Tower DB

Application Overview and Design

Clark Ritchie, [clark@inveneo.org](mailto:clark@inveneo.org)

July 13, 2013

Table of Contents

Introduction 1

History 2

Features 2

Design 3

Core Technologies 3

Other Technologies 3

JavaScript Libraries 3

CSS 3

CakePHP Plugins 3

CakePHP Behaviors 4

Object Model 4

Note on Github Repository Architecture 5

SQL 5

Triggers 5

Radios Linking to Radios 5

Spatial Extensions 6

Stored Procedures 7

Limitations 7

Many Radios, One Switch, One Router 7

OpenNMS 7

# Introduction

Tower Database, or Tower DB, is a web application to aid in the design and deployment of long-range Wi-Fi networks. It is target user is technical staff at Wireless ISPs, also known as WISPs.

# History

Inveneo, a US 501c3 non-profit, started the Tower DB project in the summer of 2012. The project was given the internal code name of “Poundcake”, a term that is still found throughout the source code in comments and other project documentation.

In its original incarnation, Poundcake was to be a short-term project – several weeks of effort to produce a simple web application for use on a project to manage the deployment of computer and network hardware. Regrettably, Inveneo’s involvement in that project stopped before Poundcake was completed. Nevertheless Inveneo decided to continue development of Poundcake and repurpose the code for another project, with similar, yet different, requirements. The application was refocused to better support the deployment of Wi-Fi radios co-located on towers, and Poundcake was renamed Tower DB. Certain significant design decisions related to this repurposing are noted later in this document.

Throughout 2012 and into 2013, Tower DB continued to evolve as an application to aid in the design and management of large-scale wireless networks. Its primary function has since evolved to be:

1. **Network Overview** – Provide a high-level overview of the network, graphically depicting sites (or tower/Wi-Fi radio locations) and the connections (Wi-Fi links) between sites. Provide a high-level overview of the network’s status via integration with other network monitoring tools.
2. **Network Inventory** – Be the definitive source for a project, detailing what is installed where, with a means to capture assorted types of site meta-data, essentialy all the critical information necessary to provide long-term support of a network.
3. **Network Design, Deployment** – Provide tools to support the design and deployment of large-scale wireless networks.

# Features

Tower DB has evolved to support a great number of features, too many to fully enumerate in this document. Listed below are a few of Tower DB’s major capabilities.

* Multi-project support
* User authentication and authorization, with three permission levels (configurable by project)
* Graphical depiction of a network on a map
* Configurable site states (configurable by project)
* Link wireless radios by SSID
* Support for Point-Point and Point-Multipoint radio configurations
* Lightweight integration with OpenNMS for network monitoring
* Support for a diverse array of site meta-data
* IP Address planning and management
* Support for variable types of radios, routers, switches, antennas
* Workorder generation
* Device configuration file generation
* Produce site-specific equipment lists for deployments
* KML import/export
* Extensive configuration for items such as radio bands, SNMP versions, tower types, and so forth
* Report generation

# Design

Because Tower DB was initially intended to be a short-term project, the CakePHP application framework was selected primarily due to familiarity and ease of use. CakePHP was also highly compatible with an existing LAMP environment. Tower DB’s initial developer was on a short timeframe, and given the application’s original set of requirements, CakePHP seemed appropriate.

## Core Technologies

As of the writing of this document, some of the core technologies that Tower DB uses include:

* PHP 5.4.10
* MySQL 5.5.29
* CakePHP 2.3.5

## Other Technologies

Other dependencies that are bundled with the application include:

### JavaScript Libraries

* jQuery 1.9.1 – base jQuery JavaScript library
* jQuery-ui-1.10.3 - JavaScript library for user interface interactions, effects, widgets, and themes
* jQuery-ui-map-3.0 – A Google-developed JavaScript library for rendering maps
* jasny-bootstrap – JavaScript library for enhanced Bootstrap-like functionality

### CSS

For UI support, Tower DB uses:

* Twitter Bootstrap 2.3.2 – Provides the site’s base CSS
* Bootbox – Provides dialog boxes consistent with Bootstrap’s overall look and feel
* Bootswatch – Other CSS/theme files are likely from Bootswatch.com, a free Bootstrap-compatible theme site

### CakePHP Plugins

Tower DB leverages CakePHP’s plugin architecture. Some of the plugins used include:

* AjaxMultiUpload – A component to handle drag-drop file attachments
* Composer – For dependency management in PHP
* Less – Converts .less files into CSS without relying on Node.js or client-side parsing
* Uploader - Plugin that will validate and upload files through the model layer (may be depreciated)
* Utility – A collection of CakePHP utility components
* ExcelWriterXML – For generation of XML files that can be opened in Microsoft Excel
* DOMPDF – For generation of PDF files

### CakePHP Behaviors

Tower DB leverages other small pieces of open source functionality, including:

* Cryptable – A CakePHP behavior for on the fly encrypting/decripting of sensitive data

## Object Model

A graphical depiction of Tower DB’s object model can be found in the Github repository in the folder /etc/docs.

The primary objects in Tower DB include:

* NetworkSwitch – A NetworkSwitch[[1]](#footnote-1) is meant to represent a common layer 2/3 switch used in Ethernet networks.
* NetworkRouter – A NetworkRouter[[2]](#footnote-2) represents a common router used to connect different Ethernet networks.
* NetworkRadio – A NetworkRadio[[3]](#footnote-3) is meant to represent a Wi-Fi radio.
* Antenna – A NetworkRadio can be associated with one of the compatible antennas.
* NetworkDevice – The superclass for NetworkSwitch, NetworkRouter and NetworkRadio.
* Site – A site is the central item to which network devices are attached. A site can have zero to many NetworkRadios, zero or one NetworkSwitch and zero or one NetworkRouter. See comments in *Limitations* below.
* Project – A project is really a collection of sites (and network devices). Administrators grant user’s view/edit permission to access projects in the application.
* User – A user has access to one or more projects, and has view or edit permission to the elements within that project. Users can have mixed permissions (i.e. view permission on one project, edit permission on another). Users who are administrators have access to the all projects in the system, as well as all system configurations.
* [item]Type – Many objects in Tower DB have a corresponding “type”. The *type* of an object further defines the characteristics of that object, such as the manufacturer, the number of ports, watts consumed, and so on.
  + For example, there exists an object named SwitchType and an administrator may configure any number of valid kinds of switches in the Tower DB application. Which is to say, an administrator of Tower DB may have defined several types of switches, say an 8-port DC switch, a 16-port AC switch, and so on. These definitions are global in scope. Thus, when a NetworkSwitch is created, the user picks what kind of switch it is; the NetworkSwitch object has a corresponding SwitchType.
  + Other examples of the [item]Type model:
    - A NetworkRouter has a RouterType
    - A NetworkRadio has a RadioType
    - An Antenna has an AntennaType
    - And so forth…

## Note on Github Repository Architecture

Note that none of the aforementioned technology dependencies are configured in the Github repository as Git submodules. Doing this is of interest. Re-architecting the Github repository, and modifying the deployment process, has simply not been a priority.

# SQL

Tower DB makes use MySQL triggers and stored procedures for certain, specific pieces of functionality.

## Triggers

Perhaps the most non-standard (at least in terms of a CakePHP application) is the use of triggers.

Triggers are named in the form tr\_trigger\_name and can be fond in the folder /etc/sql/poundcake.

Two specific triggers are worth noting:

### Radios Linking to Radios

One of the first significant design challenges that Tower DB encountered was how to link Wi-Fi radios to other Wi-Fi radios in the application. At best it was unclear if objects in CakePHP could relate to themselves (e.g. NetworkRadio hasMany NetworkRadio). Tower DB needed to support 1) point-point radio connections (NetworkRadio hasOne NetworkRadio), and 2) point-multipoint radio connections (NetworkRadio hasMany) NetworkRadios).

The approach of representing a Wi-Fi link as a full CakePHP object unto itself (which would imply NetworkRadio hasMany Link and Link hasMany NetworkRadio) was considered. Indeed one could argue there is great value in taking this approach as the properties related to that radio link, such as frequency and SSID, could then be attached to the link and not the radio. However, this approach was not accepted due to the intangible nature of a Wi-Fi link.

The approach that was taken was to employ the use of MySQL triggers (for each of insert, update, delete) to manage a join table. This approach is non-standard, yet fairly consistent with how CakePHP might manage this relation. This, when a radio is created, edited or deleted, a trigger updates the join table accordingly.

Note:

* radios\_radios is the name of the join table;
* Two radios are linked if their SSID matches. No other link properties, such as frequency, are required for radios to be linked.

#### Point-Point link example

To link two radios in a simple point-point link, simply set them to the same SSID.

For example, two NetworkRadios exist with the CakePHP IDs of 9 and 13, and share the same SSID. The join table (again, the join table is managed by the insert/update/delete triggers on the network\_radios table), would look as such:

mysql> select src\_radio\_id, dest\_radio\_id from radios\_radios where src\_radio\_id=9 or dest\_radio\_id=9;

+--------------+---------------+

| src\_radio\_id | dest\_radio\_id |

+--------------+---------------+

| 9 | 13 |

| 13 | 9 |

+--------------+---------------+

#### Point-Multipoint link example

To link three or more radios in a point-multipoint configuration, set each radio to the same SSID and denote which radio in the group is the ‘point end’ of a the multipoint link by setting the multipoint property to true on one NetworkRadio itself. This is needed to avoid all radios linking to all radios; Tower DB has to know which radio is the center of the multipoint linkage.

For example, three NetworkRadios exist with the CakePHP IDs of 310, 263 and 298, and share the same SSID. The join table would look as such:

mysql> select src\_radio\_id, dest\_radio\_id from radios\_radios where src\_radio\_id=263 or dest\_radio\_id=263;

+--------------+---------------+

| src\_radio\_id | dest\_radio\_id |

+--------------+---------------+

| 310 | 263 |

| 263 | 310 |

| 298 | 263 |

| 263 | 298 |

+--------------+---------------+

The three triggers to link radios are named tr\_network\_radio\_insert, tr\_network\_radio\_update and tr\_network\_radio\_delete, and are defined in /etc/sql/poundcake/ tr\_multipoint-links-SSID-key.sql.

### Spatial Extensions

Early during the development of Tower DB, the use of MySQL spatial extensions was explored. It was hoped these extensions might make related geospatial calculations, such as computing the distance between two coordinates, easy.

Ultimately, it was determined that using MySQL spatial extensions with CakePHP was more trouble than it was worth. Specifically, encoding and decoding latitude/longitude fields between the application and database, proved to be cumbersome.

Nevertheless, in the hope that MySQL spatial extensions might someday prove useful, it was decided to create a separate table, named *locations*, and use triggers on the sites table to keep the GPS coordinates – which are decimal(17,14) – in sync with the spatially encoded versions of those coordinates.

The three triggers tr\_location\_insert, tr\_location\_update, tr\_location\_delete and are defined in /etc/sql/poundcake/tr\_location.sql.

Note that the locations table is not currently used.

## Stored Procedures

Tower DB also uses a limited number of stored procedures for several small tasks. Stored procedures are named in the form sp\_procedure\_name and can be fond in the folder /etc/sql/poundcake.

# Limitations

## Many Radios, One Switch, One Router

When Tower DB was repurposed from an application to manage the deployment of hardware to something targeted at WISPs, it was redesigned with a specific project in mind. Every site on that project was to have one router, plugged into which would be one switch, plugged into which were multiple radios.

Therein lies one of Tower DB’s single biggest design limitations; the underlying model was setup to specifically support that scenario, e.g.

* Site belongsTo NetworkRouter / NetworkRouter hasOne Site
* Site belongsTo NetworkSwitch / NetworkSwitch hasOne Site
* Site hasMany NetworkRadio / NetworkRadio belongsTo Site

Since that time, Tower DB has been used on a project where not every site has a switch (radios plug directly into the router). Other projects have wanted to model physical connections, which are not possible given the current design.

Future revisions of Tower DB must address this significant design limitation.

## OpenNMS

Tower DB’s integration with OpenNMS is extremely primitive, and should be considered proof of concept. Currently, the following operations are possible.

* Provisioning – Provisioning an item (radio, router, switch) from Tower DB into OpenNMS (as an OpenNMS node) for network monitoring is functional but lacks in robust error checking. For example, items can be repeatedly provisioned with each provision creating a new node with a new unique Foreign ID.
* Rendering OpenNMS graphs in Tower DB is extremely simplistic due to OpenNMS not exposing graphing capabilities via its ReST API.
* Alarms are functional but only displayed when a user clicks on Alarms.
* Configuring Tower DB to query OpenNMS remains a manual task. Tower DB’s ability to ascertain a node’s status is best done on current releases of OpenNMS due to enhancements in the OpenNMS ReST API.

1. The name NetworkSwitch was used because “switch” is a PHP keyword. [↑](#footnote-ref-1)
2. The name NetworkRouter was chosen because “router” is a CakePHP keyword. [↑](#footnote-ref-2)
3. The name NetworkRadio was chosen simply for consistency with NetworkSwitch and NetworkRadio. [↑](#footnote-ref-3)