

Thingy Specification

Background Information

For a long time now we've been trying to build a web-based database/application environment that would allow users to create their own data entry applications. This started with the MailForm, which evolved into the DataForm, which has since had dozens of features slapped onto it. Some of its limitations are: the database table is fully normalized which makes it very slow; the user interface is klunky due to many features being added without refactoring; it's limited to a single list, record view, record edit, screen; it's impossible to do complex apps that require multiple tables.

Then last year Martin Kamerbeek created SQL Form, which takes the idea to a whole new level. It added the ability to create your own tables, implement search, import existing tables, and more. However, it's done in such a way that doesn't make it very practical for lots of people to use. It can import a database, but it destroys it in doing so (removes keys, adds extra fields, etc). It allows the user to create a database, but in a not-very easy manner, because you have to match up form fields to database fields. It requires an external database from WebGUI for security reasons; since the app can import a database, and the user can create new tables on the fly, we didn't want the user to be able to accidentally destroy, overwrite, or break any existing WebGUI tables.

These were both good attempts, but they don't reach the goal of being able to **eliminate the developer*** when it comes to creating these sorts of data entry applications. We need to take this to the next level, and that means building a web based database application that a normal business user can use without having a developer get involved. They shouldn't have to know databases or SQL. They shouldn't have to know Perl, and ideally, they shouldn't even have to know HTML or CSS unless they really want to tweak the hell out of the look and feel their app.

*Going forward I want to eliminate the roles traditionally defined by a complex web site (web master, designer, developer, and administrator), and put the power of these functions in the hands of the content manager (our user). I'll go into much more detail about this at the WUC. However, this app alone could single handedly wipe out the developer role in most circumstances if we do it well. This combined with some other changes I have planned, will definitely wipe out the developer's role for everything except custom apps with very situation specific requirements.

Random Thoughts

Thingy might seem like a crazy name, but give it a shot. After running a half dozen names past a bunch of non-programmer business users this is the name that was liked best.

Though this is technically an evolution of the DataForm and SQLForm assets, don't think that this asset has to replace all the features of those two assets directly. Think of this as more of a cousin to those assets rather than a descendant of them. The Thingy is meant to be an easier DataForm with some of the power of the SQLForm. However, the SQLForm will remain more powerful and therefore should continue to exist on its own outside of the core. My intent is to eliminate DataForm and SQLForm from the core, but not from people's web sites. Let them continue using DataForm and SQLForm if it suits them. But going forward Thingy will be the asset supported by Plain Black. To help ease the transition from DataForm and SQLForm to WebDB we'll want to write some sort of conversion scripts, but these will not be part of the normal upgrade process. It will need to be a conscious choice that site owners make, whether to eliminate DataForm and SQLForm from their sites.

In several instances in the requirements section I'm going to provide some implementation details, which are normally not supposed to go into a specifications document. However, I wanted to give you

some insights about what I've been thinking, and also give you a place to start with your design process. Don't treat these as gospel, if you have a better idea, bring it to the table.

If you have ideas for things that I haven't included, please bring those ideas forward. I want to make this the best app WebGUI has ever had. Therefore, the more brains working on the problem, the better.

In addition to the Thingy, I think we'll probably want to add a "Contact Us" asset to WebGUI. I think one of the problems we've been having is that we force assets to do too many things. While this makes them powerful, it also makes them unwieldy. Though the Thingy could be a contact us form, I think we can build a contact us form better for that specific purpose.

Requirements

Thingy must adhere to the WebGUI Best Practices, and all javascript code should make use of the YUI toolkit from Yahoo, rather than using some other toolkit or building it from scratch.

Behind The Scenes

Thingy is going to need to be able to construct it's own database tables. However, in order to make it useful to your average WebGUI user, it needs to do this within the WebGUI database. In addition, in order to ensure database integrity it must not be possible for it to overwrite or otherwise manipulate other WebGUI tables, or future WebGUI tables that may be added. There should be a table called `thingy_things` that should look like this:

- `assetId` – the asset this thing is associated with
- `thingId` – a guid specifically for this thing, and `thingId` guid's must not allow '-' (dashes) in them
- `label` – a human readable name for this table
- For each record in the `thingy_things` table, there should be a new table created called "`thingy_[thingId]`" that maps one for one with the record. For example:
`thingy_ax3jhK39n_sJ4HByS1oPCx`

The asset will also need to be able to add/remove/modify fields in these tables dynamically. There should be a table called `thingy_fields` that should look like this:

- `assetId` – the asset this field is associated with
- `thingId` – the unique instance id for this Thing
- `dateCreated` – the date/time the instance of the Thing was created
- `createdBy` – the `userId` of the user that created this thing
- `dateUpdated` – the date/time this Thing was last edited
- `updatedBy` – the `userId` of the user that last edited this thing
- `fieldId` – the unique id for this field
 - This should most likely be an integer which increments for each field added to the thing. No need to reuse the numbers if a field gets deleted.
 - Using integers also helps eliminate the possibility of screwed up or conflicting field names, like we have now in the SQL Form and Data Form.
- `label` – a human readable name for this field
- `fieldType` – the WebGUI form control type associated with this field
- Also add the other common fields from DataForm/SQLForm fields here like default value, options, size, extras, etc.

For each record in the `thingy_fields` table there should be a new field added to the `thingy_[thingId]` table. (NOTE: You should know that starting in 7.4, each form control will have an attribute attached to it which identifies which database field type should persist it's data.) The field will be called `field[X]`

where X is the id for the field. The reason to prefix it with “field” is because database fields cannot start with a number. Example: field12

The tables and fields as far as the humans are concerned will always be referenced by their human readable names.

User Interface

The user interface is the most important part of this asset. It has to be accessible to anyone that can wrap their brains around MS Access, and ideally, even those that cant.

While editing the thingy asset the user will be brought to a management screen that will list the current things defined in the thingy, and allow them to define new things.



If they click on the “X” next to any Thing they will be prompted whether they are sure they want to delete this Thing, and if they confirm the delete then it will delete the Thing all instances of the Thing, as well as any links to the Thing in any other Things. More about linking Things together when we talk about the edit screen.

If they click “view” they are taken to the search screen for that Thing. More about the search screen later.

If they click “edit” on a Thing, let's say “Employee” in this case, then they'll be brought to the Edit Thing screen, which enables them to define the fields for this thing and set parameters for various screens related to this thing.

The first thing you'll do when you get to this screen is name your Thing. In the example below we're editing an “Employee” Thing.

Edit Thing

Finished Editing

Fields

Edit Screen

View Screen

Search Screen

Thing Name

Employee

fields

First Name

X

Edit

|||||

Last Name

X

Edit

|||||

Phone Number

X

Edit

|||||

Favorite Color

red

green

blue

yellow

orange

X

Edit

|||||

drag

Department

Manufacturing

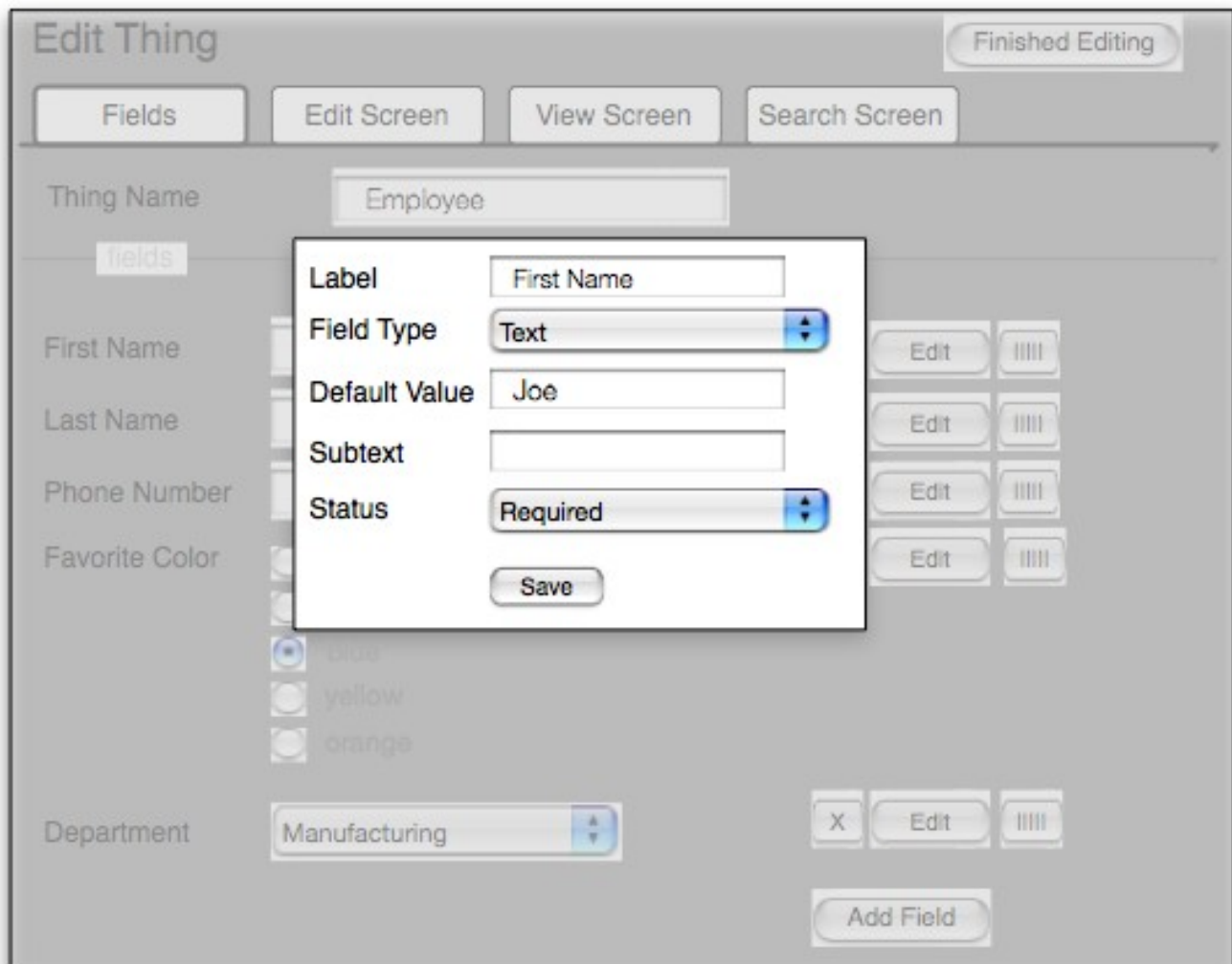
X

Edit

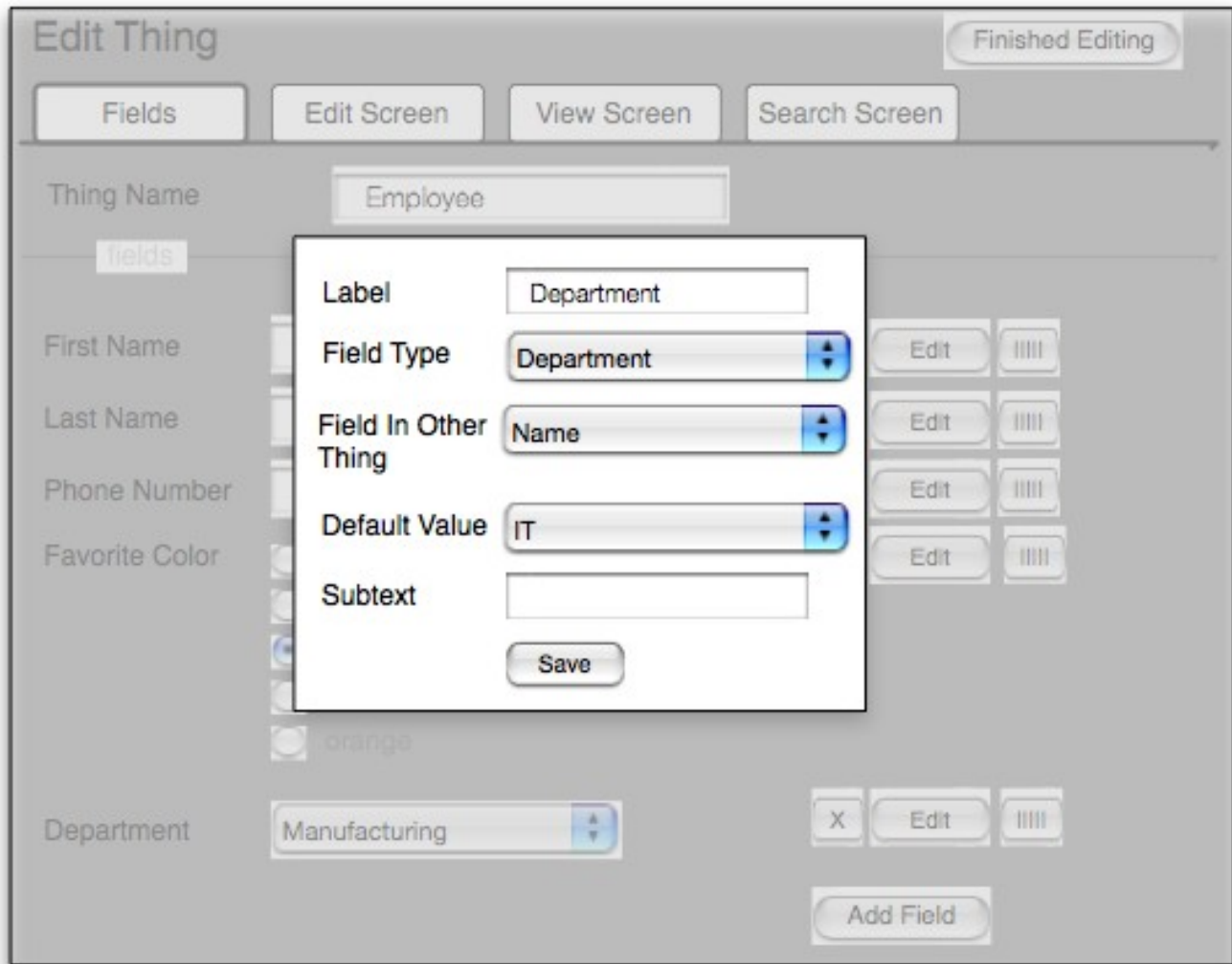
|||||

Add Field

Note that the user can drag the fields to change their order rather than having to use up/down arrows. This is then updated to the backend using ajax. Clicking “edit” or “add field” will pop up a modal dialog box where they can define the field. This too will be done with ajax so no to ever have the user leave their screen.



They type in the label, and then select the field type. Once the field type is selected the dialog will expand to include whatever other fields go with that field type. In most cases this will be the stuff you're used to like "Default Value", "Subtext", "Status", etc. However, there will also be a special field type. That field type will be for other "things" that have been defined, and will look like this:



In this case the field type is another Thing called “Department”. Then the user can select which field in the other thing to display, as well as choose a default value from all the possible choices in that particular field of the thing. This will require several calls back to WebGUI using ajax. Note that “another thing” might also be this thing. For example, you might want to be able to link an employee to their manager, who is also an employee.

As fields are added they'll persisted back to WebGUI using ajax, and the fields screen will automatically update to reflect the new field.

The user can then go on to define the edit screen for this thing adding information like title, instructions what the save button is called, and who can add or edit things (employees) in this thing.

Edit Thing

Finished Editing

Fields

Edit Screen

View Screen

Search Screen

Edit Screen Title

Instructions

Use the following form to add or edit an employee in the database.

Who can add?

Who can edit?

Save Button Label

After Save

Edit Template

triggers

On Add

On Edit

On Delete

The “After Save” button will be defined somewhat by what other things there are in this thingy. The options available will be:

- Display search screen for this Thing.
- Display view screen for the thing that was just added/edited
- Display the add screen for some other Thing.
- Display the search screen for some other Thing.
- Display the “default” screen for this Thingy.

Note that there is a template box here. I've been debating whether to even allow templating of the thingy. I was thinking maybe it should be modifyable only by style sheets. However, upon much consideration, I think it should be templated. But the default template should be sufficient for almost anyone's needs. In addition, the default template should use some extra classes that designers can make use of to position elements using only CSS, without having to modify the actual template.

Finally, there are options to trigger workflows on various edit events. We should create at least one activity for this asset that will send emails about things. That would probably be the most common use

for triggers on the Thingy, plus it would give developer's an example of how to write activities for the Thingy.

The resulting edit screen that the user just created will look like this:

Edit An Employee

> [Delete](#)
> [View](#)
> [Search](#)

Use the following form to add or edit an employee in the database.

First Name	<input type="text" value="Bob"/>
Last Name	<input type="text" value="Jones"/>
Phone Number	<input type="text" value="608-555-1212"/>
Favorite Color	<input type="radio"/> red <input type="radio"/> green <input checked="" type="radio"/> blue <input type="radio"/> yellow <input type="radio"/> orange
Department	<input type="text" value="Manufacturing"/>
<input type="button" value="Submit"/>	

Note that the menu on the upper right is automatically generated depending upon what the current user's groups are compared with the privileges defined on the various tabs for this Thing.

The content manager can also specify a few things about the view screen.

Edit Thing

Finished Editing

- Fields
- Edit Screen
- View Screen
- Search Screen

Who can view?

View Template

fields

	Display	View Screen Title
First Name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Last Name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Phone Number	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Favorite Color	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Department	<input checked="" type="checkbox"/>	<input type="checkbox"/>

As the content manager is adding/editing fields on the fields tab you'll need to automatically update this tab to include those modifications. By default every new field should be checked in the "Display" column. Also by default, only the first item added should be checked in the "View Screen Title" column. The Display column determines what is show to the user when viewing a Thing, and the View Screen Title column determines what fields are used to make up the title on the view screen.

The resulting view screen will look like this:

Bob Jones

- > [Delete](#)
- > [Edit](#)
- > [Search](#)

First Name Bob

Last Name Jones

Phone Number 608-555-1212

Favorite Color Blue

Department [Manufacturing](#)

Note that "Manufacturing" is auto linked here. Clicking on that link will take you to the view screen of the Manufacturing Department Thing. This should only be linked if the current user has the privileges for the Department Thing view screen.

Finally, the content manager can define a few things about the search screen.

Edit Thing

Finished Editing

- Fields
- Edit Screen
- View Screen
- Search Screen**

List Screen Title

Description

Who can search?

Who can import?

Who can export?

Search Template

Things Per Page

fields

	Display	Search	Sort by
First Name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="radio"/>
Last Name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="radio"/>
Phone Number	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Favorite Color	<input type="checkbox"/>	<input type="checkbox"/>	<input type="radio"/>
Department	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="radio"/>

Most of the options here are familiar from the other tabs, however, the who can export/import options are unique. The Thingy allows import/export of each Thing's data via a comma separated values (CSV) file. More on that later.

The content manager can check off the fields that should be displayed for searching, as well as the columns that will be displayed in the result set. S/he can also select how many Things should be displayed per page on the result set before paginating, and what field to sort by.

The resulting search screen will look like this:

Search Employees

[> Export](#)
[> Import](#)

Welcome to our employee directory.

First Name	Last Name	Department	
<input type="text"/>	<input type="text"/>	Manufacturing	<input type="button" value="Search"/>

First Name	Last Name	Phone Number	Department
Bob	Jones	608-555-1212	Manufacturing
Kelly	Munsen	608-555-1111	Manufacturing
Juan	Ricardo	608-555-1010	Manufacturing
Dave	Richards	608-555-1932	Manufacturing
Casey	Zellman	608-555-8891	Manufacturing

[<<Previous](#)

[Next >>](#)

The user can then change the sort order by clicking on the column headings. The user can also view a Thing by clicking on an individual row in the result set, which will then take you to the view screen.

When the user clicks on the “Export” link they will be served a CSV file containing all the field names on the first row, and all the data from this Thing on the following rows. Note that links to other Things will need to be translated into the field that is linked (in this case department name of “Manufacturing” or “IT” or “HR”) rather than whatever id is stored in the Thing.

When the user clicks on the “Import” link they will be taken to this screen where the import mechanism allows the user to import a CSV file.

Import

[> Export](#)
[> Search](#)

Using this form you can import data directly into this Thing. Be sure that your fields are specified in the order shown below.

Choose a file to import

What about duplicates?

Ignore first line? yes no

fields

	File Contains	Check Duplicates
First Name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Last Name	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Phone Number	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Favorite Color	<input type="checkbox"/>	<input type="checkbox"/>
Department	<input checked="" type="checkbox"/>	<input type="checkbox"/>

The user may choose to either skip or overwrite duplicates. Duplicates will be determined by what is checked in the “Check Duplicates” column of the field list. Blank lines and lines without information for the “Check Duplicates” columns will automatically be ignored.

The user can specify whether to ignore the first line, in case they have column names in the first line.

And finally, they can tell the import mechanism what fields they will be importing. For example, if their file doesn't contain “Favorite Color” then they don't need to provide a blank space in their import file, they can exclude it by just not checking that field. And just like exports, links to other things need to be automatically translated so that “Manufacturing” becomes thingId XX3jklld44kskhe33HYi or whatever the id is for manufacturing.