Web Clipping Development and Poplet Kit 2.0

Introduction

The Poplet Kit 2.0 includes several Poplet modules that support many web clipping features. The major web clipping features supported are the following:

1. Requesting location information from the web driven from the address book.
2. Using selections in text fields in any application to initiate web search engine queries.
3. Implementing form validation before the form is submitted to the server, using device-local validation scripts. The scripts can be contained in hidden fields in the form or can be contained in a PQA.
4. Launching a PQA.
5. Launching Clipper to go to a URL (possibly with parameters attached) to get a web clipping.
6. Dynamically generating a PQA on the device to format data.

The Poplet Kit is useful both for development of finished web clipping applications and for prototyping web clipping functionality. Examples of useful prototyping are the following:

1. Exploring different PQA formatting options.
2. Exploring PQA form “get” parameters.
3. Prototyping a server application with an initial client-side implementation.

Location Based Services and the “Locate” Modules

The six LocateXXX modules provide a standard interface to location services offered by different web sites. They are shortcuts that dramatically reduce the number of taps to get to information. See the Poplet Kit Handbook documentation for descriptions of how to use the Locate modules.

These six Poplet modules are good models for creating additional location services. They all use the "get" method of form submission, appending the form values to the end of the URL string and submitting the form with:

```
URL.goto(url);
```

Look at the source code for the Locate modules to see how the URL strings are constructed. The key to extending this code to other web sites is understanding the arguments required by the target web site.

Search Engine Queries and the “Search” Modules

The Search modules are another kind of major shortcut. They allow you to search the web for selected text in any application containing a text field. For example, while reading email, an instant message or a memo, simply highlight

text of interest, pop up the Poplet menu and select SearchBritannica, SearchGoogle or SearchWebsters. These saves several steps: (1) there is no PQA launch step, (2) the user does not need to re-enter the word into a form, and (3) after reviewing the results, a tap of the Clipper back arrow causes a return to the original application.

These search modules are good models for supporting other search engines or corporate databases.

Local Validation of Forms

Local validation of forms is accomplished by using the Clipper Palmcall facility to invoke a validation function. The following is an example of form HTML with validation using a script in a hidden field.

```
<form action="palmcall:PPLT.appl?URL.script" method="get">
First: <input type="text" name="first" size=30><br>
Last: <input type="text" name="last" size=30><br>
<input type="hidden" name="script" value=
"if (this.first=='') ^'Enter first name';
 else if (this.last=='') ^'Enter last name';
 else URL.submit(this)">
<input type="hidden" name="url" value="http://www.somwhere.com">
<input type="submit" name="save" value="Save">
<input type="reset">
</form>
```

This form works as follows: There are two visible text fields with names “first” and “last”. There are two hidden fields with names “script” and “url”. The script field contains the validation script. The url field contains the server url to which the field values are appended to submit the form to the server. The form action parameter specifies a Palmcall to invoke the “script” function in Poplet module “URL”. Module URL is a “base module”, meaning that it is contained within the Poplet Kit prc database. The URL.script function is called with a single argument, the query part of the string, which looks like:

```
“first=joe&last=smith&script=if(this.first==’’) … &url=http: …”
```

The URL.script function “objectifies” the string so that the script refers to field names and values using the dot notation (e.g., `if (this.first==’’)`). At the end of the script, if the form is valid, there is a call to the URL.submit function. URL.submit converts the object back to a query string, appends it to the url and submits that string to the server. It also strips out the script and url name/value pairs from the final query string.

Dynamically Generated PQAs

The PQALoan Module

The PQALoan module calculates the same results as the Loan module. It requests for you to enter the principal amount, then generates payment

information for several different interest rates. Whereas Loan presents its results in the text field of a dialog box, PQALoan formats the results by dynamically generating a PQA, and then launches Clipper to display the generated PQA.

This module is a good example of object inheritance, it is a submodule of Loan. PQALoan contains a single function formatResults, which re-implements the function of the same name in the Loan module to generate a PQA.

The PQAMultiPage Module
The PQAMultiPage module dynamically generates a multi-page PQA. It includes examples of typical HTML formatting, such as headers, lists, tables, bold, underline and links different pages within the PQA.

The PQAMyCache Module
The PQAMyCache module is an example of device-resident processing of HTML forms. The application keeps a database named “MyCache” containing text records, each identified by a unique key. The PQA form allows (1) entry of new records, (2) record lookup by key, and (3) browsing records backwards and forwards in key order.

The Clip Module

Overview
The Clip module is the PQA generator and launcher. It is intended to be called from other Poplet modules, rather than to be launched from the Poplet menu. The goals of the Clip module are as follows:

1. Use the power of HTML through Clipper to format data on the device.
2. Have PQA generating applications be readable easily by anyone who understands HTML and Javascript.
3. Be concise and easy to use.

An HTML page is described using the Clip module by making a series of function calls corresponding to HTML tags and passing content as arguments. A simple page can be generated as follows:

```
Clip.pqaNew(“Example”);          //Creates database Example.pqa
url = “example.html”;

//Generate a single page – the following three lines produce html
title = “The title of our simple example”;
content = “Here is a very simple page”;
Clip.page(url, title, content);  //Output page of content

Clip.pqaClose();                 //Close database
Clip.pqaLaunch(“Example”);       //Launch Clipper to display pqa
```

The middle 3 lines of code above corresponds to the following HTML:

```
<html>
<head><title>The title of our simple example></title></head>
<body>Here is a very simple page</body>
</html>
```

Nested Content
HTML is a language where there are start and end tags bracketing content. For example:

```
<b>This is bold text</b>
```

And there can be nested content – tags within tags:

```
some text <b>some bold text, partially <u>with an underline</u></b>
```

This is mirrored in the Clip module using nested arrays. The HTML above is represented as follows:

```
underlined = Clip.u("with an underline");
bolded = Clip.b({"some bold text, partially ", underlined});
content = {"some text ", bolded};
```

In general, an HTML tag pair is represented by a Clip function call. The argument to a call is content. The content can be a string or an array. If the content is a string, the characters of the string are output as is. If the content is an array, each of the array elements contains content (i.e., a string or an array). The actual tags are represented by numbers in the content array.

Clip Functions Corresponding to HTML Tags
The following are currently defined Clip functions corresponding to HTML tags:

1. a(text)
   Outputs an anchor for text, e.g., <a name=text>.

2. b(content)
   Outputs content in , brackets.

3. form(url, content)
   Outputs content in <form>, </form> brackets with url as the form action.

4. h1(content)
   Outputs content in <h1>, </h1> brackets.

5. h2(content)
   Outputs content in <h2>, </h2> brackets.

6. h3(content)
   Outputs content in <h3>, </h3> brackets.

7. hr()
   Output <hr> tag, with no options specified.

8. i(content)
   Outputs content in , brackets.

9. li(content)
   Outputs <li> tag followed by content.

10. link(page, frag, content)
    Outputs a local (within PQA) link, of form content. Note that for a local link page is a page number. The first page is 0. Pages are numbered in the order they are created. If there is no url fragment, frag==””.

11. linkx(url, frag, content)
    Outputs an externall (outside of PQA) link, of form content. Note that url can identify a web based page, another device-based PQA, or a program (palm: and palmcall:). See the Palm Web Clipping Guide for the url syntax. If there is no url fragment, frag==””.

12. ol(content)
    Outputs content in <ol>, brackets.

13. submit(name, value)
    Outputs a form submit button with name==name and label==value.

14. table(content)
    Outputs content in <table>, </table> brackets.

15. td(content)
    Outputs content in <td>, </td> brackets.

16. textArea(rows, cols, name, value)
    Outputs a form text area, with height==rows, width==cols, name==name and initial value==value.

17. textLine(size, max, name, value)
    Outputs a form text line, maximum characters==max, name==name and initial value==value.

18. th(content)

Outputs content in <th>, </th> brackets.

19. tr(content)
    Outputs content in <tr>, </tr> brackets.

20. u(content)
    Outputs content in <u>, </u> brackets.

21. ul(content)
    Outputs content in <ul>, brackets.

Using Clip
Study the sample modules to see how they use Clip to generate a PQA.

Clip does not do error checking to verify that your calls are properly structured. For example, <tr> is only valid within <table> tags. If you make this kind of structure mistake, Clipper will not produce the results you want.

The best way to develop with Clip is incrementally. Start with skeletal structures, then flesh them out. Build a little and test a little. That way, errors are related to your most recent changes.

Extending Clip
The Clip module does not support every tag or every tag option supported by Clipper. However, since every tag simply returns a content array to output, it is relatively easy for you to add any missing features you would like to use. In order to extend Clip, first read the Palm File Format Specification which defines the PQA encoding format. Clip uses the uncompressed format.

The Clip module is in the base, so its source cannot be changed. The best way to modify or extend it is to create a submodule containing the changed behavior. For example, the Clip implementation of <hr> has no arguments. It uses the default Clipper implementation of horizontal rule. If you want an <hr> where you specify the width as a percentage of the browser window, define an hr function in another module (for example, Clip1) with a percentage argument. Then your new Clip1.hr function simply returns an array for the <hr> tag as specified in the PQA encoding format.