

Assemble the Social Web with

zembly

Using zembly.com, learn how to build, host, and deploy Facebook apps, iPhone apps, flickr widgets, Google mashups, and other widgets and social applications in minutes—all using just your browser, this book, and your creativity!

Let's Make a Social Application. Right Here. Right Now. Together.

zembly

By award-winning authors

GAIL ANDERSON and **PAUL ANDERSON**

with zembly architects *Todd Fast* and *Chris Webster*

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

Sun Microsystems, Inc. has intellectual property rights relating to implementations of the technology described in this publication. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents, foreign patents, or pending applications.

Sun, Sun Microsystems, the Sun logo, J2ME, J2EE, Java Card, and all Sun and Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact: U.S. Corporate and Government Sales, (800) 382-3419, corpsales@pearsoned.com.

For sales outside the United States please contact: International Sales, international@pearsoned.com.

Library of Congress Control Number: 2008941460

Copyright © 2009 Sun Microsystems, Inc.

4150 Network Circle, Santa Clara, California 95054 U.S.A.

All rights reserved.

Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to: Pearson Education, Inc., Rights and Contracts Department, 501 Boylston Street, Suite 900, Boston, MA 02116, Fax: (617) 671-3447.

ISBN-13: 978-0-13-714431-0

ISBN-10: 0-13-714431-8

Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.
First printing December 2008

Preface

There once was an engineer named Todd who had a vision of creating the programmable web. He wrote a white paper describing his ideas and the social climate for making his vision a reality. As the participation in social networking continued to grow (and, as we have noted in Chapter 5, continues to grow each month by big numbers), the motivation for implementing such a widget-building, application-building environment becomes easier and easier to justify.

This book comes out on the leading edge of **zembly**'s existence. The environment we document and describe today will no doubt change, but for you pioneers of social network programming, it will only get richer, easier, and more rewarding (maybe even financially rewarding).

The biggest challenge we face in writing a book like this is keeping current with **zembly**. As **zembly** evolves it will improve incrementally and continuously. One of the great advantages in offering a web-based tool is that new "versions" happen often and are not tied to lengthy production cycles that traditional development tools use. To keep current, we point readers to **zembly** itself (zembly.com), its blog (blog.zembly.com), and wiki (wiki.zembly.com). These resource points will go a long way in keeping you up to date with new **zembly** features that are rolling out, even as we finish up this manuscript.

About the Audience

This book is aimed at **zembly** users of all technical levels. We hope not only to help you use **zembly** effectively, but to provide examples that will get you up to speed quickly. We anticipate that **zembly** users will represent a whole range of *technologists*; that is, users who are comfortable on the web, dabble a bit in HTML, CSS, or JavaScript, and see the internet as a tool to be exploited. You might be a professional social network game developer or a home-grown blogger ready to expand your widgetry. You might even be a community organizer ready to reach out to untapped audiences for your cause célèbre.

JavaScript Programming

As we write about the programmable web, the next logical question might be “What language do I use to program this programmable web?” The short answer is JavaScript. If you’re an experienced JavaScript, HTML, and CSS coder, you will be very comfortable constructing services, widgets, and applications on **zembly**. But what if you’ve never used JavaScript before? Maybe you know Java, or C/C++ or even C#. Or, perhaps you have a background in scripting languages, including Perl, Python, Ruby, or PHP. Fear not; at least one of the author’s first exposures to JavaScript programming occurred while working on this project.

To help get you up to speed with JavaScript, consult the web for tutorials at www.w3schools.com. Here’s a few tips to get you started.

- JavaScript tutorials are at www.w3schools.com/js.
- HTML tutorials are at www.w3schools.com/html.
- CSS tutorials are at www.w3schools.com/css.
- David Flanagan’s *JavaScript, The Definitive Guide* is an excellent reference to have at your desk.
- Begin by cloning and building upon already-published widgets, services, and applications. Not only can you learn from previously written JavaScript, CSS, and HTML code found in these examples, but you can use these as starting points to build your own artifacts.
- Consider using the Prototype JavaScript library. This library is available for your **zembly** widgets and provides some nice JavaScript programming help. Prototype tutorials and references are at www.prototypejs.org/learn.
- **zembly** provides widget templates that let you easily build and configure widgets. You just might be able to build a widget with no programming at all using templates!
- Use the **zembly** forum to ask questions. The forum (forum.zembly.com), is not only a place to ask questions of other **zembly** users, but it also provides a place to report bugs or anomalous behavior.

About the Examples

Use **zembly**’s Search mechanism to find all of the examples presented in this book. Because the examples are live, deployed services and widgets, you will always find the most current, published version on **zembly**. Provide the search term **zembly-book** and click **Search**, as shown here in Figure 1.

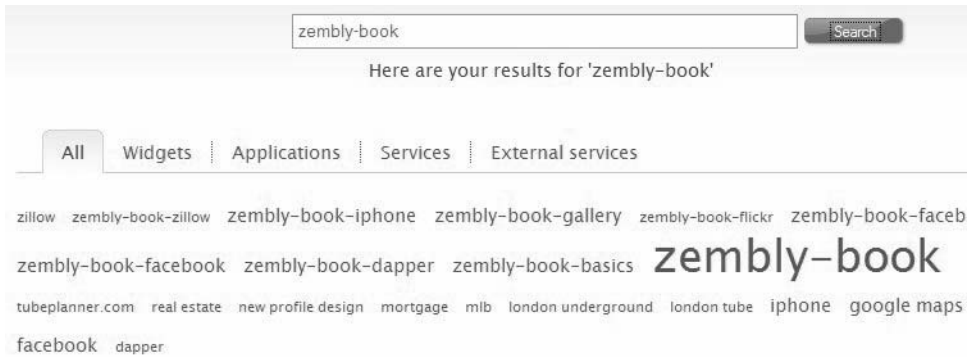


Figure 1. Finding all of the examples presented in this book

Notational Conventions

We've applied a rather light hand with font conventions in an attempt to keep the page uncluttered. Here are the conventions we follow.

Element	Font and Example
zembly UI controls	Publish, Create something! Add a new parameter
URLs	<code>zembly.com</code>
inline code elements	<code>result.user.nsid</code>
code blocks and listings	<pre>if (result.user) { . . . }</pre>
widget names	<code>LoanPaymentWidget</code>
service names	<code>LoanPaymentService</code>
application names	<code>CapitalPunishment</code>
key combinations	<code>Ctrl+Alt+F</code> , <code>Ctrl+Space</code>
user input	specify minimum 1 and maximum 20

2 zembly Basics



Welcome to zembly.

zembly lets you build services, widgets, and web applications and publicly deploy them. The philosophy behind **zembly** is to encourage you to build upon previously published services and widgets, to discover what other **zembly** users are building, and to socialize the building process by collaborating with your **zembly** contacts. As **zembly** matures, it will allow you to build widgets, services, mashups, and social applications targeting the many social networks present on the web. This chapter is aimed at those who are just getting started with **zembly**, giving you a glimpse into the future of building the web.

zembly is a social network. It encourages you to build your own contacts and collaborations. Those of you who work on group projects will appreciate the easy collaboration in code development, and by extension, idea sharing. The ultimate goal for **zembly** is to make the threshold very low for building and deploying a widget or mashup that others can drop into a web page (such as a blog or Facebook profile page). Combining social networking, collaborative development, and sharing a collection of published services and widgets, **zembly** facilitates each step that results in a

published, deployed, and fully accessible and easily importable widget, service, or application.

This chapter will help you get started. It assumes that you are a registered zembly user.

What You Will Learn

- How the zembly site is organized
- The types of things you can build with zembly
- What you'll find on the Samples page
- How to embed a widget in your web page
- How to view or edit your Profile page
- All about the zembly Keychain
- How to find zembly service providers
- How to create, test, and publish a service
- How to create, test, and publish a widget
- How to include a library with your JavaScript code
- How to manipulate drafts, versions, and the timeline
- How to create a service and widget that calls an external service

Examples in This Chapter

All examples in this chapter are tagged **zembly-book-basics**. To search for them, select **Search** at the top of any zembly page. Supply the search term **zembly-book-basics** and click the search icon as shown in Figure 2.1.

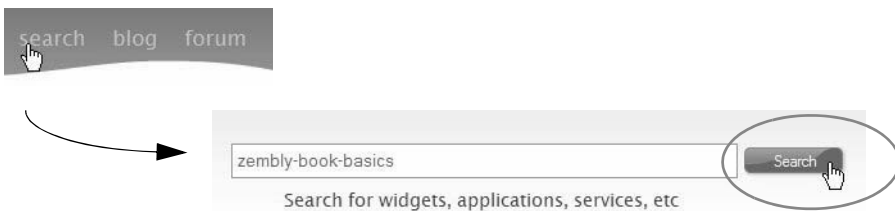


Figure 2.1 Searching for the examples in this chapter

The search results lists the widgets and services discussed in this chapter.

2.1 Exploring the Samples

Let's start with the **zembly** Samples section, which lists applications (Facebook and OpenSocial), widgets (blue badge), and services (orange badge). To see **zembly's** samples, select **samples** from the top dashboard, as shown in Figure 2.2.

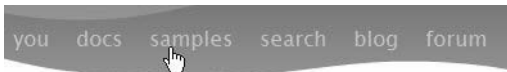


Figure 2.2 The zembly samples help you get started

The sample Facebook applications include **CapitalPunishment**, which is presented in Chapter 6 (see “Capital Punishment—A Challenging Facebook Application” on page 162). The sample services run on the **zembly** server and generally call other services out on the web to do things. Right now¹ the sample services include

- **AmazonProductSearch**—lets you search Amazon’s product catalogs and retrieve detailed product information, including prices, images, etc.
- **FlickrPhotoSearchService**—calls the flickr picture search service.
- **GoogleGeocodeSampleService**—enables you to search a ZIP code for a given address.
- **HelloWorld**—takes your name and says hello.
- **WeatherTodayService**—retrieves weather for a specific U.S. zip code.
- **YahooTripSearchService**—enables your applications to use a Yahoo! API to search for public trip plans that were created with Yahoo!.
- **YouTubeSampleService**—lists information about videos that have a specified tag.
- **ZillowSampleService**—finds a property given an address. The returned information includes the address for the property or properties, the Zillow Property ID (ZPID), the current Zestimate, the date the Zestimate was computed, a valuation range, and the Zestimate ranking for the property within its ZIP code.

1. **zembly** will add more samples to this page, so check back often.

- **zventsSearchService**—search for events that are happening around a given U.S. location.

The Sample section also includes a list of widgets. Widgets provide a user-friendly object that you can embed in a web page. All of the above services have corresponding widgets. To view a widget's page, click its name in the Samples section. For example, if you select **HelloWorldWidget**, zembly takes you to its page so you can see how it was built. To view its source, select the **View** tab as shown in Figure 2.3.



Figure 2.3 Exploring the HelloWorldWidget development page

Further down the page zembly shows you how to embed the widget in a page by providing the code you can select, copy, and paste. Figure 2.4 shows the code window to embed widget HelloWorldWidget (Share This Widget).

Share This Widget

Use this code to embed this widget in a Web page, like your blog, wiki, or other website. Make sure to fill in the real values where you see [value]!

```
<iframe
  src="http://dde7e989aa2a4122aef8a6e53f29e9fb.zembly.com/things/dde7e989aa2
```

Figure 2.4 Embedding (sharing) a widget

For example, you can create a web page and call the HelloWorldWidget using the following code:

```
<iframe
  src="http://dde7e989aa2a4122aef8a6e53f29e9fb.zembly.com/things/
  dde7e989aa2a4122aef8a6e53f29e9fb;iframe" frameborder="0">
</iframe>
```

You can then open this file in your web browser, which calls the HelloWorldWidget. This widget displays a box and provides an input field to supply a name. A call is

made to the HelloWorld service to display the name provided by the user, as shown in Figure 2.5.



Figure 2.5 Embedding and running the HelloWorldWidget in a web page

Using Clone

You can clone any application, widget, or service on **zembly**. This means that you create a copy for yourself. Once you clone a thing on **zembly**, you own it and you can then modify it. **zembly** encourages you to clone artifacts that you like; it is both a great learning tool and more importantly, you can build something innovative based on the work someone else has already done. This makes **zembly** users more productive. To clone a widget, select **Clone this widget** on its development page, as shown in Figure 2.6.

Furthermore, when you clone something on **zembly**, the score of the original widget (or service or application) increases to reflect the cloning. Scores also change when people rate **zembly** “Things” or favorite them.

You are not allowed to edit this widget, but you can get your own copy to play with by cloning it.

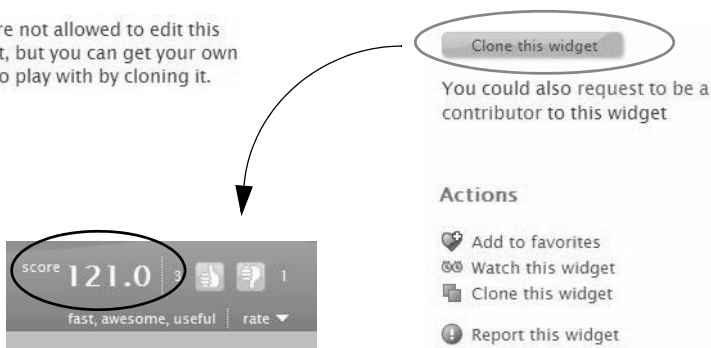


Figure 2.6 Cloning a widget increases its score

Widget Actions

Widgets (this applies to services and applications as well) list actions on their page. Besides cloning, you can add a widget to your list of favorites, report a widget, or watch a widget. When you mark something as a favorite, you have an easily accessible “bookmark” as shown in Figure 2.7.

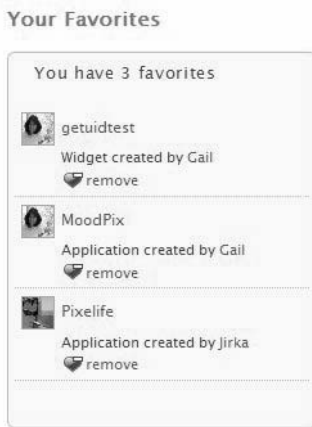


Figure 2.7 Your Favorites give you a convenient bookmark

When you watch something, zembly lets you know when its owner publishes a new version.

Tags on zembly

Use zembly tags to label your widgets, services, and applications to help others find Things through the zembly search mechanism, as shown in Figure 2.8.



Figure 2.8 Tags let you find widgets, services, and applications through searching

AmazonProductSearchWidget—Widget Preview

Let's explore the AmazonProductSearchWidget. From the Samples page, select AmazonProductSearchWidget. zemby takes you to this widget's page. You'll see a box area with the instructions **Click here to preview this widget**. When you click the box, the widget runs in a preview window. You can increase the size of the preview window by selecting the corner (or edges) and dragging until the preview window is the size you want, as shown in Figure 2.9.

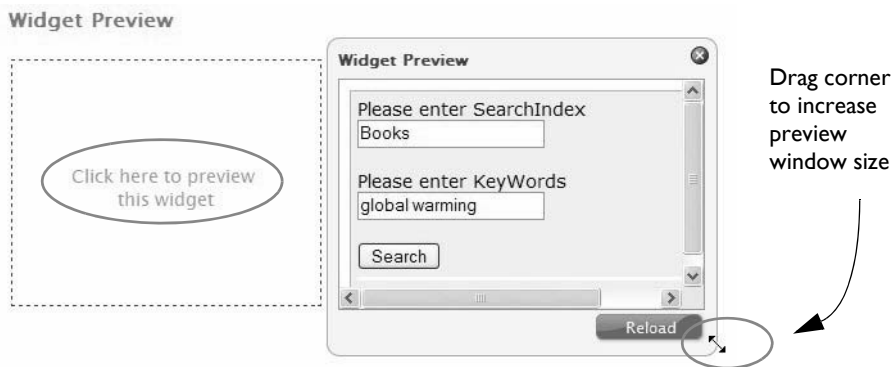


Figure 2.9 Previewing a widget and adjusting the preview window size

Provide product search index and keywords and click Search. Figure 2.10 shows the result after searching for keyword "zemby" in search index "Books."



Figure 2.10 Previewing the AmazonProductSearchWidget

Widget code includes (X)HTML (for rendering), CSS (for styling), and JavaScript for program logic and calls to external services. When you make a service call, results typically come back in XML or JSON format. Exactly what the data represents depends on the service and the format it uses. For example, with XML you may see results that are RSS 2.0, or ATOM. As it turns out, the Amazon service that AmazonProductSearchWidget calls returns data in XML format. The external web service will specify how to interpret the data that is returned.

You are encouraged to look at the XHTML, CSS, and JavaScript code for this widget (click View as shown in Figure 2.3 on page 14). This chapter will delve into building widgets soon, but first let's show you how to use this widget in a web page.

Embedding AmazonProductSearchWidget

The AmazonProductSearchWidget has sharing enabled. This means you can export the widget to many popular web sites and pages by simply selecting the logo that corresponds to the target site. zembly has partnered with ClearSpring to provide sharing and tracking of your widgets (see www.clearspring.com). We show you how to enable sharing in Chapter 3 (see "Sharing Your Widget" on page 66). However, let's first show you how to embed a widget in a web page.

Since sharing is enabled for AmazonProductSearchWidget, select **Embed** from the list of options as shown in Figure 2.11. The share window now displays option Other Sites. Select **Other Sites** and you'll see the window with the JavaScript code you need to invoke the widget from an arbitrary HTML page.

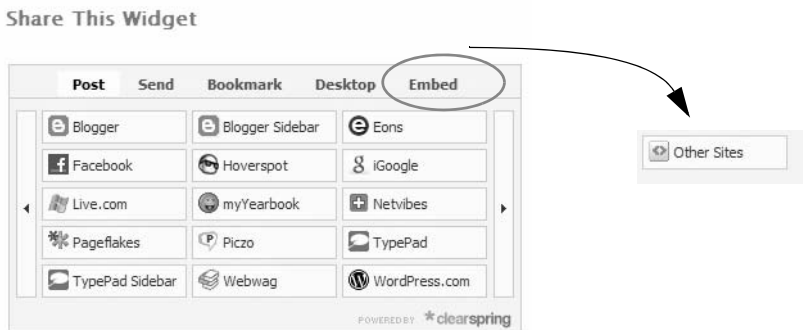


Figure 2.11 Embedding AmazonProductSearchWidget in a web page

Cut and paste this code into the HTML editor of your choice and open it in your browser. You can add other rendering code as shown in Listing 2.1. Here is the source for the HTML file used to run this widget in a browser.

Listing 2.1 AmazonProductSearchWidget HTML file

```
<h2 style='margin-left:10px; margin-bottom: 0px'>Let's search Amazon!</h2>
<script type="text/javascript"
src="http://widgets.clearspring.com/o/49249714e57f0b59/4924d27e70974fe2/
4924971425b85ee0/cafd08e6/widget.js">
</script>
```

After creating the HTML file, open it in your browser. Enter a product search index and one or more keywords, then click **Search**. Figure 2.12 shows the browser output.

Let's search Amazon!



Please enter SearchIndex
Books

Please enter KeyWords
zembly

Search

Assemble the Social Web with
zembly

Figure 2.12 AmazonProductSearchWidget running in a browser

Sharing Your Widgets with Clearspring

Besides embedding widgets in pages, you can also share widgets by adding them to any number of popular sites, such as your iGoogle Home page (see www.google.com/ig). You don't have to be the widget's owner. Click **Post** from the list of options and then select **iGoogle** from the option icons in the grid (see Figure 2.11). Now click **Open** in the Add to your iGoogle page display, as shown in Figure 2.13



Figure 2.13 Adding AmazonProductSearchWidget to your iGoogle home page

After clicking **Open**, you'll be redirected to Google and asked to confirm. Click the big blue **Add to Google** button, as shown in Figure 2.14. You will now see your iGoogle home page updated with the widget inside.

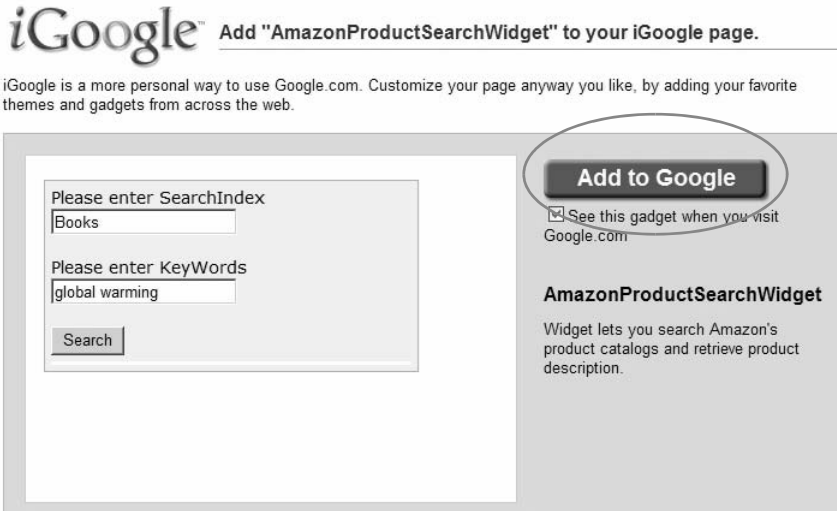


Figure 2.14 Adding AmazonProductSearchWidget to your iGoogle home page

2.2 About You—Your Home Page

zembly is about people like you who participate in building and publishing widgets, services, and other objects. The **You** tab takes you to your home page. This is the starting point for the work you do on zembly. Figure 2.15 shows your home page with the top-level tabs and the right-side navigation area. From the right-side navigation area you can

- edit or view your profile,
- manage your Keychain (a list of API keys for web services),
- view your favorite zembly things (widgets, services, or applications).

From the **Things** tab, you can

- see your work in progress,
- see all the things you own,
- select one of your things to edit.

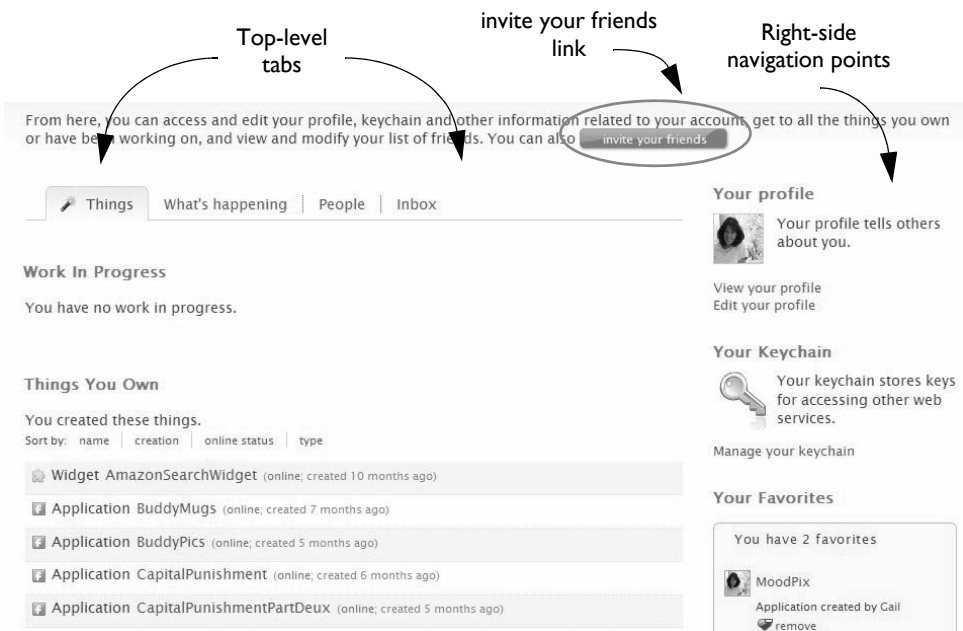


Figure 2.15 Your home page is your starting point

From the **What's happening** tab, you can see what others are doing (reported in the news feed).

From the **People** tab, you can

- view your contacts (other zembly users that you have added to your profile),
- search zembly for additional contacts.

From the **Inbox** tab, you can

- see messages others have sent you,
- see requests to collaborate that others have sent you.

And the **invite your friends** button lets you bring your friends into the world of zembly.

Your Profile

Let's start with your profile. Your profile tells other people about you. Click **View your profile**, as shown in Figure 2.16. Your profile includes a picture, your descrip-

tion, your contacts, a list of all the things you own, and a time line that shows what you've been doing.



Figure 2.16 You can Edit or View your profile

You can set your screen name and code name. (You can only set your code name once.) The code name is used to group widgets, services, and applications that you contribute. For example, if your code name is **user1234** then people can call one of your published services (say "myservice") from a widget using something like

```
Things.callService("user1234.myservice")
```

Your screen name is a conversationally nice thing you want other people to call you.

People—Adding Contacts

Your contacts are people whom you invite to collaborate with you on creating widgets, services, or applications. Contacts are visible on your profile page. You can also view and search for contacts under the People tab on your home page as shown in Figure 2.17.



Figure 2.17 Viewing contacts under People

zembly encourages collaboration when creating and editing services and widgets. Before you can request someone to collaborate with you on a project, you must add them as a contact. You add them by viewing their profile page and selecting the **Add as Contact** button. Alternatively, search for them from the **People** tab. Type a word in the search box and hit Search. To add a person to your contacts, simply click the **Add to contacts** link below the person's name, as shown in Figure 2.18. If a person is already a contact, you'll see a message saying so.

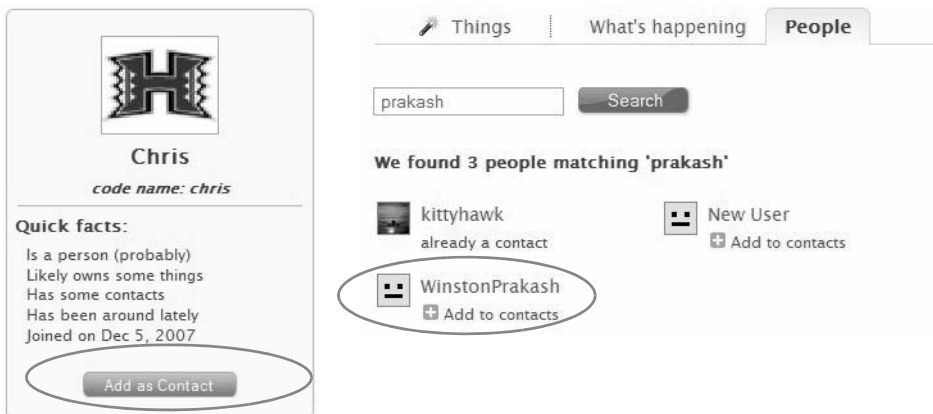


Figure 2.18 Adding contacts

Once you add contacts, you can then read what they've recently done through the news feed and add them as a contributor to one or more things that you own. To view the News Feed, select the **What's happening** tab as shown in Figure 2.19.

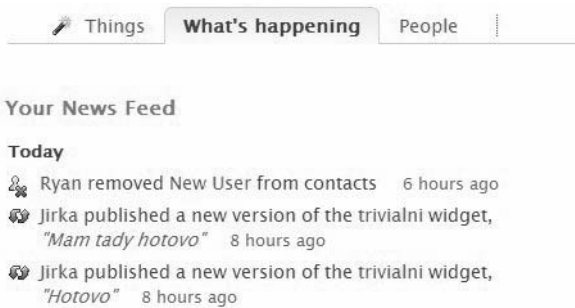


Figure 2.19 Your News Feed reports what your contacts are doing

2.3 Your Keychain and Service Providers

Under your home page (click **You** at the top of any page) you'll find your Keychain (click **Manage your keychain** on your home page, as shown in Figure 2.20). Your Keychain is a list of keys that are associated with select service providers. Service providers have *adapters* on the zembly site. Adapters are wrapper services deployed in the zembly container that provide access to one or more of the Service Provider's API calls. Adapters make using your key a simple matter of specifying your Keychain—zembly extracts the appropriate key for the specific adapter seamlessly behind the scenes.



Figure 2.20 Accessing your Keychain

When you access your Keychain, zembly lists all of the service providers that have adapters. For each service provider that you want to use in a service, specify your key. Note that you need to obtain the key on your own first. The process is slightly different for each service provider, but is usually quick. Service providers typically email

you a confirmation. Once you have a key, you enter it into your Keychain using the **Add key** link (as shown in Figure 2.21).

Your Keychain is a very important and necessary part of building the web. You want to keep your keys handy, but you also want them private. **zembly** does this all for you. When other people call your published services, **zembly** uses your key (from your Keychain), but its value remains private.

You can see a list of adapters available by clicking the service's **Check out the services offered by** link. For example, you can see the services offered by Amazon AWS by clicking the link, as shown (circled) in Figure 2.21.



Figure 2.21 Building your Keychain for service providers

When you follow this link, you'll see the service adapters currently deployed within the **zembly** container, as shown in Figure 2.22.

You can further explore each service adapter by following its link to the detailed documentation page. Here, you'll find the service's parameters, error codes, and other pertinent information, which frequently includes external links to the provider's online documentation. "Putting It All Together—Using the WeatherBug API" on page 45 steps you through the process of building a service using one of **zembly's** external service providers.

2.4 Creating Your First Service: LoanPaymentService

Using some of the posted samples as guidelines, let's create a new service. You won't call an external service here; instead, you'll build one using JavaScript. A familiar example is a service that calculates one's monthly mortgage payment based on principal, interest rate, and length of loan (years).

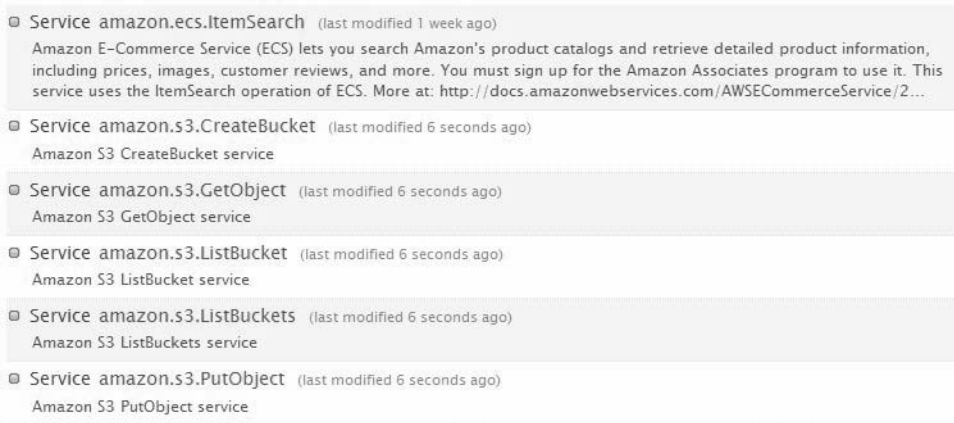


Figure 2.22 Amazon services include Simple Storage Service and E-Commerce Service (ECS)

Here’s a summary of the steps you’ll follow.

1. Create a new service. Give it a name and a description.
2. Add parameters to the service (optional).
3. Provide JavaScript code that returns data to the caller.
4. Add any error types (optional).
5. Test drive your service and modify as necessary.
6. Capture example return data (optional).
7. Publish your service.

Let’s start. To create a service, click Create something! at the top of the page and select **Service** as shown in Figure 2.23. You’ll see a new page that asks you to provide a description of the service. The default service name is `NewService`, which you should change to something meaningful. Many times, service names end in “Service,” but this is not a requirement. Call the service **LoanPaymentService**.



Figure 2.23 Creating a Service

This service requires three input parameters and returns a single numerical result. Error handling for input validation is handled completely by **zembly**; we discuss this further in the next section. Here's the JavaScript that provides the service.

Listing 2.2 LoanPaymentService (JavaScript)

```
// LoanPaymentService
// Input parameters are all NUMBERS and all Required

var principal = Parameters.principal;
var interest = Parameters.interest;
var interest_rate = interest / 1200;
var years = Parameters.years;

//Perform the calculation
var months = years * 12;
var x = Math.pow(1 + interest_rate, months);
var payment = (principal * x * interest_rate)/(x-1);

return payment.toFixed(2);
```

Specifying Parameters in a Service

When you create a web service, you tell the service interaction page about the parameters for your service. To add parameters, click **Add a new parameter** in the Call - Parameters window. You specify a parameter's characteristics in a dialog box.

When you add a new parameter you choose its type. By using the appropriate type, you take advantage of **zembly**'s built-in parameter validation. Table 2.1 lists the types supported.

TABLE 2.1 Parameter Types for Services

Type	Additional Fields	Examples
Binary	-	1101
Boolean	-	true, false
Email	-	info@buildtheweb.org
JSON	-	{"firstName": "John", "lastName": "Smith"}
Key	Keyset Provider	(Depends on provider)
Number (integer, real, or floating point)	Min Value, Max Value	55, 25.3
String	Max Length, Escape value	any string <= Max Length
URI	-	http://www.asgteach.com
XML	-	<firstName>John</firstName> <lastName>Smith</lastName>

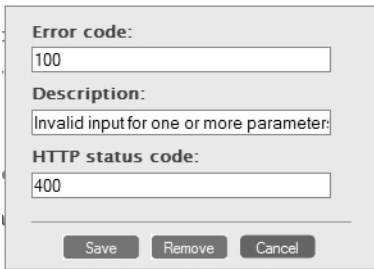
For this service, specify three parameters (principal, interest, and years). Make them all **required** and Type **Number**. With Number you also specify the minimum and maximum values. For principal use minimum **10** and maximum **2 million (2,000,000)**. For interest specify minimum **1** and maximum **20**. Finally, for years use minimum **1** and maximum **99**. Figure 2.24 shows the Parameter Editor for parameter years.



Figure 2.24 Creating and editing a web service parameter

Programming Tip

*If your service expects numbers for input, be sure to specify **Number** for the parameter type. The built-in parameter validation will verify the correct type and provide range checking as well. Figure 2.27 on page 30 provides an example result for out of range input.*



The image shows a dialog box with a light gray background and a thin border. It contains three text input fields stacked vertically. The first field is labeled 'Error code:' and contains the number '100'. The second field is labeled 'Description:' and contains the text 'Invalid input for one or more parameter:'. The third field is labeled 'HTTP status code:' and contains the number '400'. Below the input fields, there are three buttons: 'Save', 'Remove', and 'Cancel', each with a small icon to its left.

Figure 2.25 Creating and editing web service error codes

Note that if you make a parameter required, **zembly** flags an error if the caller doesn't provide a value. If you want the parameter to be optional, uncheck **Must use this parameter in the call**.

Once you've specified the parameters, you can access them in JavaScript. For example, you access the `LoanPaymentService` `principal` parameter with `Parameters.principal`.

zembly Tip

*It's a good practice to provide a description for parameters as shown in Figure 2.24. The description will then appear in your service's documentation page. It will also appear when you add code to call the service through **zembly**'s Find & Use feature (see "Calling LoanPaymentService in Your Widget" on page 39).*

Error Handling

When you detect a problem in your service, error codes can communicate status to the caller. You specify error codes in the service's Error Codes section. To add an error code, click **Add a new error type**. Figure 2.25 shows the dialog box that lets you specify a new error type. (You may also edit error codes that you have already defined.)

The error code, description, and HTTP status code all appear on your web service's documentation page. Note that you don't need to define an error code for the `LoanPaymentService`, since all error handling is performed by the built-in parameter validation.

Testing LoanPaymentService

Once you've built a service, you'll want to test it. Use the Call tab located next to the source editor window. You must provide values for any required parameters and click **Test drive now**. This calls the service with the parameter values you've provided and displays any results (or error codes) in the window. Figure 2.26 shows an example with a successful test.

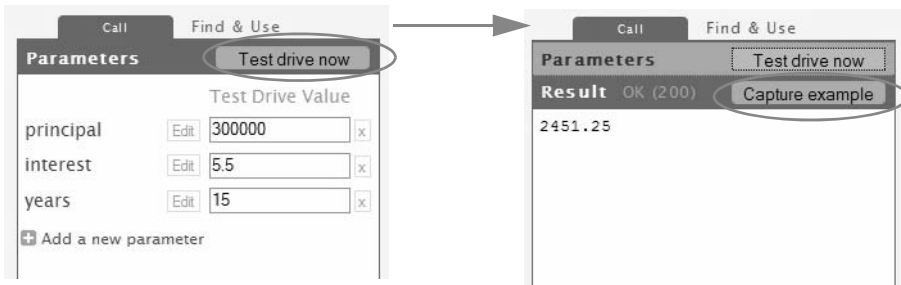


Figure 2.26 Testing a service

Figure 2.27 shows the built-in parameter validation when you provide a value outside the range for parameter years.

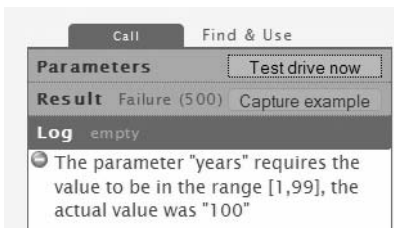


Figure 2.27 zembly's parameter validation

Capturing Example Return Data

To help others use your service, you can capture the return data after testing your service. Simply click the **Capture example** button (as shown in Figure 2.26). zembly creates a new heading on your service's documentation page and displays the output. This helps users, especially if the return data contains specific formatting (such as XML or JSON data). Figure 2.28 shows an example for LoanPaymentService.

Example Output

This is a sample of the plain text returned by this service.

```
2451.25
```

Figure 2.28 You can display sample output on your service’s documentation page

Saving Drafts

Each time you edit your code and test drive the service, your current code is automatically saved in a draft for you. **zembly** displays a small bar to indicate the current draft (the bars are displayed on the right with the most recent modification saved on top of the stack).

You can force a saved draft by clicking the Save Code icon at the bottom of the editor (or typing Ctrl+Alt+S). You can return to any previously saved draft or published version simply by clicking the bar. Also, you can see the timestamp and draft or version number by holding the cursor over the bar. See “Drafts, Versions, and Timelines” on page 42 for a more detailed discussion.

Using the JavaScript Editor

The JavaScript editor color codes key words, comments, and objects. The editor includes icon commands in the lower right window (as shown in Figure 2.29) to save your draft (Ctrl+Alt+S), toggle full screen editing (Ctrl+1), format code (Ctrl+Alt+F), undo editing (Ctrl+Z), redo editing (Ctrl+Y), or create a code snippet (Ctrl+Shift+N). You can also invoke code completion with Ctrl+Space.

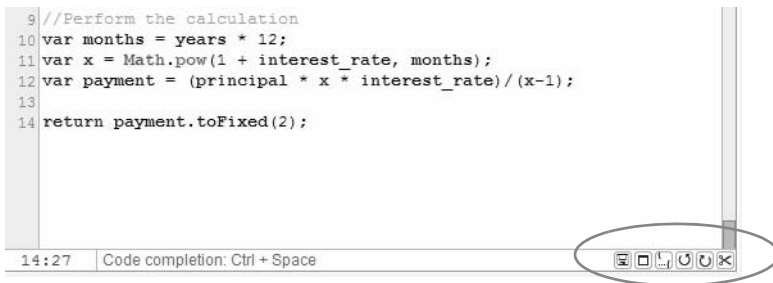


Figure 2.29 JavaScript editor command icons

Publishing LoanPaymentService

Click the **Publish** button to publish your service. This is the magic step that **zembly** provides to make services and widgets available to others. When you publish your service, **zembly** creates a deployable web service and deploys it in its own managed container. As you modify your service, **zembly** keeps track of drafts (unpublished edits) and versions (published edits). With each version you are encouraged to specify how the new version has improved (why it is cool).

Calling LoanPaymentService

Once you've tested and published your service, you'll want to call it from another service or widget. The page provides the code you need to call the service from another service, from a widget, or through the browser. However, the easiest way to call your service is to use **zembly's** Find & Use feature which automatically adds the code as a template in the editor. The Find & Use feature is context sensitive, so it will import the correct code depending on whether you're currently developing a widget or service. In addition, with Find & Use you'll see documentation about the service and its parameters.

Use the following (JavaScript) code to call your service from another service. **zembly** generates the comments for each parameter from the documentation you provide.

zembly Tip

*Note that this example calls the service using code name `ganderson`. When you create your own service, **zembly** uses your code name, which is unique to you.*

```
var result = Things.ganderson.LoanPaymentService({
  principal: 0, // The principal of the loan (in dollars)
  interest: 0, // The interest rate (per cent) (e.g., 6.5)
```

```
    years: 0 // How long your loan will endure (in years)
  });
```

Use the following template (JavaScript) code to call your service from a widget.

```
Things.callService("ganderson.LoanPaymentService",
{
  principal: 0, // The principal of the loan (in dollars)
  interest: 0, // The interest rate (per cent) (e.g., 6.5)
  years: 0 // How long your loan will endure (in years)
},
{
  onSuccess: function(data) {
    Log.write(data);
  },
  onFailure: function(error) {
    Log.write("Error: " + error.code + " : " + error.message);
  }
});
```

Programming Tip

*The statement `Log.write(data)` writes messages to the JavaScript debugger Firebug if it's installed. If not, **zembly** loads Firebug lite to get you started (use F12 to bring it up). Logging is enabled by default when you preview drafts and disabled for published versions. You can keep your `Log.write()` statements in the code. They will go to null (unless you enable debugging of published versions with the debug query parameter).*

A third way to call a service is to cut and paste the URL provided on the service page into the address line of your browser. Pasting the URL in your browser address line calls the service from HTTP. You must specify the parameter values in place of each [value] marker. Here is the LoanPaymentService URL with values replacing [value] in the URL.

```
http://zembly.com/things/2ef3f34205c84fca9b0d91c538fc6a5b;exec
?principal=232000&interest=4.5&years=15
```

zembly Tip

Note that you must delete the brackets when you specify the actual value for each parameter. However, if you accidentally leave in the brackets (in this example), the built-in parameter validation returns an error.

Figure 2.30 shows the browser window after calling the LoanPaymentService.

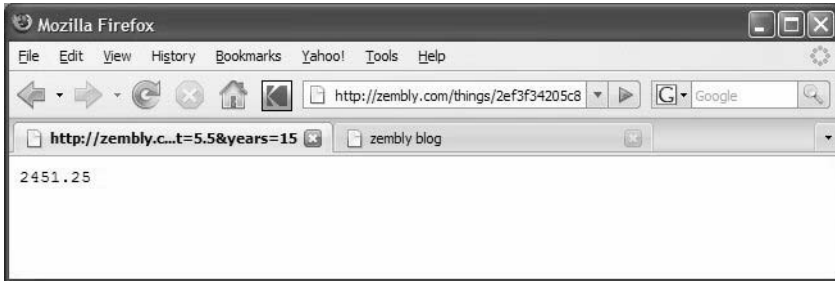


Figure 2.30 Calling a service with HTTP using its URL in the browser

2.5 Creating Your First Widget: LoanPaymentWidget

Now it's time to build a widget that uses the previously built `LoanPaymentService`. Here's a summary of the steps you'll follow to build a widget.

1. Create a new widget. Give it a name and a description.
2. Upload any resources, such as images (optional).
3. Include any libraries your widget uses.
4. Provide the HTML, CSS, and JavaScript code.
5. Use **Find & Use** to call **zembly** services from your widget.
6. Preview and publish.
7. Embed in a web page.

At the top of the page, select and then select **Widget**. **zembly** pops up a secondary dialog that lets you either select a template for your widget or simply create a blank widget. Select **Create a blank widget** as shown in Figure 2.31.



Figure 2.31 Creating a blank widget

zembly Tip

Widget templates are a recent addition to **zembly**. Templates let you choose a configurable starting point for building widgets. You navigate and find the template widget that's closest to what you want to build and click **Choose this template**. At this point the widget's code is visible but not editable. Use the form in the Configure tab to change the widget's look or behavior. If the customizations available do not cover your needs, you can edit the code by selecting button **Switch to edit mode**. Then, you can change anything you want in the CSS, JavaScript, and HTML source code, as well as libraries, resources, and so on. We encourage you to experiment with the various template categories before building a widget from scratch. For this example, however, you will start with a "blank" widget.

When you create a widget, you have the opportunity to supply three different files. You can also add images (which you'll do in this example) and include libraries in a separate step. You'll use (X)HTML for page markup, CSS for style specifications, and JavaScript for program logic. Figure 2.32 shows LoanPaymentWidget, the target widget that you'll build, running in a browser.

This widget includes an image, three input text fields with labels arranged in a table, a Calculate Payment button, and an output field that displays the result returned from the service.



Figure 2.32 LoanPaymentWidget running in a browser

Uploading an Image

To include an image with your widget, you upload it to **zembly**'s servers. To upload the image, click **Browse** in the Resources dialog. Navigate and select an image from your file system. After you click **Upload**, the image appears in the Resources window, as shown in Figure 2.33. Now click on the image to add it to your widget as an `` tag.

zembly Tip

*The easiest way to do this is to preview the already-built LoanPaymentWidget. First, specify LoanPaymentWidget in the **zembly** Search window and click on LoanPaymentWidget in the results list. While the widget is running in the preview area, right-click the image and save it to your local machine. You can then upload the image as described above.*



Figure 2.33 Uploading images to add to your widget

After selecting the image, the HTML code includes an `` tag as follows. (Select the **(X)HTML** tab.)

```

```

You'll move the `` tag inside the outermost `<div>`, as shown in Listing 2.3.

Including Library Prototype JS

From the Resources tab, select **Libraries** at the bottom. Select **Prototype** from the list of libraries, as shown in Figure 2.34. Prototype is a general-purpose JavaScript library that includes functions (such as enhanced array iteration) and syntax shortcuts for DOM elements.

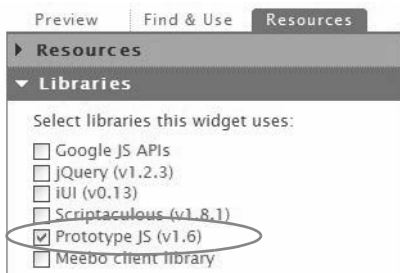


Figure 2.34 Including the Prototype JavaScript library in your widget

Building LoanPaymentWidget

Listing 2.3 shows the HTML code for this widget. The input elements are organized within a table element to create symmetrical spacing. The <div> tag with id="resultDiv" holds the results returned from the LoanPaymentService; the <div> tag with id="errorDiv" holds any returned error messages.

Listing 2.3 LoanPaymentWidget (HTML)

```
<div id="loanDiv" class="widget">
  
  <div id="headingDiv" class="heading">
    Loan Payment Calculator
  </div>
</table>
<table>
  <tr><td>Principal:</td>
    <td>
      <input id="principal" type="text" size="20" value="300000">
    </td></tr>
  <tr><td>Interest:</td>
    <td>
      <input id="interest" type="text" size="20" value="6.0">
    </td></tr>
  <tr><td>Term (years):</td>
    <td>
      <input id="years" type="text" size="20" value="30">
    </td></tr>
  <tr><td colspan="2">
    <div id="calcdiv" class="calc">
      <button id="calcButton">Calculate Payment</button>
    </div>
  </td></tr>
  <tr><td colspan="2">
    <div id="resultDiv" class="results">
    </div>
    <div id="errorDiv" class="errorResults">
    </div>
  </td></tr>
</table>
</div>
```

Using CSS for Styling

The widget's CSS file provides style sheets for the widget. Note that there is a separate style for results and errorResults. Here is the CSS code.

Listing 2.4 LoanPaymentWidget (CSS)

```
div.widget {
  background-color: #e6e6ff;
  border: 1px solid #aaaaff;
  padding: 5px 5px 5px 5px;
  font-size: 0.8em;
}
div.heading {
  font-size: 1.3em;
  font-weight: bold;
  text-align: center;
}
div.calc {
  margin: 5px;
  text-align: center;
}
div.results {
  margin: 5px 2px 2px 2px;
  padding: 1px 2px 2px 2px;
  font-weight: bold;
  text-align: center;
}
div.errorResults {
  margin: 5px 2px 2px 2px;
  padding: 1px 2px 2px 2px;
  font-weight: bold;
  color: #C61C1C;
  text-align: center;
}
}
```

Calling LoanPaymentService in Your Widget

zembly makes it easy for you to add code to call services in your widget (this works when building services, too). Open the JavaScript editor for your widget (click the JavaScript tab in the editor zone). Then select the **Find & Use** tab in the window to the right of the editor zone. You'll see a Search window. Type in LoanPaymentService and click **Search**.

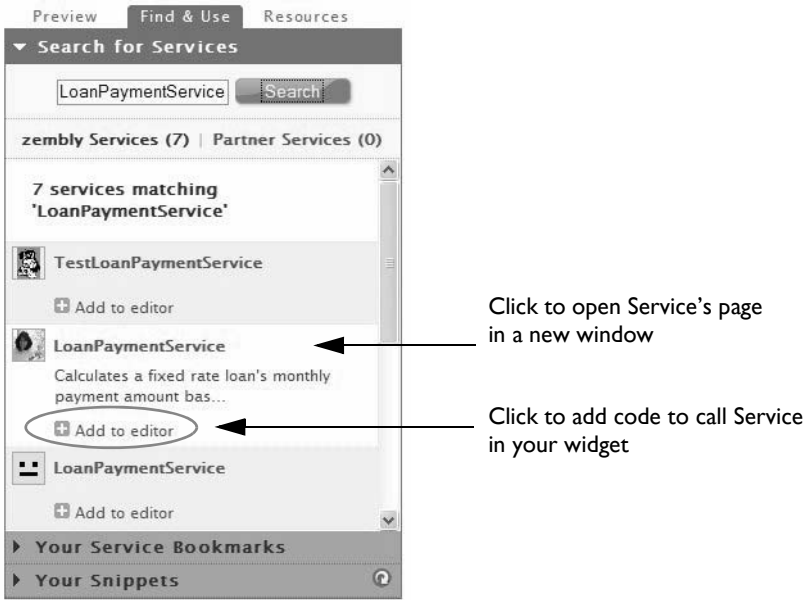


Figure 2.35 Find & Use Search for Services lets you easily add code to your widget

Figure 2.35 shows the results (quite a few, as it turns out). Using the avatars as a clue, find the `LoanPaymentService` near the start of the list. If you click its description, zembly opens the `LoanPaymentService` page in a new window. You can then study its documentation (or code). To add code to call the service to your widget, simply click **Add to editor** in the search results window. This will give you the correct calling template as a starting point. You can then edit the JavaScript to specify the parameter values and add the coding logic for your widget.

Listing 2.5 shows the JavaScript that calls `LoanPaymentService` using the three input values. Prototype's `Event.observe` function connects the `calcButton` click event to the handler.

After obtaining the principal, interest, and years input from the user, you pass these parameters to the `LoanPaymentService`. The return value is in `data`, which you prepend with a dollar sign (\$) (for a successful payment result) or `error`, which returns an error code (`error.code`) and message (`error.message`). You insert the payment or the error information into the page's HTML markup.

```
$("#resultDiv").innerHTML = [insert payment html here];  
$("#errorDiv").innerHTML = [insert error html here];
```

Because `style errorDiv` defines its text color as red as shown below (and in Listing 2.4 on page 39), error messages appear in red and normal return results are in black.

```
div.errorResults {
  margin: 5px 2px 2px 2px;
  padding: 1px 2px 2px 2px;
  font-weight: bold;
  color: #C61C1C;
  text-align: center;
}
```


The Prototype shortcut notation uses `$("#element_id")` instead of the more verbose `document.getElementById("element_id")`.

Listing 2.5 LoanPaymentWidget (JavaScript)

```
Event.observe($("#calcButton"), 'click', function() {
  var principal = $("#principal").value;
  var interest = $("#interest").value;
  var years = $("#years").value;

  Things.callService("ganderson.LoanPaymentService",
  {
    "principal": principal, // The principal of the loan (in dollars)
    "interest": interest, // The interest rate (per cent) (e.g., 6.5)
    "years": years // How long your loan will endure (in years)
  },
  { onSuccess: function(data) {
    var resultsHtml = "$" + data;
    $("#resultDiv").innerHTML = resultsHtml;
    $("#errorDiv").innerHTML = "";
  },
    onFailure: function(error) {
    $("#errorDiv").innerHTML = error.code + ": " + error.message
      + "<br/>";
    $("#resultDiv").innerHTML = "";
  }
  });
});
```

Previewing and Publishing

Test your widget using the Preview tab or the Preview widget box. When you're satisfied that the widget is working, publish your widget by selecting the  button. After you publish a widget, you can embed it in any web page.

Embedding LoanPaymentWidget

This widget is embedded in a web page with the following HTML source.

```
<iframe width=400 height=280
  src="http://94f52847744e493b944aed46cf255e63.zembly.com/things/
    94f52847744e493b944aed46cf255e63;iframe" frameborder="0">
</iframe>
```

You copy/paste the source from the widget's documentation page (under Share This Widget). Here, we've supplied height and width attributes to adjust the size of the iframe tag area.

zembly Tip

You can add width and height attributes to the iframe tag as shown above to make the widget larger than the default iframe size.

2.6 Drafts, Versions, and Timelines

When editing a service or widget, **zembly** records the changes you make as intermediate drafts. **zembly** automatically saves a new draft every time you make a change, such as adding a parameter or editing the code. A draft is a clone of the entire state of your service or widget. A draft comes to an end when you publish a service or widget. You can see which version you are looking at or editing below your object's title, as shown in Figure 2.36.



Figure 2.36 Draft and version number of object's current edit session

Edit History

A stack on the right side of the edit zone (small boxes) shows the history of your published versions and drafts, as shown in Figure 2.37. The oldest versions are on the bottom of the stack and, if you hover the mouse over a box, you can see when **zembly** saved the draft (light box) or the published version (dark box).

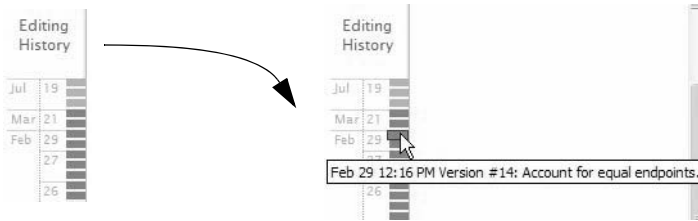


Figure 2.37 Edit history is a stack of published versions and drafts

If you look quickly when you make a change to the service or widget, or when you click around on the page after making a change, you'll see a new box being added to the top of the stack, as shown in Figure 2.38.

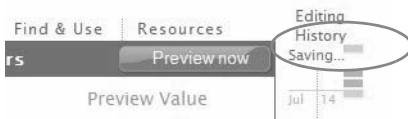


Figure 2.38 Saving a new draft

When you click on one of the boxes, the current changes are saved (as a new draft) and the old draft or version you clicked on is loaded. This lets you move back and forth within your edit history seamlessly. Once you begin editing by changing something, zembly creates a new draft based on the draft you changed.

Any changes you make are related to your current draft, which is the thing you're working on before you've published. You can keep multiple drafts for any length of time. You only create a new version of a service or widget when you publish. Then your edit history for the draft is wiped out (along with the current draft, which converts to a published version). At this point, you start over and any changes are saved as new drafts based on the latest published version.

You can also remove all current drafts of a service or widget if you don't want to keep any of the changes saved for your code. Select **Erase and start over**, as shown in Figure 2.39. zembly confirms the action before removing the drafts.

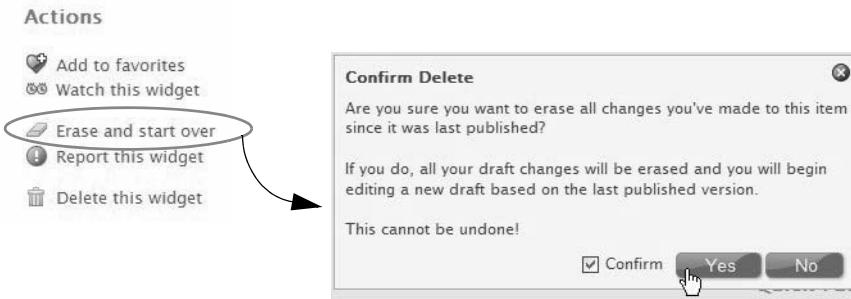


Figure 2.39 Erase and start over removes all saved drafts

Viewing Versions

The timeline lets you look at older versions of widgets or services as shown in Figure 2.40.

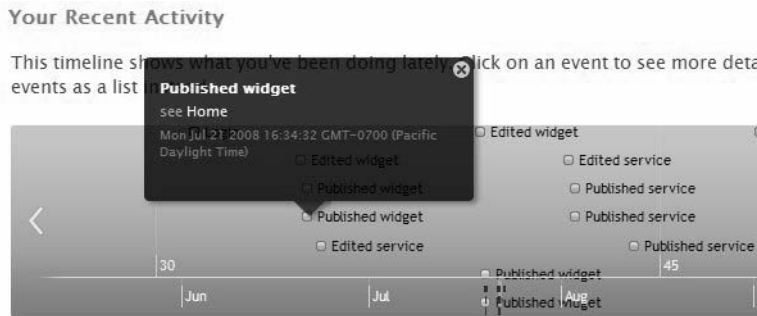


Figure 2.40 Finding a previous version of a published object

When you click on one of the points on the timeline, zembly reloads the page with that version.

Online/Offline Status

You can bring a published service or widget offline by toggling its online/offline status indicator as shown in Figure 2.41. Currently, this affects all published versions of

the item. In the future, you will be able to specify individual versions for offline publishing.

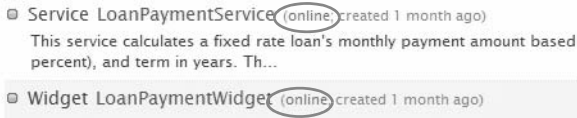


Figure 2.41 Online/offline toggle for a service or widget

2.7 Putting It All Together—Using the WeatherBug API

You'll now build a service and widget that uses the WeatherBug API. (Note that *zembly* already provides a sample service and widget for the WeatherBug API. However, in this section you'll build your own.) Here's a summary of the steps you'll follow to build this service and widget.

zembly Tip

*As with all the examples throughout the book, you are encouraged to clone the examples (find them using the **zembly** search mechanism). We provide the steps here so that you can easily build services and widgets on your own.*

Steps to create the WeatherBugService

1. Obtain a WeatherBug API key (from WeatherBug) and add it to your *zembly* Key-chain.
2. Create a new service to access the WeatherBug API.
3. Add a parameter to the service.
4. Using **Find & Use**, add the JavaScript code to call the WeatherBug API.
5. Format the return data.
6. Test and publish the service.

Steps to create the WeatherBugWidget

1. Create a new blank widget.
2. Provide HTML for formatting.

3. Using **Find & Use**, add the call your WeatherBug service.
4. Include the Prototype library.
5. Add JavaScript code.
6. Preview and publish the widget.
7. Embed it in a web page.

Let's begin. The WeatherBug API is included in the list of services you can access from **zembly**. These services have been wrapped as adapters by the **zembly** structure, giving you easy access to WeatherBug's API.

Using Your Keychain

To start, access your Keychain (click **Manage your Keychain** from your home page). This displays the service providers currently supported on **zembly**. Scroll down to the WeatherBug's listing, as shown in Figure 2.42.



Figure 2.42 Accessing WeatherBug's registration page

The first step is to register with the WeatherBug service to obtain a key. You do this directly with WeatherBug. (Click **Register with WeatherBug** to get started.) After several email confirmations, WeatherBug will send you a key. When you add the WeatherBug API key to your keychain, **zembly** prompts for your name. (Each API site has its own requirements for what constitutes a legal API key.) Your name is stored with the key in your private keychain data.

zembly knows this key is associated with the WeatherBug API. When you access the WeatherBug API using `owner.keychain`, the WeatherBug key is used in exactly the correct format required by WeatherBug. Also, when other people use a service you build, they access the service using your key. Your key is protected since it is wrapped in the Keychain mechanism.

Building WeatherBugService

Now that you have a WeatherBug key, you can build a service. From your Keychain page, scroll down to the WeatherBug listing and click **Check out the services offered by WeatherBug**. This directs you to the list of services; currently LiveWeatherRSS is the only one. Now click on **WeatherBug.LiveWeatherRSS** and zembly directs you to the LiveWeatherRSS page, as shown in Figure 2.43.

The WeatherBug.LiveWeatherRSS information page describes how to call the adapter and use its data. Besides the description, the page provides information on the parameters and error codes and a text box tells you how to call the adapter from a service.

Check out the services offered by WeatherBug.



<http://zembly.com/things/2b63038e48144bb5a8519fd225fecb87>

Figure 2.43 Drilling down to check out the services offered by WeatherBug


Here's the code template to access the LiveWeatherRSS adapter.

```
var result = Things.WeatherBug.LiveWeatherRSS({
  zipcode: "", // 5-digit ZipCode, U.S. cities only
  // unittype: "", // Optional. Default value is 1. Values are 0
  // (US customary units) or 1 (metric units - kms, degrees Celsius, etc).
  keys: Owner.keychain
});
```

What Are Things?

zembly provides the Things object in the environment. It represents all artifacts (services, widgets, and applications) that are potentially accessible. Group WeatherBug specifies the publishing owner. Currently LiveWeatherRSS is the only service available under this group.

The LiveWeatherRSS API call takes three parameters: the target zip code, the unit type (this is optional, but it defaults to '1' which means metric units), and keys. When you build this service, you'll supply values for unit type and keys and allow the user to supply the target zip code.

Now, let's build the service. Click  at the top of the page and choose **Service** from the drop down menu. For this service, you'll have a single parameter (the target zip code). The unit type defaults to "1" (metric units) but you'll specify U.S. units (such as Fahrenheit and inches) which is "0". For the keys data use `Owner.key-chain`, which pulls your WeatherBug-specific API key. You write this service using JavaScript. The LiveWeatherRSS service returns XML. You'll extract just the data you want and return the results to the caller in JSON.

Using E4X and JavaScript

zembly returns XML data as a DOM document object. This means that you can access the object directly in your JavaScript using E4X notation.

zembly Tip

*Although E4X support is not standard in all browsers, it is supported in the **zembly** environment. Therefore, you can use E4X without sacrificing portability in the services you build. However, for maximum portability, avoid using E4X in widgets. (This is why this service returns results in JSON.)*

Listing 2.6 shows a sample of the XML response from WeatherBug. Let's show you how to access the various nodes using JavaScript and E4X.

Listing 2.6 Sample XML Response from WeatherBug

```
<rss version="2.0">
  <channel>
    <title>Observations from Encinitas, CA - USA</title>
    <link>...</link>
    <description> . . . (contains HTML code) . . . </description>
    . . . omitted data . . .
  </channel>
</rss>
```

Because the data is already returned as a DOM document object, you access node `title` using (for example),

```
data.channel.title
```

As discussed earlier, you provide a single parameter (“zipcode”), which is a String, it’s required, and is escaped, as shown in Figure 2.44.

zembly Tip

Only a limited subset of characters are allowed in URLs. Escaping means that any special characters such as space, quotation marks, or the ampersand sign are encoded and then unencoded within the service. Typically characters must be encoded because they have a special meaning within a URL (such as ; ? or =) or there is a possibility of misinterpretation (such as space or #).



Parameter Editor

Name:

Description:

Must use this parameter in the call

Type:

Max Length:

Escape value

Figure 2.44 Editing a parameter

To add the code you need to call the adapter to the editor, select the **Find & Use** tab in the box to the right of the editor. Specify **LiveWeatherRSS** and click **Search**. zembly returns the matching services. Click **Add to editor** and zembly adds the code you need to the editor, as shown in Figure 2.45.



Call Find & Use

Search for Services

zembly Services (1) | Partner Services (0)

1 service matching 'LiveWeatherRSS'

WeatherBug.LiveWeatherRSS

The LiveWeatherRSS method retrieves weather for a specifi...

Figure 2.45 Find & Use lets you search for services and add code to the editor

Listing 2.7 shows the JavaScript source for WeatherBugService. Function `Log.write` allows you to write information to a log file that you can view in the Call window to the right of the editor. Select **Log** at the bottom of the Call window to view. Here, function `Log.write(typeof data)` writes “xml” to the log file.

Using the E4X notation, the WeatherBugService extracts the data for the title, description, and link to pass to the caller. This is the data you’ll work with when you create a widget that uses this service.

Listing 2.7 WeatherBugService (JavaScript)

```
var data = Things.WeatherBug.LiveWeatherRSS({
  "zipcode": Parameters.zipcode, // 5-digit ZipCode, U.S. cities only
  "unittype": "0",
  "keys": Owner.keychain
});

//log type of the result object (XML object)
Log.write(typeof data); // writes "xml"

//You can use E4X notation to access the elements and attributes
//inside this object directly

var result = new Object();

result.title = ""+data.channel.item.title;
result.description = ""+data.channel.item.description;
result.link = ""+data.channel.link;

// Returns JSON
return result;
```

As you build the service, you can test drive it and look at the results that are returned.

Calling WeatherBugService

The next step is to build a widget that uses this service. The point of building a widget is to create a user-friendly snippet of code that others can paste directly into a web page. This widget should provide nice formatting of the data returned by WeatherBugService. Before leaving the WeatherBugService page, you’ll see that it tells you how to call this service from a widget. Here is the template code zembly provides.

```
Things.callService("ganderson.WeatherBugService",
  {
    zipcode: "" // 5-digit ZipCode, U.S. cities only
  },
  {
```


```
    onSuccess: function(data) {
        Log.write(data);
    },
    onFailure: function(error) {
        Log.write("Error: " + error.code + " : " + error.message);
    }
});
```

When you build your widget, you'll use this code to call the service. Let's take a brief look at the response you get when you make a successful call to a service (the `onSuccess` handler). The code within `Things.callService` examines the response and gives you a pointer to the data object directly (as the first argument).

- If the service returns an object (if the response content type is *application/json*), then `data` is a JavaScript object. The `WeatherBugService` returns JSON data to the calling widget.
- If the service returns an XML document (if the response content type is *application/xml*), then `data` is a DOM object.
- If the service returns a plain string, number, boolean or date, then the `data` object will have its string representation. The `LoanPaymentService` returns a number, which will have its string representation in the calling widget.

Note that the `onFailure` handler accesses the error object.

Building WeatherBugWidget

From the top of the page, click  and select **Widget** from the drop down menu. Choose **Create a blank widget**. The widget page lets you rename the widget, provide a description, and specify HTML, CSS, and JavaScript code. The HTML code should provide an input field for the zip code and a button to click and grab the weather data. You'll also need a named `<div>` tag to display the results. (This is `id="weatherBugResults"` in Listing 2.8 below.)

Here is the HTML code for the `WeatherBugWidget`.

Listing 2.8 WeatherBugWidget (HTML)

```
<div id="weatherBugWidget">
  Please enter a ZipCode to search WeatherBug service: <br/>
  <input id="zipcode" type="text" value="92024" />
  <br/>
  <button id="weatherButton">Get Weather</button>
  <br/><hr/><br/>
  <div id="weatherBugResults"></div>
</div>
```

Sample JSON Output

Before you look at the JavaScript code for this widget, let's look at the data that the service returns. Listing 2.9 shows sample JSON output (property description has been shortened). The result object is embedded in curly braces { } and each property is identified in quotation marks followed by a colon and its value. Properties are separated with commas.

Because the data arrives to the caller as a JavaScript object, you do not need to perform any parsing. For example, the title property is accessed using (for example)

```
data.title
```

Listing 2.9 Sample JSON Output

```
{
  "title": "Live Conditions from Encinitas, CA - USA",
  "description":
"<img src=\"http://deskwx.weatherbug.com/images/Forecast/icons/cond026.gif\"
  border=\"0\" alt=\"Current Conditions\"/>

  ( . . . data omitted . . . )

  <br />",
  "link":
"http://weather.weatherbug.com/CA/Encinitas
weather.html?ZCode=Z5546&Units=0&stat=ENCNT"
}
```

WeatherBugWidget JavaScript

This widget doesn't specify any CSS. Listing 2.10 shows the JavaScript code for this widget. Note that each property in the returned data object (data) is accessed directly using JavaScript notation. Since the data in property description is straight HTML, you can use that directly in the HTML markup.

zembly Tip

The JavaScript code in Listing 2.10 relies on the Prototype Library. Be sure to select Prototype from the list of libraries under the Resources/Libraries tab.

Listing 2.10 WeatherBugWidget (JavaScript)

```
// WeatherBugWidget (WBW)
// Register a listener for the "weatherButton" with Prototype Event.observe
```

```

Event.observe($("#weatherButton"), 'click', function() {
  var zipcode = $("#zipcode").value;
  // call service and pass zipcode
  Things.callService("ganderson.WeatherBugService", {
    zipcode: zipcode},
    {
      onSuccess: function(data) {
        // format the return data and inject into page markup
        var resultsHtml = "<b>" + data.title + "</b><br/>" +
          data.description + "<br/><b><a href=" + data.link +
            ">weather details</a></b><br/> " ;
        $("#weatherBugResults").innerHTML = resultsHtml;
      },
      onFailure: function(error) {alert(error.code);}
    });
});

```

Each time you edit the widget, you can test it directly on the widget page. Once you're finished editing, publish it. **zembly** provides the code you use to run the widget in a browser. You can configure the `iframe` by specifying height and width attributes. Here is the HTML source.

Listing 2.11 Sample HTML source to call the WeatherBugWidget

```

<iframe width=500 height="350"
  src="http://0eb6e21170b8405ca2658cec54fc5005.zembly.com/things/
0eb6e21170b8405ca2658cec54fc5005;iframe" frameborder="0">
</iframe>

```

Figure 2.46 shows sample output from running the above HTML.

zembly Tip

See “LiveWeatherBugWidget” on page 274 and “LiveWeatherMapWidget” on page 284 for enhancements to this widget.

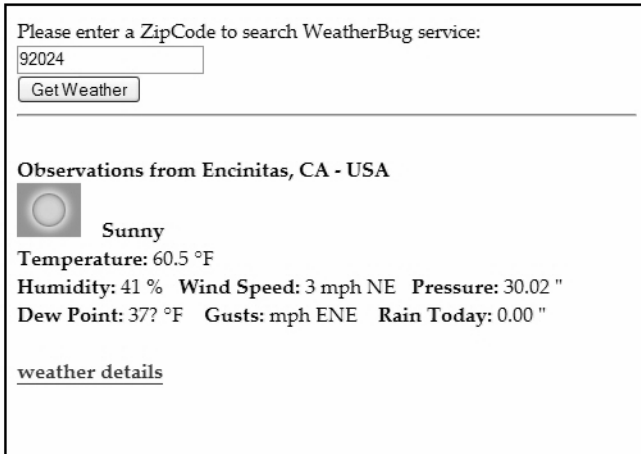


Figure 2.46 WeatherBugWidget running in a browser

Index

Symbols

`$("#element_id")` Prototype notation 41

A

About Page, Facebook 125
acquire Facebook session, **zembly** testing 155
actions, **zembly** 16
 `{*actor*}` token 183
adapters, **zembly** 24
Add to editor, Find & Use Tab 40
Add to Home Screen, iPhone icons 317
Add to Info button, Facebook 114, 212
Add to Profile button, Facebook 141, 144, 176
 with HTML 166
`addOverlay` function, Google Maps 97
admin tool, Facebook data store model 193
`Ajax.FBML`, FBJS (`responseType`) 143, 150
`Ajax.JSON`, FBJS (`responseType`) 217
allow access, Facebook applications 111, 136
Amazon AWS services 25
AmazonProductSearch sample service 13
`AmazonProductSearchWidget` 17–20
animation library, Facebook 151
API calls, **zembly** 24
API key, Facebook 118
Apple iPhone Developer 356
application context, Facebook 134
Application Directory, Facebook 130
Application Info section, Facebook 212, 219
Application object, **zembly** 134
Application Tabs, Facebook 110, 223
applications
 See also Facebook applications
 create Facebook 114–120
 favorites 16
 score 15
Applications menu bar, Facebook 106
arrays with FBJS 158
associations, Facebook data store model 192
attributes, XML node 277

B

Binary parameter type for services 28
bookmark, Facebook applications 107, 108
Boolean parameter type for services 28
Boxes tab, Facebook 108

BuddyMugs Facebook application, HTML 160
BuddyPics Facebook application 137–158
 Home Widget (*JavaScript*) 148
 service `GetFriendInfo` 153
 service `UpdateProfileBox` 157

C

call service, **zembly** 32
Call tab, **zembly** 30
canvas page URL, Facebook 117
canvas page, Facebook 104
Capital Punishment Facebook application 163–190
 `countrydata.js` file 165
 data store model 193
 DeleteScore service 204
 detecting users who have authorized 172
 FBMLGetFriendScores service 201
 FBMLGetMyScores service 201
 feed stories 182–186
 getSessionKey service 348
 Home Widget (*JavaScript*) 170
 iPhoneHome widget 337–348
 PublishChallenge service 190
 PublishScore service 186
 SeeScores widget 205
 SendInvitation widget 187
capture example, **zembly** 30
`clearInterval` JavaScript function 63
Clearspring widget sharing 18, 66
`clearTimeout` JavaScript function
 with FBJS 147
clone, **zembly** 15
 Facebook applications 142
code name, **zembly** 22
code testing tools, Facebook 133
collaboration requests, **zembly** 21
comma formatted numbers, JavaScript 90
contacts, **zembly** 21, 22–24
 search 23
context data, Facebook applications (*table*) 135
cookie helper functions, JavaScript 330
count-down timer, JavaScript 170
CreateDataStore service, Facebook data store
 model 196
CSS styles 35, 59
 hover events 60
 opacity 62

D

- Dapp 238
 - create 240–243
 - feed reader 266
 - flickrPhotoSearch 240–243
 - GambitsfromGailFeed 267
 - LondonTubeJourneyPlanner 250–252
 - mlbupdate 259–262
- Dapper 239
 - flickrPhotoSearch service 243
 - flickrPhotoSearchWidget 244–248
 - GambitsfromGailFeed service 268
 - GambitsSummaryWidget 268–271
 - LondonTubeJourneyPlanner service 252–253
 - LondonTubeMapWidget 289–298
 - LondonTubeWidget 254–258
 - mlbscores service 262
 - mlbScoresWidget 263–265
- dapper.net 238
- data extraction 239
- data store model, Facebook 192
- Developer, Facebook application 116
- Dialog FBJS object 159
- directional controls, Google Maps 94
- drafts, **zembly** 31, 42–45
 - Erase and start over 43
 - saving new 43

E

- E4X notation 48, 84, 276
 - namespace 277
- each Prototype function 63
- ECMAScript for XML 84
- EDGE network, iPhone 310
- editor commands, **zembly** 31
- Email parameter type for services 28
- email, Facebook 112
- embed a widget 14
- enumeration with Prototype 63
- Erase and start over (all drafts) 43
- Error Codes, services 29
- Escape value 49
- Event.observe Prototype function 64
- examples
 - BuddyPics Facebook application 137–158
 - Capital Punishment Facebook application 163–190
 - Chapter Building Flickr Widgets 56
 - Chapter Building for the iPhone 310
 - Chapter Building Zillow Widgets 80
 - Chapter Facebook Basics 102
 - Chapter Facebook Integration 162
 - Chapter Widget Gallery 274
 - Chapter Working with Dapper 238
 - Chapter **zembly** Basics 12
 - FlickrPeopleService 68–71
 - flickrPhotoSearch Dapp 240–243
 - flickrPhotoSearch service 243
 - flickrPhotoSearchWidget 244–248
 - FlickrSlideShow 57–67
 - GambitsfromGailFeed Dapp 267
 - GambitsfromGailFeed service 268
 - GambitsSummaryWidget 268–271

- iButtons widget 359–361
- iCapital Punishment widget 331–337
- iLeaves widget 356–359
- iLiveWeather widget 322–330
- iLoanPayment widget 314–322
- iLondonTube widget 348–356
- iPhoneHome widget 337–348
- LiveWeatherBugService 276–279
- LiveWeatherBugWidget 279–283
- LiveWeatherMapWidget 284–289
- Loan Calculator Facebook application 121–125
- LoanPaymentService 25–34
- LoanPaymentWidget 34–42
- LondonTubeJourneyPlanner Dapp 250–252
- LondonTubeJourneyPlanner service 252–253
- LondonTubeMapWidget 289–298
- LondonTubeWidget 254–258
- mlbscores service 262
- mlbScoresWidget 263–265
- mlbupdate Dapp 259–262
- Mood Pix Facebook application 208–231
- MyFlickrRandomSlideshow 71–77
- RecentSalesMashup 92–99
- RecentSalesService 82–88
- RecentSalesWidget 88–92
- WeatherBugService 45–51
- WeatherBugWidget 51–54
- zemblyblog pipe 299–304
- zemblyConnectDemo Facebook application 232–235
- zemblyrumblings widget 304–307

F

- Facebook
 - background* 103–114
 - acquire session 155
 - Add to Info button 114, 212
 - Add to Profile button 141, 166, 176
 - allow access dialog 111
 - animation library 151
 - API 131
 - API key 118
 - API test console 133
 - application context 134
 - Application Directory 130
 - Application Tabs 110
 - Applications menu bar 106
 - Boxes tab 108
 - canvas page 104
 - canvas page URL 117
 - code testing tools 133
 - Connect facility 231–235
 - create applications 114–120
 - Data Store API 191–208
 - data store model 192
 - detecting users who have authorized 172
 - Developer application 116
 - email 112
 - fb_sig_added 134, 216
 - fb_sig_canvas_user 134, 217
 - fb_sig_friends 134, 147
 - fb_sig_user 134, 217
 - FBJS 120
 - FBML 120

- overview 138
 - dynamic content 142–143
 - test console 133
- fbSessionKey 134
- fbUserID 134, 177
- features and integration points (*table*) 104
- feed forms 211
- feed preview console 133
- feed story types 183
- feed.publishUserAction 186
- FQL 199
- friend 105
- Home link 113
- Info section 212
- Info tab 114
- invite friends widget 187
- iPhoneHome widget 337–348
- JavaScript Client Library 233
- left-hand column 109
- locked services and widgets 120
- news feed 112
- notices 112
- preview widget 124
- profile 105
- profile boxes 108, 180
- profile publisher 113
- profile.getInfo adapter 221
- profile.setInfo adapter 221
- publishing feed stories 182–186
- Registered Template Bundles Console 182
- registered templates console 133
- secret key 118
- session authorization 136, 339
- template bundles 182
- tokens, feed templates 183
- user 105
- XFBML 233
- Facebook applications
 - About Page 125
 - Add to Profile button 144
 - allow access and login techniques 136
 - bookmark 107, 108
 - BuddyPics 137–158
 - Capital Punishment 163–190
 - clone 142
 - context data (*table*) 135
 - developer mode 129
 - FriendChooser 191
 - Help widget 126
 - Home widget 124
 - HTML widgets 121
 - icon 128
 - Loan Calculator 121–125
 - logo 129
 - Make it public button 129
 - Mood Pix 208–231
 - permissions 108
 - private installation 129
 - PublishScore service 186
 - registered template bundles 189
 - RegisterTemplates service 184
 - requireLogin, FBJS 137, 206
 - responseType, FBJS 143, 150
 - template widgets and services 126
 - Terms of Service widget 126
 - UpdateProfileBox service 126, 181
 - ways to enhance 127–129
 - zemblyConnectDemo 232–235
- Facebook Connect 231–235
 - zemblyConnectDemo application 232–235
- Facebook data store admin tool 193
- Facebook Data Store API 191–208
- Facebook data store model 192
 - associations 192
 - CreateDataStore service 196
 - delete object 204
 - FQL 199
 - object types 193
 - properties 192
 - SaveScore service 198
- Facebook Developer application 116
- Facebook Developers Wiki 133
- Facebook JavaScript, *See* FBJS
- Facebook Markup Language, *See* FBML
- Facebook Query Language, *See* FQL
- Facebook services
 - acquire Facebook session 155
 - testing 155
 - UpdateProfileBox 157
- favorites, zembly 16
- fb:action FBML tag 187
- fb:add-section-button FBML tag 144, 214
- fb:board FBML tag 139
- fb:dashboard FBML tag 144
- fb:else FBML tag 139
- fb:friend-selector FBML tag 139, 227
- fb:help FBML tag 187
- fb:if FBML tag 139
- fb:if-is-app-user FBML tag 138
- fb:multi-friend-selector FBML tag 187
- fb:name FBML tag 138
- fb:profile-pic FBML tag 138
- fb:pronoun FBML tag 138
- fb:request-form FBML tag 187
- fb:rock-the-vote FBML tag 139, 144
- fb:user FBML tag 138
- fb_sig_added, Facebook 134, 216
- fb_sig_canvas_user, Facebook 134, 217
- fb_sig_friends, Facebook 134, 147
- fb_sig_user, Facebook 134, 217
- FBJS 120
 - differences with JavaScript 158
 - arrays 158
 - clearTimeout function 147
 - Dialog object 159
 - setInnerFBML function 143
 - setTimeout function 147
 - vs JavaScript 120
- FBML 120
 - overview 138
 - dynamic content 142–143
 - fb:action 187
 - fb:add-section-button 214
 - fb:board 139
 - fb:dashboard 144
 - fb:else 139
 - fb:friend-selector 139, 227
 - fb:help 187
 - fb:if 139
 - fb:if-is-app-user 138
 - fb:multi-friend-selector 187

- fb:name 138
- fb:profile-pic 138
- fb:pronoun 138
- fb:request-form 187
- fb:rock-the-vote 139, 144
- fb:user 138
- feed forms 215
- test console 133
- vs HTML 120
- FBML widgets 144
- FBMLGetFriendScores service, Facebook 201
- FBMLGetMyScores service, Facebook 201
- fbSessionKey, Facebook 134
- fbUserID, Facebook 134, 177
- feed forms, Facebook 211, 215
- FeedHandlerService 218
- feed reader Dapp 266
- feed stories, Facebook 182–186
- feed.publishUserAction Facebook adapter 186
- www.feedburner.com 267
- FeedHandlerService, Facebook feed forms 218
- Fetch Feed object, Yahoo! Pipes 300
- fieldset tag, iUI Library 319
- Filter operator, Yahoo! Pipes 302
- Find & Use tab, zembly 39, 49, 69
 - search Facebook adapters 131
 - Yahoo! Pipes 304
- Firebug debugger 33
- Flickr
 - FlickrPeopleService 68–71
 - flickrPhotoSearchWidget 244–248
 - FlickrSlideShow widget 57–67
 - MyFlickrRandomSlideshow widget 71–77
- Flickr API
 - key 68
 - photo source URL 64
 - sample JSON data 65
- flickr.interestingness.getList adapter 56
- flickr.people.findByUsername adapter 56, 68
- flickr.people.getInfo adapter 56
- flickr.people.getPublicPhotos adapter 56, 68
- flickr.photos.addTags adapter 57
- flickr.photos.comments.addComment adapter 57
- flickr.photos.geo.getLocation adapter 57
- flickr.photos.search adapter 57
- flickr.photosets.getList adapter 57
- flickr.photosets.getPhotos adapter 57
- FlickrPeopleService 68–71
- flickrPhotoSearch Dapp 240–243
- flickrPhotoSearch service 243
- FlickrPhotoSearchService sample service 13
- flickrPhotoSearchWidget (Dapp-based) 244–248
- FlickrSlideShow widget 57–67
 - (JavaScript) 62
- FQL 191–208
- fql.query Facebook adapter 154, 202
- friend, Facebook 105
- FriendChooser Facebook application 191
- full story type, Facebook 112, 183

G

- GambitsfromGailFeed Dapp 267
- GambitsfromGailFeed service 268

- GambitsSummaryWidget 268–271
- geocodes 92
 - conversion from degrees 171
 - tube station data 292
- gesture events, iPhone 311
- getSessionKey service, Facebook integration 348
- GMap2 Google Maps function 93, 288
- Google, iGoogle Home page widgets 19–20
- Google Maps
 - addOverlay function 97
 - API key 93
 - Capital Punishment Home widget 171
 - directional controls 94
 - geocodes 92
 - GMap2 function 93, 288
 - GPolyline function 294
 - information window 288
 - iPhone 323, 348
 - latitude 92
 - LiveWeatherMapWidget 284–289
 - load JavaScript client script file 93
 - LondonTubeMapWidget 289–298
 - longitude 92
 - map type control 94
 - markers 97, 288
 - obtaining API key 95
 - panDirection function 354
 - polylines 294
 - RecentSalesMashup, Zillow 92–99
 - setCenter function 94
 - zoom control 94
 - zoomIn function 354
 - zoomOut function 354
- GoogleGeocodeSampleService sample service 13

H

- HelloWorld sample service 13
- HelloWorldWidget sample widget 14
- Help widget, Facebook application 126
- Hewitt, Joe 313
- Home link, Facebook 113
- home page, zembly 20
- Home widget, Facebook 124
 - configure 124
- hover events
 - and CSS styles 60
- HTML
 - Facebook friend invite widget 191
 - Facebook widgets 121
 - page markup 35
 - radio buttons 167
 - select tag options array 294
 - vs FBML 120
- HTTP, call service 33

I

- iButtons widget 359–361
- iCapital Punishment widget 331–337
- icon, Facebook applications 128
- iframe 14
- iGoogle Home page 19–20
- iLeaves widget 356–359

- iLiveWeather widget 322–330
- iLoanPayment widget 314–322
- iLondonTube widget 348–356
- image, upload an 36
- img tag 36
 - src attribute 77
- Inbox tab, **zembly** 21
- Info section, Facebook 212
- Info tab, Facebook 114
- information window, Google Maps 288
- innerHTML 40
- instant publishing, Facebook 113
- invite friends, Facebook widget 187
- iPhone
 - overview 310–314
 - EDGE network 310
 - Facebook integration 337–348
 - Facebook session authorization 339
 - getSessionKey service 348
 - Google Maps 323
- iButtons widget 359–361
- iCapital Punishment widget 331–337
- icons to Home Screen 317
- iLeaves widget 356–359
- iLiveWeather widget 322–330
- iLoanPayment widget 314–322
- iLondonTube widget 348–356
- iPhoneHome widget 337–348
- iUI Library 313
 - multi-touch screen 311
 - numeric key pad 324
 - orientation changes 313
 - run widgets 315
 - screen 313
 - simulator 317
 - URL for Facebook application widget 339
 - URLs 316
 - virtual key pad 312
 - web applications 311
- iPhoneHome widget 337–348
- iPod Touch 311
- iUI Library, iPhone 313
 - fieldset tag 319

J

- JavaScript 35
 - differences with FBJS 158
 - arrays with FBJS 158
 - cookie helper functions 330
 - Facebook Client Library 233
 - formatting numbers 90
 - Number function 98
 - regular expressions 150
 - vs FBJS 120
- JavaScript Editor
 - code completion 131
 - Facebook API 131
- JavaScript Editor, **zembly** 31
- joehewitt.com 313
- JSON notation 52, 281
 - and portability 87, 276
 - Facebook adapters 156
 - Flickr API data 64
 - sample response from Yahoo! Pipes 305

- JSON parameter type for services 28

K

- Key parameter type for services 28
- keychain
 - Flickr API 68
- keychain, **zembly** 24–25, 46

L

- latitude, Google Maps 92
- left-hand column, Facebook 109
- Libraries
 - Prototype JS 37
 - Resources tab 37
- LiveWeatherBugWidget 276–279
 - add geocode data 285
 - sample JSON data 281
- LiveWeatherBugWidget 279–283
- LiveWeatherMapWidget 284–289
- LiveWeatherRSS adapter 47–50, 277
- Loan Calculator Facebook application 121–125
- LoanPaymentService 25–34
 - (JavaScript) 27
- LoanPaymentWidget 34–42
 - (JavaScript) 38
- lock badge, Facebook services and widgets 120
- Log.write 33
- logo, Facebook applications 129
- LondonTubeJourneyPlanner Dapp 250–252
- LondonTubeJourneyPlanner service 252–253
- LondonTubeMapWidget 289–298
 - stations_data.js file 293
- LondonTubeWidget 254–258
- longitude, Google Maps 92

M

- main profile page, Facebook 109
- Manage your Keychain, **zembly** 46
- map type control, Google Maps 94
- markers, Google Maps 97
- mashups
 - Capital Punishment Home widget 171
 - LiveWeatherMapWidget 284–289
 - LondonTubeMapWidget 289–298
 - RecentSalesMashup 92–99
- Math.random JavaScript function 75
- max-width style attribute 60
- McIlroy, Doug 298
- messages, **zembly** contacts 21
- mlbscores service 262
- mlbScoresWidget 263–265
- mlbupdate Dapp 259–262
- Mood Pix Facebook application 208–231
 - application Info section 219
 - application tabs 223
 - Info section 213
 - profile publisher 224
 - PublisherFriend service 227
 - PublisherSelf service 226
 - SeeMessages widget 231

SendMoodPix widget 227
 SetInfoOptions service 222
 Mood Pix Home Widget, (*JavaScript*) 217
 multi-touch screen, iPhone 311
 MyFlickrRandomSlideshow 71–77
 (*JavaScript*) 75

N

namespace, XML notation 277
 news feed story types, Facebook 112, 183
 news feed, Facebook 112
 notices, Facebook 112
 Number JavaScript function 98
 Number parameter type for services 28
 numeric key pad, iPhone input 324

O

object types, Facebook data store model 193
 one-line story type, Facebook 112, 183
 onFailure service call condition 66
 online/offline status, zembly 44
 onMouseout JavaScript event 65
 onMouseover JavaScript event 65
 opacity style attribute 62
 openInfoWindowHtml Google Maps function 288
 orientation changes, iPhone 313
 Owner.keychain, zembly 46

P

panDirection Google Maps function 354
 Parameter Editor, zembly 28, 70
 parameters
 add a new 70
 escape value 49
 extracting in a widget 75
 in widgets 72
 required 29
 services 27
 types for services (*table*) 28
 validation 30
 People tab, zembly 21, 23
 permissions, Facebook applications 108
 Pipes, *See* Yahoo! Pipes
 PNG image file, iPhone 318
 polyline, Google Maps feature 294
 Preview tab, zembly 41
 preview widget, Facebook 124
 preview widgets, zembly 17
 profile boxes 180
 main 109, 180
 profile boxes, Facebook 108
 profile parameter, setFBML adapter 157
 profile publisher, Facebook 113, 224
 profile, Facebook 105
 profile, zembly 21–22
 profile.getInfo Facebook adapter 221
 profile.setFBML Facebook adapter 157
 profile.setInfo Facebook adapter 221
 profile_main parameter, setFBML adapter 157
 properties, Facebook data store model 192

Prototype JS library 37
 \$("element_id") 41
 each function 63
 enumeration 63
 Event.observe function 64
 Publish action, Yahoo! Pipes 304
 publish, zembly 32, 41
 PublishChallenge service, Facebook application 190
 PublisherFriend service, Facebook 227
 PublisherSelf service, Facebook 226
 PublishScore service, Facebook 186

R

radio buttons, HTML 167
 JavaScript event handlers 173
 random number, generate in JavaScript 75
 real estate service, Zillow 80
 RecentSalesMashup 92–99
 (*JavaScript*) 97
 RecentSalesService 82–88
 (*JavaScript*) 86
 RecentSalesWidget 88–92
 (*JavaScript*) 90
 registered template bundle, Facebook application 189
 Registered Template Bundles Console, Facebook 182
 registered templates, Facebook 133
 RegisterTemplates service, Facebook application 184
 regular expressions, JavaScript 150
 required parameters, zembly 29
 requireLogin, FBJS 137, 206
 Resources tab, zembly 36
 Libraries 37
 responseType: Ajax.FBML, FBJS 143, 150
 responseType: Ajax.JSON, FBJS 217
 RSS feed 267

S

Safari, webkit 356
 sample data from LiveWeatherBugService 281
 samples, on zembly 13
 Save action, Yahoo! Pipes 304
 save current draft, zembly 31
 SaveScore service, Facebook data store model 198
 scores, zembly Things 15
 screen name, zembly 22
 screen, iPhone 313
 search for contacts, zembly 23
 secret key, Facebook 118
 SeeScores widget, Facebook 205
 select tag options array, JavaScript 294
 SendInvitation widget, Facebook 187
 SendMoodPix widget, Facebook 227
 service providers, zembly 24
 services
 call 32
 Call tab 30
 capture example 30
 creating 25

- drafts 42–45
- E4X notation 276
- Erase and start over (all drafts) 43
- Error Codes 29
- favorites 16
- Find & Use tab 49, 69
- FlickrPeopleService 68–71
- flickrPhotoSearch (Dapp-based) 243
- GambitsfromGailFeed 268
- getSessionKey, iPhone/Facebook integration 348
- LiveWeatherBugService 276–279
- LoanPaymentService 25–34
- LondonTubeJourneyPlanner 252–253
- mlbscores 262
- online/offline status 44
- parameter types (*table*) 28
- parameter validation 30
- parameters 27
- publish 32
- published versions 42
- RecentSalesService 82–88
- score 15
- search for using Find & Use tab 39
- test drive now 30
- timeline 44
- viewing versions 44
- watch action 16
- WeatherBugService 45–51
- Yahoo! Pipes 306
- session authorization, Facebook 136
- setCenter Google Maps function 94
- SetInfoOptions service, Facebook 222
- setInnerFBML FBJS function 143
- setInterval JavaScript function 62
- setStyle FBJS function 144
- setTimeout JavaScript function
 - with FBJS 147
- short story type, Facebook 112, 183
- slide show widget 57–67
 - BuddyPics Facebook application 137–158
- Sort operator, Yahoo! Pipes 303
- src attribute, `img` tag 77
- stations_data.js file, LondonTubeMapWidget 293
- story types, Facebook news feed 112
- String parameter type for services 28

T

- tags, Flickr photo search 63
- tags, zembly 16
- {*target*} token, Facebook feed story 183
- template bundles, Facebook 182
- template services and widgets, Facebook applications 126
- templates, widgets 35
- Terms of Service widget, Facebook application 126
- test services, zembly 30
- www.testiPhone.com simulator 317
- Things tab, zembly 20
- Thompson, Ken 298
- timeline, zembly 44
- timer code, JavaScript 170
- tokens, Facebook feed story 183
- touch events, iPhone 311

U

- Union operator, Yahoo! Pipes 301
- Unix pipe mechanism 298
- UpdateProfileBox service, Facebook applications 181
- UpdateProfileBox, Facebook applications 126
- upload an image, zembly 36
- URI parameter type for services 28
- URL
 - Facebook canvas page 117
 - Flickr photo source 64
 - iPhone widgets 316
- user, Facebook 105

V

- versions, zembly
 - published widgets and services 42
 - timeline 44
- virtual key pad, iPhone 312

W

- watch a service or widget, zembly 16
- WeatherBug API 45
 - sample XML data 276
- WeatherBugService 45–51
 - (JavaScript) 50
- WeatherBugWidget 51–54
 - (JavaScript) 52
- WeatherTodayService sample service 13
- web applications, iPhone 311
- webkit, Safari display engine 356
- What's happening tab, zembly 21, 24
- widgets
 - actions 16
 - adding to iGoogle Home page 19–20
 - clone 15
 - create 35
 - drafts 42–45
 - embed 14
 - Erase and start over (all drafts) 43
 - execute on iPhone 315
 - favorites 16
 - flickrPhotoSearchWidget 244–248
 - FlickrSlideShow 57–67
 - GambitsSummaryWidget 268–271
 - HTML and Facebook 121
 - iButtons 359–361
 - iCapital Punishment 331–337
 - iLeaves 356–359
 - iLiveWeather 322–330
 - iLoanPayment 314–322
 - iLondonTube 348–356
 - iPhone simulator 317
 - iPhoneHome 337–348
 - Libraries 37
 - LiveWeatherBugWidget 279–283
 - LiveWeatherMapWidget 284–289
 - LoanPaymentWidget 34–42
 - LondonTubeMapWidget 289–298
 - LondonTubeWidget 254–258
 - mlbScoresWidget 263–265

- MyFlickrRandomSlideshow 71–77
- online/offline status 44
- parameters 72
- preview 17
- Preview tab 41
- publish 41
- published versions 42
- RecentSalesMashup 92–99
- RecentSalesWidget 88–92
- Resources tab 37
- score 15
- sharing with ClearSpring 18, 66
- templates 35
- timeline 44
- viewing versions 44
- watch action 16
- WeatherBugWidget 51–54
- zemblyrumblings 304–307

X

- XFBML, Facebook 233
 - server tags 234
- (X)HTML, page markup 35
- XML data 48, 84, 276
 - namespace 277
 - node attributes 277
- XML parameter type for services 28

Y

- Yahoo! Pipes 298
 - Fetch Feed object 300
 - Filter operator 302
 - format of service call 306
 - JSON notation response 305
 - Publish action 304
 - Save action 304
 - Sort operator 303
 - Union operator 301
 - zemblyblog pipe 299–304
 - zemblyrumblings widget 304–307
- YahooTripSearchService sample service 13
- You tab, zembly 20
- YouTubeSampleService sample service 13

Z

- zembly
 - actions 16
 - adapters 24
 - Add to editor 40
 - API calls 24
 - Application object 134
 - Call tab 30
 - capture example 30
 - clone 15
 - code name 22
 - collaboration requests 21
 - contacts 22–24
 - create a service 25
 - create a widget 35
 - create Facebook applications 114–120

- drafts for services and widgets 42–45
- editor commands 31
- Erase and start over action 43
- favorites 16
- Find & Use tab 39, 49
- home page 20
- Inbox tab 21
- keychain 24–25, 46
- Manage your Keychain 46
- messages 21
- online/offline status 44
- Owner.keychain 46
- Parameter Editor 28
- parameter validation 30
- People tab 21, 23
- Preview tab 41
- preview widgets 17
- profile 21–22
- publish 32, 41
- Resources tab 36, 37
- save current draft 31
- scores 15
- screen name 22
- search for contacts 23
- service providers 24
- tags 16
- test services 30
- Things tab 20
- timeline 44
- upload resources 36
- watch a service or widget 16
- What's happening tab 21, 24
- You tab 20
- zemblyblog pipe 299–304
- zemblyConnectDemo Facebook application 232–235
- zemblyrumblings widget 304–307
- Zillow
 - RecentSalesMashup 92–99
 - RecentSalesService 82–88
 - RecentSalesWidget 88–92
 - sample XML data 84
 - zpid 82
- Zillow real estate service 80
- zillow.homevaluation.GetChart adapter 80
- zillow.homevaluation.GetComps adapter 80
- zillow.homevaluation.GetDemographics adapter 81
- zillow.homevaluation.GetRegionChart adapter 81
- zillow.homevaluation.GetRegionChildren adapter 81
- zillow.homevaluation.GetSearchResults adapter 81, 83
- zillow.homevaluation.GetZestimate adapter 81
- zillow.propertydetails.GetDeepComps adapter 81, 85
- zillow.propertydetails.GetDeepSearchResults adapter 81
- ZillowSampleService sample service 13
- zoom control, Google Maps 94
- zoomIn Google Maps function 354
- zoomOut Google Maps function 354
- zpid, Zillow property ID 82
- zventsSearchService sample service 14