Map Mashups #2 with Popfly: A Super Sized Fast Food Finder

A “store finder” is one of the most common types of mapping mashup applications. To help you find the nearest donut and coffee, or burger and fries, this lesson builds a “fast food finder” – mashing up live search data about restaurant locations with Virtual Earth mapping services.

The lesson approaches this task as a “real world” software development project on a very small scale, introducing concepts of user and functional requirements, design considerations, and web application architecture.

Prepared by Mark Frydenberg
Computer Information Systems Department
Bentley College, Waltham, MA
mfrydenberg@bentley.edu

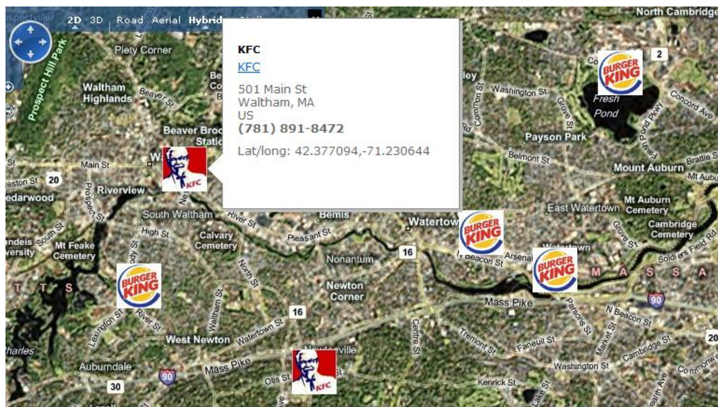
© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx .
Last Modified: 7/10/20087/9/20087/8/2008.

Map Mashups #2 with Popfly: A Super Sized Fast Food Finder

Professor Popfly Mashups Referenced in this Lesson:

- FastFoodFinderDataPOC
  - (http://www.popfly.com/users/professorpopfly/FastFoodFinderDataPOC-)
- FastFoodFinderIconPOC (http://www.popfly.com/users/professorpopfly/FastFoodFinderIconPOC-)
- FastFoodRestaurantIcons (data block) (http://www.popfly.com/users/professorpopfly/FastFoodRestaurantIcons) ~~data block~~
- FastFoodFinder (http://www.popfly.com/users/professorpopfly/FastFoodFinder-)
- FastFoodFinder2 (http://www.popfly.com/users/professorpopfly/FastFoodFinder2-)

Formatted: Indent: Left: 0.5", No bullets or numbering



Learning Outcomes

After completing this lesson, you should be able to:

- ~~Use the Phonebook search block to access Live data for a mashup~~
- Break a complicated task into smaller parts
- Identify different "tiers" of a multi-tiered (web) application
- Understand the role of creating a proof of concept in a software project prototype
- Distinguish between user requirements and functional requirements

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx -

Overview

A "store finder" is one of the most common types of mapping mashup applications. To help you find the nearest donut and coffee, or burger and fries, this lesson builds a "fast food finder" – mashing up live search data about restaurant locations with Virtual Earth mapping services.

The lesson approaches this task as a "real world" software development project on a very small scale:~~,~~ it ~~introducing~~ introduces concepts of user and functional requirements, design considerations, and web application architecture.

A software application's requirements specify what an it must do in order to "work." User requirements describe the needs and goals of users of a particular software application or web site. They serve as a "check list" to determine whether the user's needs have been met when the application is complete.

Functional requirements describe how a software application behaves – and the tasks it must perform, from the point of view of the application developer. These requirements may be specified using more technical language than that used in specifying the user requirements because they often describe the inner-workings of a system, and the tasks it must perform to handle input, processing, and output.

Note: Build and run the mashups in this lesson using Internet Explorer. The User Input blocks and the Virtual Earth block sometimes did not display correctly together using Firefox.

Formatted Table

~~Note: Build and run the mashups in this lesson using Internet Explorer. The User Input blocks and the Virtual Earth block sometimes did not display correctly together using Firefox.~~

User Requirements

To begin, let us specify the user's requirements for this mashup:

1. The user will enter a zip code.
2. The user will select the name of a fast food chain from a dropdown list.
3. The user will see the locations of the locations of up to ten of the selected fast food restaurants, each within a five mile radius of the specified zip code, plotted on a map. An appropriate icon will indicate the location of each restaurant.

Functional Requirements

At the most basic level, the mashup must perform three different tasks, translated from the user's requirements into these functional requirements.

1. Input: The mashup will obtain a zip code from the user.
2. Input: The mashup will obtain the names of two fast food restaurants from the user.
3. ~~The~~ Processing: The mashup will find information about up to 10 of the specified restaurants within a 5 mile radius of a specified zip code.
4. Processing: The mashup will find the icon that corresponds to the selected fast food restaurant.

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

3.5. Output: The mashup will plot the icon on the map along with information about each restaurant found.

Designing the FastFoodFinder Application

The user must enter a zip code and select the name of a restaurant. Rather than store the names of several restaurants as comma-separated values in a User Input block, we choose to store this data in a Popfly data block for two reasons:

1. By storing the restaurant names and icons as pairs in a data block, it will be easier to associate each icon with its corresponding restaurant when plotting the restaurants on a map.

2. Should the list of restaurants need to change later, only the data blockin the block needs to change; settings or connections to blocks that comprise the mashup itself does not need to be modified.

Formatted: Indent: Left: 0.5", No bullets or numbering

Note: If you created the data block and the mashup using it, you can simply edit the data block, save the changes to the data, then save and re-run the mashup.

1. If you are viewing or editing another Popfly user's mashup that contains a shared data block, you will not be able to edit the data in that person's data block directly. Instead, you will have to "rip it" (create your own copy), or create your own data block from scratch to use in your mashup. Be sure your data block has the same column headers as the one you are replacing

2. By storing the restaurant names and icons as pairs in a data block, it will be easier to associate each icon with its corresponding restaurant when plotting the restaurants on a map..

Formatted: Normal, No bullets or numbering, Border: Box: (Single solid line, Auto, 0.5 pt Line width)

Comment [ST1]: Doesn't the mashup need to be resaved for changes to the data source block to take effect? Please correct me if I'm wrong…

Formatted: Centered

In real-world mashups, user interfaces may change depending on the device displaying the mashup (a web application running on your laptop looks different then when displayed on your cell phone because the laptop screen is bigger.) However, the data is the same. Different skills From a software management point of view,t he process of are often required to query or obtain data, than to format it or display it.

Comment [ST2]: Skills? How about "code"? It's usually more of a "if we change the display, we don't want to have to touch code throughout the system – it's less error-prone if developers keep changes to a smaller area."

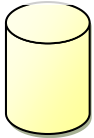
© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx .

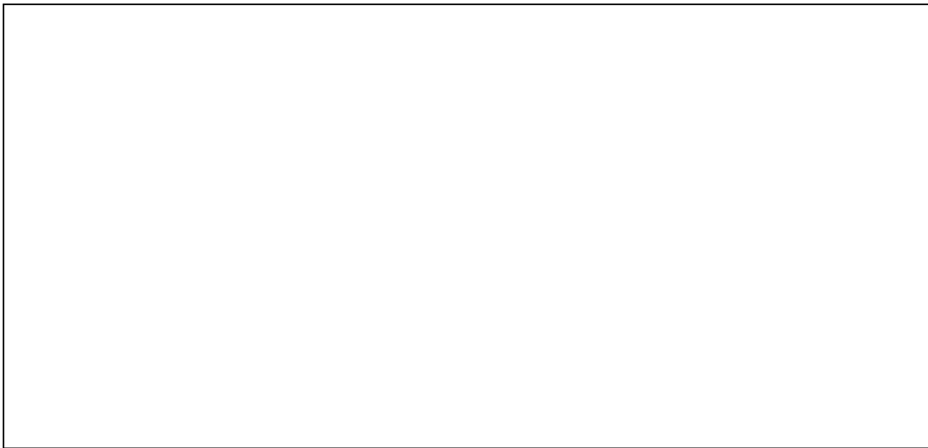
This separation of data from presentation steps and processing steps (Popfly blocks in this case, or application code in the real world) is common in web applications that use a multi-tiered architecture approach, as shown in the diagram below:

Client's Browser

Internet

Web Server



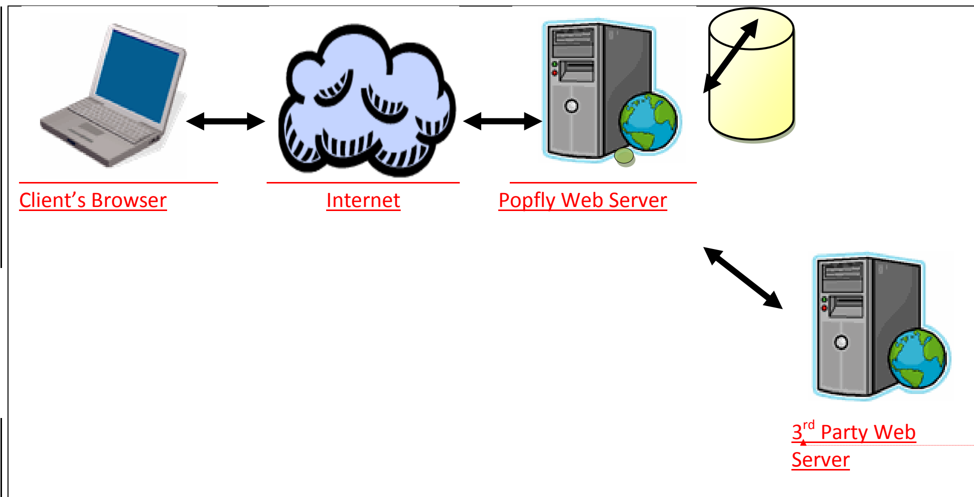


Popfly Database Server

Formatted: Right, Space After: 0 pt, Line spacing: single

Formatted: Space After: 0 pt, Line spacing: single

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx



Formatted: Centered, Space After: 0 pt, Line spacing: single

Formatted: Right, Space After: 0 pt, Line spacing: single

Formatted Table

Formatted: Right, Space After: 0 pt, Line spacing: single

Formatted: Right, Space After: 0 pt, Line spacing: single

Formatted: Superscript

The client's browser connects via the Internet to a web server. The client is responsible for sending user input (the zip code and restaurant name) over the Internet to the mashup engine running on the Popfly ~~Database Server~~ web server.

The mashup engine on the Popfly web server runs the mashup, handling all of its ~~handles all of the mashup's~~ "business logic", including making calls to APIs (application programming interfaces) to web services that are likely to be residing on external 3rd party web servers, and mashing the results. It also has access to application-specific data (in this case, the list of restaurants and icons) that is stored separately from the application in a database.

Formatted: Superscript

Proof of Concept #1: Finding Fast Food Restaurant Data

For each functional requirement, we will create a "proof of concept mashup" to convince ourselves that the mashup is possible to build. Each proof of concepts centers on being able to access the data needed at each step of the mashup, without concern about issues related to the user interface, input, or output.

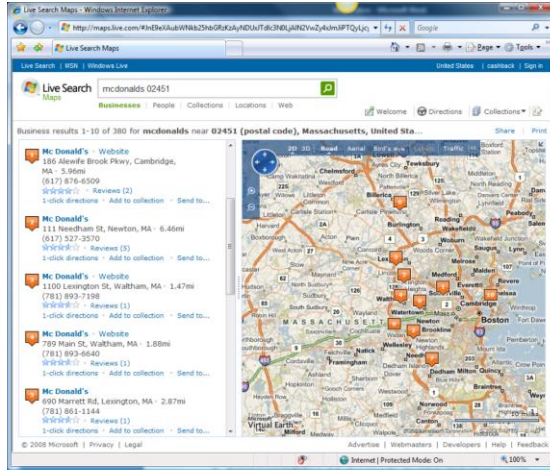
In functional requirement #1, the critical piece is determining if information about fast food restaurants is available, and if so, how ~~how~~ can Popfly access it?

One approach might be to create a static ~~database~~ table containing selected restaurant locations and store them in a Popfly data block. This list would be take a long time to gather by hand, and would need to be updated or verified frequently. ~~So t~~This approach does not lend itself to a more general solution for restaurants in any zip code.

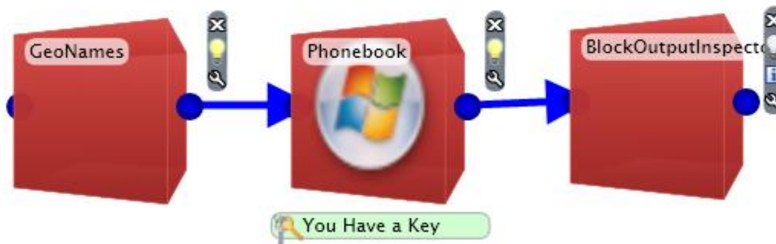
Many Internet search engines have "local" pages, where it is possible to search for local business information. For example, on the http://local.live.com search page, upon entering the name of a

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

restaurant and a zip code (i.e., “McDonalds 02451”), the Live search engine will return~~-~~ the names and locations of several McDonalds restaurants in that area along with their locations on a map.



The FastFoodFinderDataPOC (http://www.popfly.com/users/professorpopfly/FastFoodFinderDataPOC) mashup is a proof of concept mashup to complete this task. ~~The FastFoodFinderDataPOC “proof of concept” mashup shows~~It shows that it is possible to access the data that we need for the “real” fast food finder mashup we are trying to build:



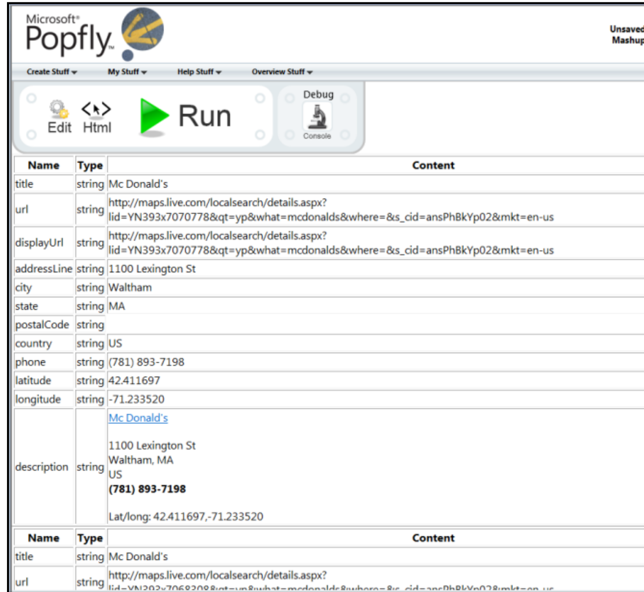
The inputs to the Phonebook block’s search operation are a search query, the number of items to return, the latitude and longitude on a map for the starting point of a the search, and a radius in miles within which items must be located in order to be included in the results.

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

The GeoNames block's lookupOnZipCode operation takes a zip code and finds the corresponding latitude and longitude. This will make locating the starting point for a search much easier than requiring a user to specify the latitude and longitude of a starting location.

The BlockOutput inspector shows the output of the Phonebook search. Because the Phonebook block outputs a latitude and a longitude for each item (among other things), and prior experience shows that Popfly can plot locations on a map given a latitude and longitude, this exercise affirms that we should be able to create the desired Fast Food Finder mash up.

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx



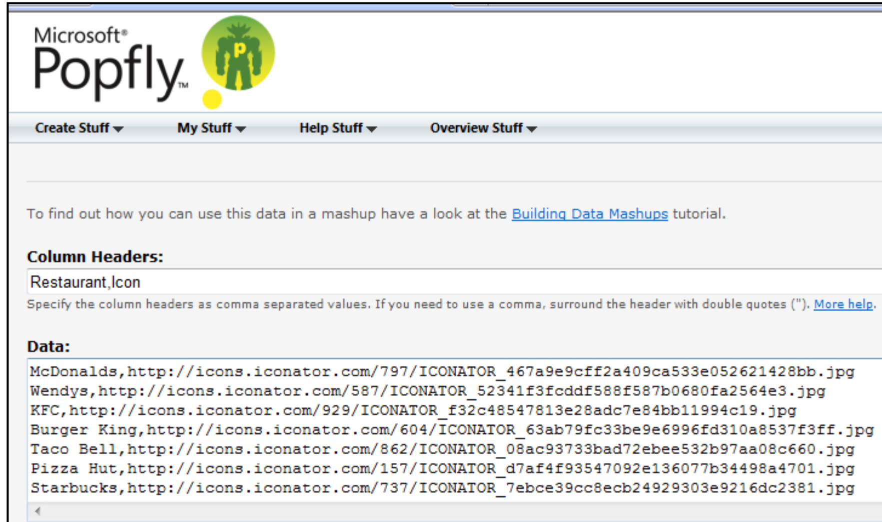
Name	Type	Content
title	string	Mc Donald's
url	string	http://maps.live.com/localsearch/details.aspx?lid=YN393x7070778&qt=yp&what=mcdonalds&where=&s_cid=ansPhBkYp02&mkt=en-us
displayUrl	string	http://maps.live.com/localsearch/details.aspx?lid=YN393x7070778&qt=yp&what=mcdonalds&where=&s_cid=ansPhBkYp02&mkt=en-us
addressLine	string	1100 Lexington St
city	string	Waltham
state	string	MA
postalCode	string	
country	string	US
phone	string	(781) 893-7198
latitude	string	42.411697
longitude	string	-71.233520
description	string	Mc Donald's 1100 Lexington St Waltham, MA US **(781) 893-7198** Lat/long: 42.411697,-71.233520

Name	Type	Content
title	string	Mc Donald's
url	string	http://maps.live.com/localsearch/details.aspx?

Knowing this, we are ready to begin to design the FastFoodFinder mashup to meet its input, processing, and output (functional) requirements.

To mimic this architecture, a Popfly Data block called FastFoodRestaurantIcons will store the names of nine restaurants and URLs of icons reflecting their logos. This block uses icons found at http://www.iconator.com .

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

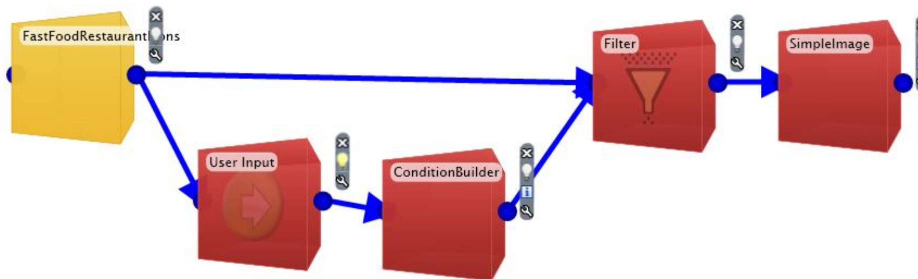


~~The next step is to specify the user input, processing, and output requirements.~~

Proof of Concept #2: Matching Icons with Restaurants

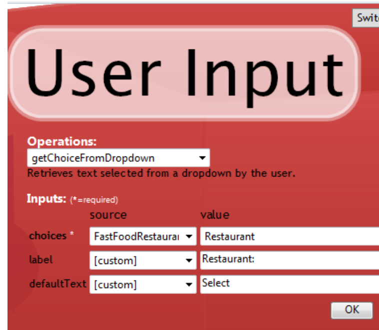
The second task (finding the icon that corresponds to the restaurant name) is accomplished by using a filter condition on the Restaurant list from the FastFoodRestaurantIcons data block to obtain the entire record (restaurant name and icon pair) that matches the restaurant chosen by the user.

The FastFoodFinderIconPOC (http://www.popfly.com/users/professorpopfly/FastFoodFinderIconPOC ~~mashup~~) mashup is a proof of concept mashup ~~of~~that completes this task:

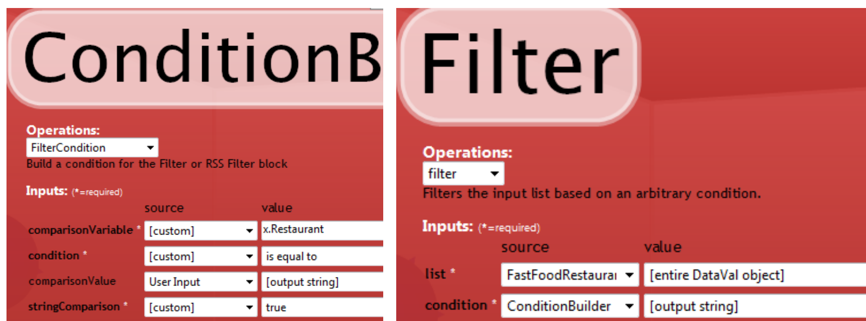


The FastFoodRestaurantIcons block sends the list of restaurant names to the User Input block, where Restaurant list becomes the data source for the dropdown list.

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx



The ConditionBuilder block sets up a FilterCondition for the Filter block, to return a list of records from the data block whose names are equal to the selected restaurant. In this case there will be only one matching record.



Finally, the SimpleImage block displays the image URL from the Filter block as an image in the browser when the mashup runs. Here is the output after selecting Wendys, Burger King, Starbucks, Dunkin' Donuts, and KFC:

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

Putting the Pieces Together: User Input

A user input block with a text box will enable the user to enter a zip code.

The list of restaurants from the data block serves as the data source for another User Input block. This block will display dropdown list containing the names of the restaurants for the user to make a selection.

(It is also possible to use the User Input block's getTextAndChoice operation, but since a later example will enhance this mashup to display two different fast food chains on the same map, it is better to keep the inputs as coming from separate User Input blocks.)

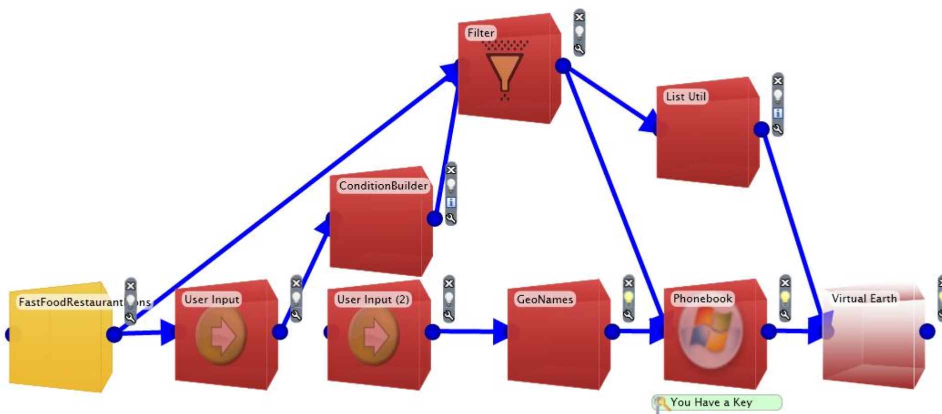
Putting the Pieces Together: Processing

Proof of Concept #1 showed how to access the restaurant information for a particular zip code. The final mashup connects the two User Input blocks, and replaces the hard-coded zip code and restaurant name values with those from the corresponding user input blocks.

Proof of Concept #2 showed how to find the icon associated with a selected restaurant.

Putting the Pieces Together: Output

After merging these two proof-of-concept mashups, all that's left is to do is work on creating the map display. To plot the restaurants on a map, a Virtual Earth block will need the icon URL and data from the Phonebook block (which provides the latitude, longitude, and restaurant information.) The completed mashup is shown below:



Connecting a List Util block (see below) to obtain the restaurant's icon URL and a Virtual Earth block to plot the items on the map complete the mashup.

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

Here is the output of the FastFoodFinder1 mashup:

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

Note that you can keep on selecting restaurants from the dropdown list several times, and Popfly will run the mashup for the new restaurant, and add the results to the existing map.

What's the List Util Block Doing There? Technical Details

There is only one subtlety in the completed FastFoodFinder mashup: it needs to use the List Util block. In the FastFoodFinderIconPOC proof of concept mashup, we said that the output of the Filter block contains the URL of the icon corresponding to the selected restaurant. However, as taught in the Logic lesson, the Filter block really outputs a *list* of items that match the specified criteria (in this case, a matching restaurant name.) In this example, the output of Filter is a list containing exactly one item.

To make the mashup work correctly requires extracting the first (and only) item from the icon list and passing it to the Virtual Earth block as an item, not a list containing it. That's where the List Util block comes in. The List Util block has a get operation to extract an item from a specific position from a list, and return it as an item.

Here is a closer look at how this works:

The figure below shows the beginning of the list of restaurants and icons contained in the FastFoodRestaurantIcons data block. The number at the left of each pair indicates its position within the list. The Filter block returns a list of all the restaurants that match the filter condition, in this case, [x.Restaurant] [is equal to] [Burger King].

0	McDonalds	http://icons.iconator.com/797/ICONATOR_467a9e9cff2a409ca533e052621428bb.jpg
1	Wendys	http://icons.iconator.com/587/ICONATOR_52341f3fcddf588f587b0680fa2564e3.jpg
2	KFC	http://icons.iconator.com/929/ICONATOR_f32c48547813e28adc7e84bb11994c19.jpg
3	Burger King	http://icons.iconator.com/604/ICONATOR_63ab79fc33be9e6996fd310a8537f3ff.jpg
4	Taco Bell	http://icons.iconator.com/862/ICONATOR_08ac93733bad72ebee532b97aa08c660.jpg

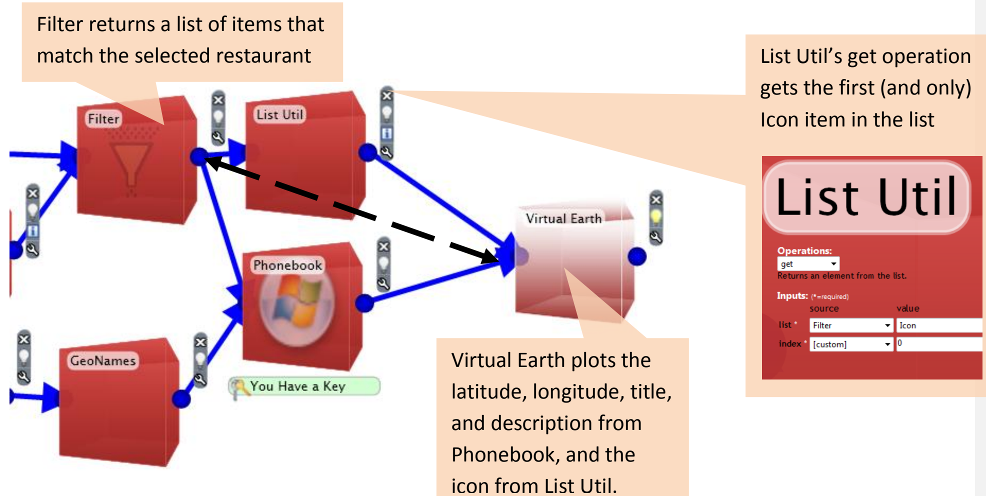
…

The highlighted restaurant is the only one that matches, so the Filter block returns it as a list of one item:

0	Burger King	http://icons.iconator.com/604/ICONATOR_63ab79fc33be9e6996fd310a8537f3ff.jpg

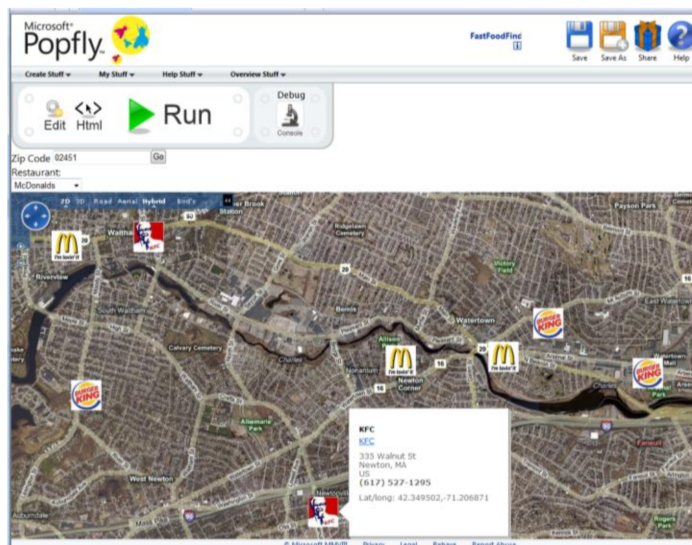
The get operation of the List Util block obtains the icon in position 0 of that list, returning only the URL of the icon corresponding to the name specified in the Filter Condition.

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx



Had the Filter block been connected directly to Virtual Earth (without using the List Util block), only one restaurant would appear on the map. While the Virtual Earth block operates on both lists connected to it, if the lists are of different lengths, processing ends after all of the items in the shorter list – in this case, the list containing exactly one restaurant icon – have been exhausted.

By using the List Util block to obtain the icon URL, and connecting it and the Phonebook block to the Virtual Earth block, there is now only one list (of restaurants from Phonebook) over which the Virtual Earth block must iterate in order to plot all of the pushpins on the map.



© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

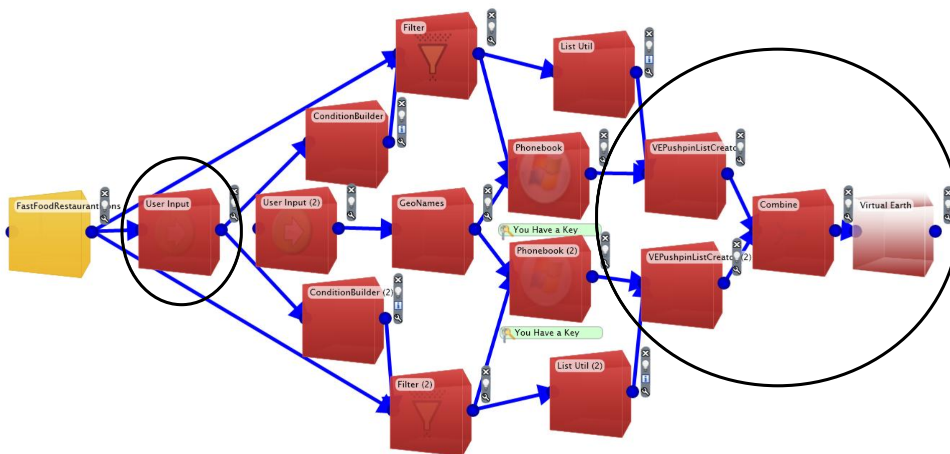
Super Size the Mashup: Display Information about Two Fast Food Restaurants on the Same Map

~~Display Information about Two Fast Food Restaurants on the Same Map~~

This enhancement modifies the FastFoodFinder mashup so that it will display data about two different fast food restaurants on the same map at the same time. The strategy this time will be to duplicate the earlier "combined" mashup, and then combine the data from both "halves" to plot on a map.

A minor change to the user interface is shown in the circle to the left in the mashup below: Use the getTwoChoicesFromDropdowns operation (instead of getChoiceFromDropdown to get a single value) from the User Input block circled below, in order for the user to be able to enter the names of two restaurants. Connect the output to the two ConditionBuilder blocks.

See FastFoodFinder2 for the completed mashup, which looks like this:



To combine the data from both Phonebook and List Util blocks, as shown by the four circled blocks above, requires creating a new object (or container) to store each new item to be plotted on the map using the same format.

To combine data from different sources on the same map, use a Combine block to combine Pushpin lists output from different VEPushpinListCreator blocks into a single list, and send the combined list into Virtual Earth.

Block for Creating Pushpin Lists

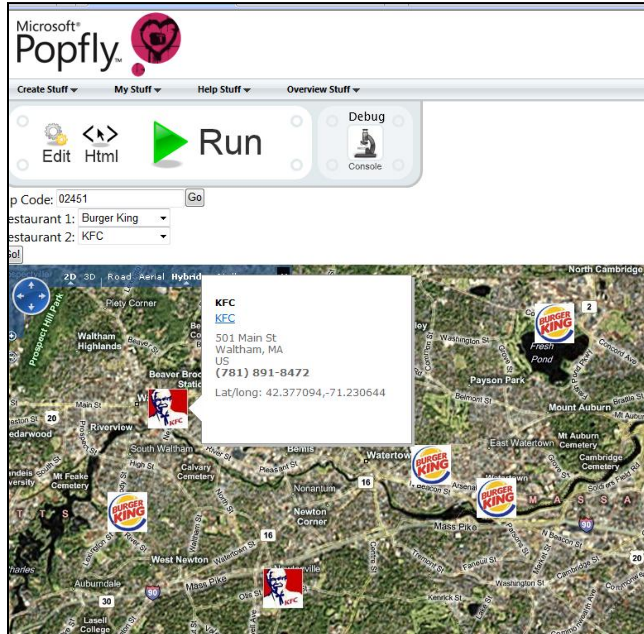
Block	Description
VEPushpinListCreator	Creates lists of pushpin information for plotting on Virtual Earth.

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

The VEPushpinListCreator block creates a new list of pushpin objects to plot on Virtual Earth. Each pushpin will contain latitude, longitude, title, and description from the Phonebook block, and the URL for the pushpin icon obtained from the List Util block.   The completed mashup uses one VEPushpinListCreator block to create a list of Pushpin objects for each of the restaurant's items, and then combines both lists into one to provide to Virtual Earth for plotting on the map.

Run the mashup once. Here's sample output:

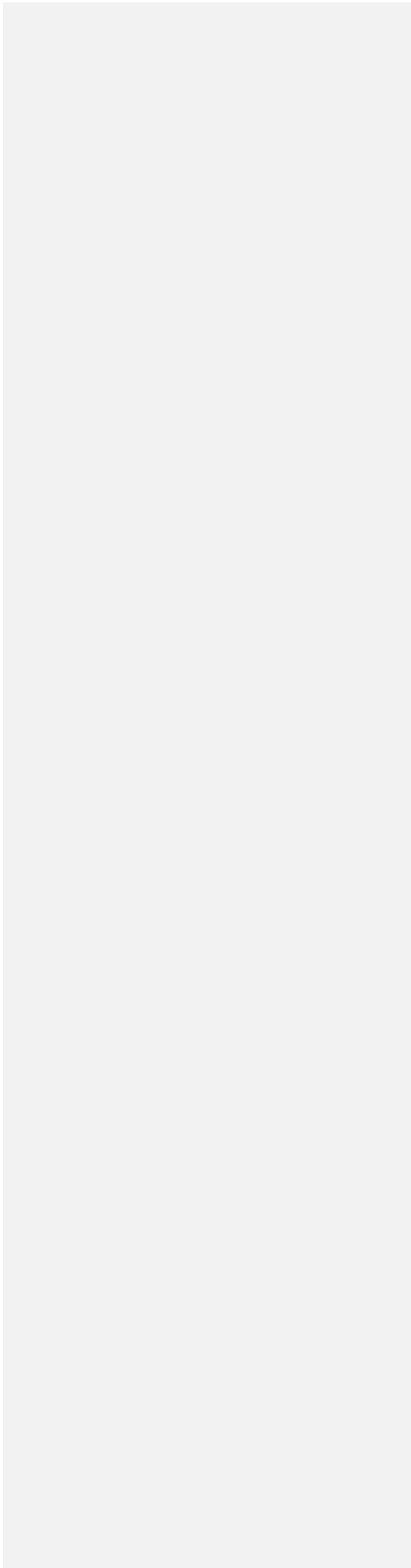
© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx



This completes the Fast Food Finder mashup.

Formatted: Left

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx



© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

~~Popfly PopQuiz~~

Popfly PopQuiz

Formatted: Heading 1

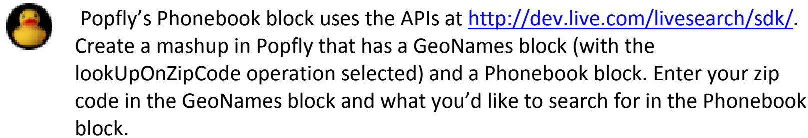
1. The Phonebook block requires latitude and longitude values for a location from where to begin the search. What is an easy way to use a zip code with the Phonebook block instead of a latitude and longitude?
2. How does using a data block for the list of restaurants and their corresponding icons help to make both the data and the mashup itself easier to modify?
3. Why are functional and user requirements necessary when building real-world software applications? How do they differ?
4. Why was it useful in the development of this mashup to break it down into smaller, discrete problems?
5. What does the Combine block combine in the FastFoodFinder 2 mashup?
6. What are the different tiers or “levels” in a multi-tiered web application? What role does each play in running the application?

Comment [ST3]: I don’t think there’s enough in the doc on this topic for students to answer this well.

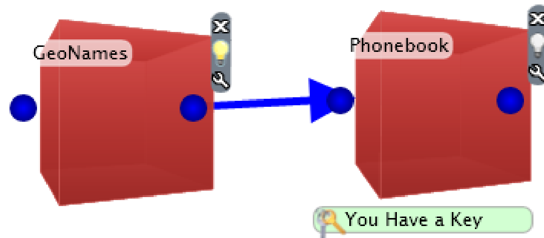
© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

Try It in Popfly

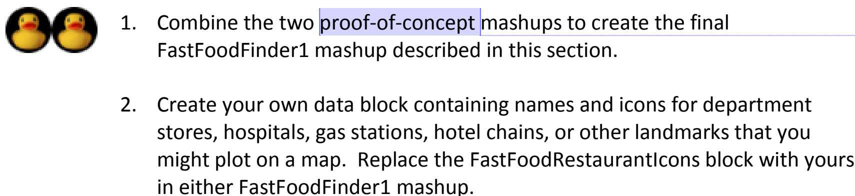
Modify or build these Popfly mashups to demonstrate your understanding of this lesson. The more ducks, the bigger the challenge.



Popfly's Phonebook block uses the APIs at http://dev.live.com/livesearch/sdk/. Create a mashup in Popfly that has a GeoNames block (with the lookUpOnZipCode operation selected) and a Phonebook block. Enter your zip code in the GeoNames block and what you'd like to search for in the Phonebook block.



Verify that the data returned is exactly the same as that obtained from the Live Search Interactive SDK page for the Basic phonebook search. View the data in both HTML and XML on the Live Search Interactive SDK page.



1. Combine the two proof-of-concept mashups to create the final FastFoodFinder1 mashup described in this section.

2. Create your own data block containing names and icons for department stores, hospitals, gas stations, hotel chains, or other landmarks that you might plot on a map. Replace the FastFoodRestaurantIcons block with yours in either FastFoodFinder1 mashup.

Comment [ST4]: Remove hyphens, you don't use these elsewhere for "proof of concept"

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

1. Modify the FastFoodFinder1 mashup to connect the Filter block directly to the Virtual Earth block and convince yourself that the mashup displays only one restaurant on the map.

2. Now put back the List Util block and set it up to use the get operation as described, and verify that the mashup works correctly again. Explain what's going on.

1. Create your own mashup to plot two types of data from two different sources on the same map. For example, enhance the Real Estate mashup described in the earlier Mapping lesson to create a map mashup that displays homes for sale and also Indian restaurants in the same neighborhood.

Comment [ST5]: ?

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx

Learn More about Map Mashups

These are some interesting mapping mashups. How might you create similar versions in Popfly?

- FastFoodMaps.com(http://www.fastfoodmaps.com/ -)
  This mashup displays information about ten different fast food restaurants on the same map for anywhere in the USA. Thanks for the idea!
- HousingMaps.com (http://www.housingmaps.com) displays apartment listings from Craigslist on a Google map.
- Gas Buddy.com (http://gasbuddy.com/)
  Find locations of this week's low and high gas prices on a map.
- Windows Live Map Mashups Showcase (http://dev.live.com/mashups/-)

Popfly Documentation for Map-related Blocks

- Phonebook block (http://www.popflywiki.com/users%3Bpopfly%3Bphonebook.ashx)
- VEPushpinListCreator block (http://www.popflywiki.com/users%3Bpopfly%3BVEPushpinListCreator.ashx)
- Virtual Earth block (http://www.popflywiki.com/users;popfly;Virtual%20Earth.ashx)

© Mark Frydenberg, 2008. Some rights reserved. See http://popflywiki.com/CurriculumEULA.ashx .