while a desktop application could have a dozen columns. Off screen content is frequently accessed via vertical scroll, but could also be accessed via horizontal swipe or could be accessed via tabs or links. CSS Media Queries allow style rules to be applied conditionally based on device characteristics such as screen dimensions or device type. Scaling is another technique for responsive design, but must be used carefully because widget size must remain readable on all devices and making widget size extra-large on large devices may not be useful. Scaling is best used for maps or diagrams in which geospatial location is important and users can zoom to the level of detail they are interested in.

Pluggable Data Widgets can be decoupled from their data source using the mediator design pattern. With the mediator pattern an intermediary component interfaces with the widget and the data source such that widgets and data sources need not be aware of one another. Widgets and data sources can be created independently and integrated using a mediator component. A web display manager with a pluggable data source interface can be extended to work with any data source, including a relational database proxy or an EPICS 7 pvAccess proxy.

PUDDYSTICKS

The second version of WEDM is named Puddysticks, which means “easy”, because we believe it makes displays easy to build and use. The source code, documentation, and a demo can be obtained from GitHub [19]. The prototype software provides a web based display builder tool, and an initial basic set of read-only widgets including a Panel, Label, Indicator, and Gauge. The proof-of-concept creates and consumes display files in JavaScript Object Notation (JSON) format, and is built on Svelte, a reactive compile time library.

Overview

The proof-of-concept currently supports viewing, editing, and saving display files in a new format created for Puddysticks with extension *.puddy*. These files contain the JSON model that make up a display, and loading and storing this model is simple and efficient since JSON can be converted to and from JavaScript objects natively. Currently files can be opened locally or from a remote server and can be saved locally. A sidebar can be toggled into view and provides a tree control for examining and manipulating the structure of the display file. Nodes in the tree represent widgets, and they can be selected to reveal

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPR008

their properties. Selected widgets can be re-ordered and removed, and if a container widget named Panel is selected then other widgets can be added to it. Properties are exposed in input boxes on a form and changes typed into the input boxes update the display in real time providing familiar What You See Is What You Get (WYSIWYG) capability. New widgets can be added by creating a new Svelte component and updating the registry file to reference it.

Data Sources

There are currently three data sources provided, a Static data source that allows hard-coding fixed values, a Random Number Generator data source that is useful for testing and demonstration purposes, and finally an epics2web data source provides live EPICS data. The epics2web Channel Access Web Socket proxy server is reused from WEDM version 1. Data sources can be added by creating a data source component, updating the registry file to reference the new data source, and adding a data provider implementation for each widget that is to consume data from the new data source. The data provider is the mediator in the mediator design pattern. It allows a widget and its data source to be loosely coupled.

The API between a data source, data provider, and widget consists of JavaScript update events and two JavaScript objects: a configuration object and a data object. Each data source accepts a configuration object specifying metadata such as which channel to monitor. The data source is then expected to generate JavaScript events containing data objects. The job of data providers is to listen for events and translate data objects produced by a data source into data objects formatted for a particular widget. In the simple case the data object may contain a single property named value and the data provider may simply pass the data object along unchanged.

Styling and Theming

Controlling how widgets look and are arranged can be controlled via style and theme. Widgets can be styled individually by directly specifying CSS rules. In addition, a theme can be chosen to override default widget styles. The provided default theme follows Google's Material Design. The theme is a property of the top level Display widget. A template exists demonstrating using CSS grid layout to provide a responsive layout. Using this template, an example display renders well on both large and small screens as seen in Fig. 2.

Figure 2: Display with alternate theme on mobile device.

FUTURE WORK

The new display manager is far from complete. New components need to be added and new data sources created. There are also many exciting technologies to watch. At Jefferson Lab we have a test server, but have not setup a production server yet and we plan on obtaining feedback from users before moving forward.

Components

There are too many components missing at this time to replace a mature display manager, but Puddysticks can supplement an existing one until more components are available. To expedite component development it may be most effective to integrate an existing set of widgets instead of creating all new from scratch. This could be facilitated using Web Components. Currently there are no components for making changes to the control system in the prototype. When input components are added care must be taken to ensure control system security and authentication capabilities are in place as well.

Data Sources

Many options exist for data sources such as custom relational databases, directory servers, web services, and additional control system sources. Specifically, integration with TANGO and EPICS 7 pvAccess are logical future directions. The quickest path to EPICS 7 is perhaps to create a plugin for the proxy PV Web Socket as used by the Display Builder Web Runtime.

Technology to Watch

Web Components Having a standard set of reusable widgets is a well-known strategy for user interface design, but how to implement new shared web widgets is not without issue. The HTML standard itself defines a small set of standard widgets like button and checkbox that are recognized by every browser automatically, and Web Components are an attempt to provide a standard way to add to the built-in widgets. In 2012 the first draft of the Web Components specification was published, and it was motivated by a growing number of component libraries

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPR008

such as Google's Angular and Facebook's React. Today Web Components still are not fully realized and some of the original specifications have been dropped, such as HTML Imports and CSS /deep/ combinator.

The main problem with Web Components is they are not as good as mature JavaScript component frameworks in many ways. Further, to avoid repeating common code in each custom widget and to work around limitations with web components, developers end up factoring out common code, and in doing so end up creating their own framework anyway. A home-grown framework is one that is likely not as well polished as a popular JavaScript component framework. A third party framework may also provide other value, such as reactivity, theming, and alternate composition models. Concerns with Web Components are numerous and include slow evolution, lack of browser support, problematic styling of internal component structure, and unsupported SVG composition. Often a reactive framework can optionally generate either web components or plain JavaScript classes. Generating Web Components may be useful if you intend to share widgets outside of the project, but in practice web components often are tied to a particular framework ecosystem and are therefore not as easily sharable as one may expect.

Web Assembly Web Assembly (WASM) is a virtual machine bytecode format (not machine assembly as the name implies) for both client and server applications including web browsers (but not just for the web). The goal of WASM is to provide a portable and high performance compilation target for high level programming languages such as C++ and Rust. Web Assembly has a lot of potential, but is not generally a good replacement for JavaScript at this time. One of the big problems with WASM right now is that browsers do not include much in the form of standard libraries built-in, which means if needed, they must be downloaded by each client. JavaScript source code is generally smaller because it does not need to include basic libraries. Further, the interface between WASM modules and the DOM has historically been slow, cancelling out many speed improvements gained from streaming compilation and lower level code. Many of these issues are currently being resolved however and even now WASM is a good tool for a specific set of problems. Web Assembly may be the next big thing, or it may suffer the same fate as similar solutions from the past such as Java Applets and Flash, which ultimately were phased out.

WebAuthn Authentication on the web no longer requires remembering and typing a long password, often into a tiny phone screen. With WebAuthn, users can leverage their fingerprint, face, or hardware key inside a phone to login. Token-based password-less Single Sign On (SSO) has existed on local intranets using Kerberos and SPENGO, but now will be available over the Internet as well. Web services are often protected by access tokens, which OAuth will continue to provide, but now the login form can be replaced with a WebAuthn credential. For a web based display manager it would be dangerous to have control system write access always authenticated, because of the risk of losing your device. However, authentication could require escalation as is done with web application leaders such as Amazon – users can be read-only authenticated, but as soon as the user attempts to make changes they would be prompted to re-authenticate.

HTTP/2 Pushing data from a server to a client efficiently has long been done effectively with HTTP/1.1 and Web Sockets, but HTTP/2 may soon be an even better option. Historically Web Sockets were not supported in HTTP/2, but web browsers will soon add support [20]. Advantages of HTTP/2 include faster encrypted connection establishment, faster supporting resource file download, as well as multiplexed connections. Related technologies include Server-Sent Events (SSE) and HTTP/2 Push. The former is unidirectional and not supported by Internet Explorer, while the latter is only for resources such as .css and .js files. Another technology to watch is gRPC, which operates over HTTP/2 and leverages Google's binary Protocol Buffers to provide an extremely efficient and portable protocol between applications. At this time gRPC is not natively supported in web browsers and browsers do not expose low level packet details to JavaScript clients so a JavaScript browser application cannot communicate at the application level with gRPC.

CONCLUSION

The Web Extensible Display Manager 2 proof-of-concept demonstrates a promising web based display manager. Basic functionality such as viewing and editing web displays have been shown. Many technologies were researched and tested before a selection was made, and ultimately three stood out: compiled reactive frameworks, responsive web design, and pluggable data sources. Compile time reactive frameworks provide both developer convenience and the performance users expect. Responsive web design allows a single display to be used on devices of varying screen size. Pluggable data enables choice when it comes to the data source connected to a widget. Not all technologies that we researched were used, and many such as Web Components and Web Assembly warrant our attention as technologies to watch. The new display manager is useful now, but further work remains to elevate the prototype to a fully production tool.

REFERENCES

[1] R. J. Slominski and T. L. Larrieu, "Web Extensible Display Manager", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 852-856. doi:10.18429/JACoW-ICALEPCS2017-TUPHA181

[2] Pew Research Center, February 2019, "Smartphone Ownership Is Growing Rapidly Around the World, but Not Always Equally."

[3] Progressive Web Applications, https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

[4] G. Tortone *et al.*, "Web Based Visualization Tools for Epics Embedded Systems: An Application to Belle2", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEMPR008

2017, pp. 1857-1860. doi:10.18429/JACoW-ICALEPCS2017-THPHA181

[5] S. Sasaki, "Web-Based Data Visualization Using EPICS pvAccess RPC at SuperKEKB", in *Proc. 15th Meeting of Particle Accelerator Society of Japan (PASJ'18)*, Nagaoka, Japan, Aug. 2018. THP099.

[6] L. Zambon, A. I. Bogani, S. Cleva, E. Coghetto, F. Lauro, and M. De Bernardi, "Web and Multi-Platform Mobile App at Elettra", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 984-988. doi:10.18429/JACoW-ICALEPCS2017-TUSH103

[7] T. D'Ottavio, P. S. Dyer, G. J. Marr, and S. Nemesure, "Creating Interactive Web Pages for Non-Programmers", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 976-978. doi:10.18429/JACoW-ICALEPCS2017-TUSH101

[8] T. D'Ottavio, K. A. Brown, A. Fernando, and S. Nemesure, "Building Controls Applications Using HTTP Services", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 1320-1322. doi:10.18429/JACoW-ICALEPCS2017-THMPA06

[9] G. S. Sedykh, V. G. Elkin, and E. V. Gorbachev, "Tango Web Access Modules and Web Clients for NICA Control System", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 806-808. doi:10.18429/JACoW-ICALEPCS2017-TUPHA167

[10] M. Broseta, A. Burgos, G. Cuni, D. Fernandez-Carreiras, D. Roldan, and S. Rubio-Manrique, "A Web-Based Report Tool for Tango Control Systems via Websockets", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17),* Barcelona, Spain, Oct. 2017, pp. 826-829. doi:10.18429/JACoW-ICALEPCS2017-TUPHA173

[11] M. Oskarsson *et al.*, "MXCuBE3 Bringing MX Experiments to the WEB", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 180-185. doi:10.18429/JACoW-ICALEPCS2017-TUBPL05

[12] A. Milan-Otero *et al.*, "Usage and Development of Web Services at MAX IV", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 1826-1830. doi:10.18429/JACoW-ICALEPCS2017-THPHA170

[13] K. Kasemir, "Display Builder Web Runtime", EPICS Collaboration Meeting June 2019, https://indico.cern.ch/event/766611/contributions/3438289/

[14] G. S. Fedel, D. B. Beniz, L. P. Do Carmo, and J. R. Piton, "Python for User Interfaces at Sirius", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 1091-1097. doi:10.18429/JACoW-ICALEPCS2017-THAPL04

[15] K.-U. Kasemir and M. L. Grodowitz, "CS-Studio Display Builder", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 1978-1981. doi:10.18429/JACoW-ICALEPCS2017-THS303

[16] V. Costa and B. Lefort, "Inspector, a Zero Code IDE for Control Systems User Interface Development", in Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17), Barcelona, Spain, Oct. 2017, pp. 861-865. doi:10.18429/JACoW-ICALEPCS2017-TUPHA184

[17] O. O. Andreassen, R. M. Knudsen, and J. W. Rachucki, "Radar 2.0, a Drag and Drop, Cross Platform Control System Design Software", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 1873-1876. doi:10.18429/JACoW-ICALEPCS2017-THPHA185

[18] C. Pascual-Izarra *et al.*, "Taurus Big & Small: From Particle Accelerators to Desktop Labs", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017, pp. 166-169. doi:10.18429/JACoW-ICALEPCS2017-TUBPL02

[19] Puddysticks, https://github.com/JeffersonLab/puddysticks

[20] RFC 8441, https://tools.ietf.org/html/rfc8441