**General notes:**

Sentences: 3 or so sentences.

Lists for things like types we support

Some graphic.

**Configuration**

The server is configured using an XML and accompanying XML Schema document. The configuration file is separated into two sections. The first section describes how data is stored within the system. Data can be formatted as a list of individual fields or groups. Individual fields cover most primitive data types and can be of type int, bool, string or enum to name a few. Groups allow users to associate multiple related elements. Elements may be individual fields, or even other groups. By using the XML schema document, users can add further customization by defining custom groups.

The second section allows users to define functions for the server to preform. Functions contain a single string representing a path to an executable program. By pressing a button within the web UI, users can execute this function from anywhere. Users may also associate a list of variable fields with a function. By doing so, the contents of the fields will be passed to the function as command line arguments when called.

**Generation**

* Ruby
  + Nokogiri: used to validate xml against schema
  + Rgen: Modeling framework used for code generation
    - Reads in xml and builds model in Ruby code
      * Logic checker confirms that xml logically makes sense
    - Generation templates
      * Utilizes own DSL building upon ERB
    - Output is C++ code used to properly build the server with all data from xml
* (Flow chart diagram)

**Compilation/Execution**

* Based on the lightweight Mongoose webserver
* C++98/03 compatible (except testing)
* Autotools support
  + e.g.
    - autoreconf -if; ./configure; make -j10
* Plugin support
  + Can be anything, so long as it exports certain functions with C ABI

**Interaction**

There are three components you can use to interact with the server. First there is the web UI, it allows for interaction with the device much like a router’s web UI does. In addition to changing settings on the device, the interface also allows for invoking functions and provides documentation for using the web API. The other two components are APIs (web and local). The web API uses RESTful JSON communication, while the local API uses a C interface. The APIs both allow for the same functionality as the Web UI, but also allow for registering callbacks.

**Problem Description**

Some text here

**Pugin Architecture**

Some text here.