October 2004

BlackBerry Developer Journal

main()

"The computer will become the hub of a vast network of remote data stations and information banks feeding into the machine at a transmission rate of a billion or more bits of information a second. Laser channels will vastly increase both data capacity and the speeds with which it will be transmitted. Eventually, a global communications network handling voice, data and facsimile will instantly link man to machine--or machine to machine--by land, air, underwater, and space circuits. [The computer] will affect man's ways of thinking, his means of education, his relationship to his physical and social environment, and it will alter his ways of living... These [forces] will coalesce into what unquestionably will become the greatest adventure of the human mind."

David Sarnoff, President of RCA, 1964

David Sarnoff was a pioneer in broadcast radio and television. He was also responsible for laying the groundwork for most of the technology we use today. His forecast in 1964 hit close to the bone because he was bright, had a rich understanding of the industry, and could witness trends as they were developing.

David Sarnoff's projections have largely become reality, with future trends open for all to see. Processors will continue to get faster, memory and storage will get larger and prices will fall by the megabyte or megahertz. Operating systems and software will improve and wired and wireless network speeds will increase substantially. Use of technology will spread throughout the world in ways that we cannot envision today and everything we're using right now will be considered antiquated within a relatively short period of time.

BlackBerry™ technology will also continue to evolve. The handheld will get faster, more powerful and more full featured. Features and strength of the network products will improve, network speed will increase, costs will drop, and use will spread throughout the world in countless unseen ways.

The BlackBerry Developer Journal will have a front row seat as BlackBerry evolves. Each issue we'll focus on bringing information to light that will help you to take part in this evolutionary process. In this issue, and previous issues, we have laid the foundation for future articles by providing detailed information on supported languages. In future issues we will move forward to provide working examples of how to take full advantage of everything cool about BlackBerry. It should be fun!

As always we are interested in your comments, suggestions, letters, and anything else that you feel would be of benefit to our developer community.

Please feel free to email us at **editor@blackberrydeveloper.com**.

BlackBerry Developer Journal team

Java Bits & Pieces

Final Strings

Non-*final static Strings* are faster to access than *final static Strings*. This is a bit counter-intuitive and does not apply to any other data type (in fact the opposite is true for *int*, *long,* etc).

```
static final string myString = "foo"
```

... is like a C++

```
#define myString "foo"
```

Literal strings are confined using a hashtable so there will only be one copy of "foo" in memory. For a non-*final*, the VM does the hashtable lookup only once when it initializes `myString`. For a *final* string, the compiler eliminates `myString` and replaces it with "foo" in the bytecode, so the VM has to do a hashtable lookup each time you reference `myString`.

The one advantage of the *static final* is that it saves 4 bytes since no storage is allocated for `myString`.

You also need to take into account the life of the process when using just the *static*. In the case of the browser, for example, if a lot of the *Strings* in the browser were *static*; and not *final static*, the memory would just be hanging around the handheld even if the browser was never really used because the browser is always running.

To summarize, when speed is important use:

```
static String FOO = "foo";
```

When memory efficiency is important, use:

```
static final String BAR = "bar";
```

Also, if a *String* constant is non-*final*, a comment should be provided to indicate that the *String* is non-*final* for performance reasons and a statement of the performance reason.

Inner classes

Whenever possible, *inner classes* should be declared as *static*. The only time you should use a non-*static inner class* is when you need access to the *outer class's* instance data from within methods of the *inner class*. If you are just using an *inner class* for name scoping, make it *static*.

There are costs involved in using a non-*static inner class*. They contain an implicit pointer to the *outer class's this*, which costs in terms of object size and code generated for constructor calls, and outer class references.

The following code:

```
class outer {
  int i;
  class inner {
    inner() {}
    int foo() { return i; }
  }
}
```

... is really just a short form for this:

```
class outer {
  int i;
}

class outer$inner {
  private outer outer$this;
  outer$inner( outer o ) { outer$this = o; }
  int foo() { return outer$this.i; }
}
```

However, this code:

```
class outer {
  static class inner {
  }
}
```

... really just scopes the name of the *inner class,* like this:

```
class outer {
}

class outer$inner {
}
```

Inner classes should also avoid being *private*. Use of the *private* access specifier means that javac has to generate extra code to access the default constructor. Use the default *package* access, or *public*, if that is appropriate.

Stack versus Member object accesses

When accessing a data member three or more times, it is cheaper to cache the field value in a local variable.

Conditionals are boolean

Instead of this:

```
if( something_that_evaluates_to_true ) {
  return true;
} else {
  return false;
}
```

... use:

```
return( something_that_evaulates_to_true );
```

In addition to being easier to read, the resultant class file is 6 bytes shorter.

Saving objects for large tree-like data structures

For large tree-structured data structures like hierarchies or folder structures, object ordinal usage can be reduced by the following idiom:

```
class Tree {
  Tree left;
  Tree right;
  int value;
};
```

Convert 1000 objects with three members to three arrays:

```
short left[1000];
short right[1000];
int value[1000];
```

... and use integer values to refer to tree nodes within the array.

Remove unused code

During development, there are often several attempts at creating a software solution to a problem. As the code transforms from the first cut to the final result, leftover pieces of code accumulate. These are classes and methods that are never used. Neither javac nor RIM's compiler removes these unused pieces of code for you. Once a body of code is in working form, it should be inspected for these leftover pieces and they should be removed manually.

This cleanup reduces code size on the handheld and makes the code easier to understand and maintain.

Use static final boolean for debug code

When tracking down a bug, it is often necessary to instrument the code. After the bug has been found and fixed, there is still the extra debug code. Simple *println()* calls should most likely be removed, or at least commented out. Sometimes finding the bug required adding some more significant code. Often this code took some effort and would come in handy when another bug appears. Thus it is reasonable to leave the debug methods in the code, so that they don't have to be recreated the next time. On the other hand, we don't want to pay the price of the code size when we aren't debugging. Javac supports the following idiom:

```
class fubar {
  public static final boolean DEBUG = false;

  private figureOutBug() {
    if( DEBUG ) {
      // do something interesting
    }
  }

  public usefulMethod() {
    ...
    if( DEBUG )
      figureOutBug();
    ...
  }
}
```

When this code is compiled, and DEBUG is false, the portions that can never be executed are omitted by javac. Furthermore, when the compiler detects that the private figureOutBug method is never called and has no code, it will remove the method entirely.

If another bug appears, the static final DEBUG boolean can be changed to true and all the debug code will be reactivated.

Use StringBuffer wisely

Javac tries to be helpful. It supports the '+' operator for *Strings*. This allows for code like:

```
int left = 1;
int right = 2;
int value = left + right;
System.out.println( "When adding " + left + " to
" + right + " the result is: " + value );
```

What is often not apparent is that the above expression produces this code:

```
StringBuffer temp = new StringBuffer();
temp.append( "When adding " );
temp.append( left );
temp.append( " to " );
temp.append( right );
temp.append( " the result is: " );
temp.append( value );
System.out.println( temp.toString() );
```

The first form makes the code easier to read but obscures the cost associated with the code.

Note that the one time when javac does make it cheaper is this:

```
System.out.println( "First line of message\n" +
"Second line of message" );
```

This produces this code:

```
System.out.println( "First line of
message\nSecond line of message");
```

So, when concatenating *Strings*, think about whether it is necessary and then look at the arguments.

Creating garbage

String concatenation using the '+' operator is particularly good at creating stealth garbage. Under the covers the compiler generates code to create a new *StringBuffer* object, append some strings into it and then creates a new String object using *StringBuffer.toString()*. An innocent looking line like *String* s = s1 + s2 + s3 could create half a dozen bits of garbage. We suggest using the StringBuffer directly and then set the object to null when you're done with it.

Use switch instead of cascading if/else

When checking a variable of primitive type against a set of values, use *switch()* instead of a series of *if/else* statements.

This code:

```
if( x == 1 ) {
  // do something
} else if( x == 2 ) {
  // do something else
} else if( x == 3 ) {
  // do another thing
} else {
  // do the last thing
}
```

Should be transformed to:

```
switch( x ) {
  case 1:
    // do something
    break;
  case 2:
    // do something else
    break;
  case 3:
    // do another thing
    break;
  default:
    // do the last thing
    break;
}
```

This lets the VM do the series of comparisons and reduces the number of times that the value of 'x' must be fetched.

The flip side is that whenever possible, use a primitive type for a range of values, rather than using a *String*, or some other reference type.

Use 'char' for single character Strings

It is not recommended to create a *String* like "a". Use 'a' instead.

For example:

```
StringBuffer tmp = new StringBuffer();
tmp.append( left );
tmp.append( "+" );
tmp.append( right );
tmp.append( "=" );
tmp.append( left+right );
System.out.println( tmp.toString() );
```

... should be coded as:

```
StringBuffer tmp = new StringBuffer();
tmp.append( left );
tmp.append( '+' );
tmp.append( right );
tmp.append( '=' );
tmp.append( left+right );
System.out.println( tmp.toString() );
```

This saves the time required to instantiate two extra *Strings* which immediately become garbage and have to be collected.

Know your libraries

Whenever you are about to write some code to do a primitive operation on a common data type, check first to see if that code already exists in a utility class.

Initialization code should be optimized for size

Since initialization code is only meant to be run once, it should favor size over speed. If there are two ways to write the init code, one that is smaller and one that is faster, pick the smaller one.

Exception code should be optimized for size

Since exception code is meant to rarely, if ever, be executed, ensure that it is as small as possible. Make sure that exception messages aren't any longer than necessary. Often the stack traceback is all that is really needed.

When defining exception classes, consider whether a hierarchy is necessary. If the calling code isn't going to distinguish which exception was thrown, only define one type. This reduces the code size cost associated with multiple exception types.

C++ Bits & Pieces

Using __declspec(dllexport) and __declspec(dllimport)

If method_a() and method_b() are located in a .dll (exporting dll) and you want to call these methods from another dll (importing dll), do the following:

In the exporting dll, try something like:

```
__declspec(dllexport) int method_a(void){...}
__declspec(dllexport) void method_b(int b){...}
```

... and in the importing dll:

```
__declspec(dllimport) int method_a(void);
__declspec(dllimport) void method_b(int b);

void PagerMain(void) {
   int a = method_a();
   method_b(a);
}
```

You can also use __declspec(dllexport) to define a macro to make your code more readable.

How to add Address Book entries

The Address Book API allows your application to access the BlackBerry handheld's current Address Book. The following program illustrates some of the more useful capabilities of the Address Book API, including how to add your name, phone number, and email address to the Address Book.

```
MESSAGE msg;
int i; // counter
char buffer[30]; // buffer for display data
const NUM_FIELDS = 3; // number of fields to
iterate through

//fields to iterate through
const AddressFieldType fieldType[NUM_FIELDS] = {
  NAME,
  EMAIL,
  PHONE
};

//address handle
AddressHandle ah = NULL;

//address field handle
AddressFieldHandle afh =
DB_INVALID_FIELD_HANDLE;

//information to retrieve
char *fieldname;
AddressFieldType aftype;

// returns number of Johnson AND points "ah"
// to the first address
i = get_address_match_count("Johnson", &ah);

// iterate through fields
for (i = 0; i < NUM_FIELDS; i++) {
  // Gets a handle to a field in an address
  afh = get_address_field_handle
    (ah, fieldType[i], NULL);

  //Get the field's type
  aftype = get_address_field_type(ah, afh);

  //Get the fields name
  fieldname = (char *)
    get_field_type_name(aftype);

  //Get the fields data pointer and length
  int len;

  const byte* data = get_address_field_data
    (ah, afh, &len);

  //Display information
  if (data) {
    RimSprintf(buffer, sizeof(buffer),
      "%s:  %s", fieldname, data);
    LcdPutStringXY(0,i*10 + 15,buffer,-1,0);
  }
} // addressfield loop

// Wait for click
for (;;) {
  RimGetMessage(&msg);

  if (msg.Event == THUMB_CLICK) {
    break;
  }
} // thumbclick loop
```

Java Low Memory Manager - A Development Guide

Mike Kirkup, Research In Motion

The *Low Memory Manager* (*LMM*) is used to maintain memory resources on the BlackBerry handheld when those resources pass a low-level threshold indicating that memory resources are in scarce supply. The *LMM* will attempt to free up existing memory in an effort to provide more memory space on the handheld. All applications, including third party solutions, should work with the *LMM* to clear up as much space as possible when the handheld is running low on memory resources.

Low Memory Manager Conditions

It is important to understand that there are three conditions that can cause the *LMM* to try to free up memory resources.

1. The amount of available flash memory on the handheld decreases below a certain threshold. The free flash threshold is actually dependent on the amount of free RAM in the system. Generally, the free flash threshold varies between 400 KB and 800 KB with a guarantee to invoke the *LMM* if free flash drops below 400KB.
2. The number of object handles available on the handheld decreases below 1,000 handles. The number of available handles depends on the amount of flash on the handheld. For 8MB handhelds, there are approximately 12,000 available handles. For 16 MB handhelds, there are approximately 27,000 available handles.
3. The number of reference ordinals available on the handheld decreases below a certain threshold. On current handhelds this threshold is set to 1,000 reference ordinals. The number of available reference ordinals depends on the amount of flash on the handheld. For 8MB handhelds, there are approximately 24,000 available reference ordinals. For 16MB handhelds, there are approximately 56,000 available reference ordinals.

Working with the Low Memory Manager

There are two stages for applications to take advantage of the *LMM*. We will discuss these two stages in detail.

1. Registering your application as a *LowMemoryListener.*
2. Handling events received by the *LowMemoryListener.*

Stage # 1 - Registering your application as a LowMemoryListener

In order to use the *LMM*, you need to register your application with the *LMM*. The only way to do that is to provide an implementation of the *LowMemoryListener* interface in your application. When your application is started for the first time, you want to register the listener implementation with the *LMM*. Note that you only want to register your listener once. It is very important not to register more than once because you would receive multiple calls when the *LMM* is invoked.

The *LowMemoryListener* has the following method:

```
public boolean freeStaleObject( int priority )
```

This method will be automatically invoked when the *LMM* recognizes that the system is running low on memory, or when it has been invoked directly by a call to the poll method in the *LowMemoryManager* class.

In order to implement the *freeStaleObject()* method, you need to first understand the concept of where priority comes into this discussion. There are three levels of priority: low, medium and high. In each case, the operation of the application could differ in terms of what memory resources it gives up.

For example, when *freeStaleObject()* is invoked with low priority, the application should consider clearing up some transitory variables and anything that is currently not necessary for complete functionality such as cached data. No extraordinary efforts should be made.

When invoked with medium priority, the application should consider cleaning up stale data such as very old emails or old calendar entries.

When invoked with high priority, the application should clean out objects in the application on a *Least Recently Used* (*LRU*) policy. For example, when the *LMM* is invoked on the Email application it will start to delete messages at the bottom of the list of emails since those are likely to be the least used. The application should remove all the stale objects it has in order to reduce the amount of memory consumed on the handheld.

The *freeStaleObject()* method returns a boolean which is used to indicate if persistent data was released during the call to *freeStaleObject()*. It returns true if persistent data was released.

Implementation Details

Now that we have discussed the different approaches to freeing objects and how that relates to the priority, it is important to discuss how an application should go about freeing up objects. The *LMM* provides one method that enables this exact procedure.

The application should first remove all references to an object and delete it from its data structures. On completion, the application should call *LowMemoryManager.markAsRecoverable()*, passing in the

object that should be freed. This command will indicate to the underlying *JVM* that this object can be removed as part of the *LMM* operation.

It is important to call the *markAsRecoverable()* method because the *LMM* has a specific amount of memory it is trying to recover. It will spread this amount over all of the registered applications. If your application does free up memory resources it is important to notify the *LMM* of this fact so that it can count that freed memory against its target amount.

The following is a sample implementation of the *freeStaleObject()* method that frees up items from different persisted vectors associated with each of the priority levels.

```
/**
* The implementation of the freeStaleObject method required by the
* LowMemoryListener.  This method is invoked when the LMM is running out of
* memory related resources.  The application is asked to free up resources according
* to the priority level passed into this method.
* @param priority the priority of the call which is either High, Medium or Low.
* @return a boolean indicating whether or not memory was freed by this call.
* It is VERY important that the proper value is returned from this method.
*/
public boolean freeStaleObject( int priority )
{
  boolean dataFreed = false;

  switch( priority ) {
    case LowMemoryListener.HIGH_PRIORITY:
      dataFreed = freeVector( _data._high );
      _priority = LowMemoryListener.LOW_PRIORITY;
      break;
    case LowMemoryListener.MEDIUM_PRIORITY:
      dataFreed = freeVector( _data._medium );
      _priority = LowMemoryListener.HIGH_PRIORITY;
      break;
    case LowMemoryListener.LOW_PRIORITY:
      dataFreed = freeVector( _data._low );
      _priority = LowMemoryListener.MEDIUM_PRIORITY;
      break;
  }

  if( dataFreed ) {
    _persist.commit();
  }
  return dataFreed;
}

/**
* A private method that will free up the priority vector.
* @param vector the vector to free up.
* @return a boolean indicating whether any objects were freed by this method.
*/
private boolean freeVector( Vector vector )
{
  boolean dataFreed = false;
  int size = vector.size();

  for( int i = size - 1; i >= 0; i-- ) {
    Object obj = vector.elementAt( i );
    vector.removeElementAt( i );
    LowMemoryManager.markAsRecoverable( obj );
    dataFreed = true;
  }

  return dataFreed;
}
```

WML 101

Richard Evers, Editor

The Wireless Markup Language (WML) is an easy to learn and lightweight language designed to develop Wireless Application Protocol (WAP) applications for lightweight devices. It has become a standard that is supported by most wireless browsers. This article focuses on WML development and covers most features of the language in a style that will encourage rapid learning.

The Basics

A WML file (or page) is called a deck, and modules within the deck are called cards. Cards can contain 'href' links to other cards, URLS, telephone numbers, email addresses and WMLScript functions that are within external files.

As with XML, all WML tags and attributes must be provided in lower-case form.

All WML scripts begin with this mandatory WML header:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
```

... and are wrapped between the start WML tag:

```
<wml>
```

... and the end WML tag:

```
</wml>
```

Note: *all code snippets to follow must be wrapped between WML tags and include the WML header to work.*

The following illustrates the basic construct of a WML page:

```
<card id="Hello">
  <p>
    &quot;Hello, ... <br/>
    ... World!&quot;
  </p>
</card>
```

Metaphorically, a WML page is a 'deck' of 'cards'. In this case, there is one card in the deck.

In the code above:

- The card is given an identifier of “Hello”
- A new aragraph is started
- The phrase “Hello, ...” is displayed on one line, prefixed with a double-quote (") and terminated with a self-terminating line feed (<br />). Note that " and <br /> are included to demonstrate use.
- The phrase “... World!” is displayed on the next line, suffixed with a double-quote.
- The next two lines close off the aragraph and </card>.

HTML programmers will find it fairly easy to develop WML pages. The key things to remember are that WML requires use of lower-case tags and attributes, and each opening tag must have a matching closing tag.

```
<card id="CoolQuote">
  <p>
    &quot;In the extraordinary transformation of
    heat to light, Gibbon rested...&quot;
  </p>

  <p>
    <br />
    <i>From the introduction to Edward Gibbon's
&quot;Decline and Fall of the Roman
Empire&quot;</i>
  </p>
</card>
```

Exploring Specific Features of WML

Aligning Paragraphs

Paragraphs can be left, right and center aligned as follows:

```
<p align="left"></p>
<p align="right"></p>
<p align="center"></p>
```

The example below shows how to right-align monetary values. Note that '$' is a reserved character to identify variables. By escaping through '$$', a single '$' will be displayed.

```
<card id="Alignment" title="Right">
  <p align="right">
    DVD $$19.95<br />
    CD $$9.95<br />
    Player $$119.95<br />
  </p>
</card>
```

Do it with Style

The following style tags will work on the BlackBerry Browser:

Style & Tag	Example
Bold ****	So ****Bold**** and Beautiful
Emphasis ****	So Bold ****and**** Beautiful

Style & Tag	Example
Italic ****	So Bold and ****Beautiful****
Strong ****	****So**** Bold and Beautiful
Underline ** **	**<u>**So Bold and Beautiful**</u>**

Note: *The BlackBerry Browser does not always display spaces as you might expect when using styles. Multiple spaces are translated into single spaces, and spaces that trail closing style tags do not appear. To get around this difficulty, use non-breaking space entities ( ) rather than true spaces, or insert carriage returns after closing tags. Carriage returns are automatically translated into a space within displayed strings.*

Showing Pix

The BlackBerry Browser supports several image types:

Type	Description
WBMP	Monochrome Wireless Bitmaps, which are low-resolution black and white images. For more information, refer to the Wireless Application Environment Defined Media Type Specification (WAP-237-WAEMT-20010515-a) at http://www.wapforum.org.
GIF	GIF87 and GIF89 image formats. Note that the browser only displays the first frame of animated GIF files.
JPEG	Only color screen BlackBerry handhelds support JPEG images directly. With the BlackBerry Browser, the Mobile Data Service converts JPEG images for display on monochrome handhelds. With the WAP browser, the WAP gateway must convert JPEG images for display on monochrome handhelds.
PNG	By default, the browser converts GIF images to PNG images. PNG images have a higher compression ratio and they support alpha channels.

The standard browser operates through a WAP gateway responsible for conversion of images to a format acceptable to the handheld. When the gateway receives a request to transmit a JPEG image to a monochrome handheld, it either converts the image to an acceptable format or returns a 406 error response. The gateway may also impose size restrictions that prevent larger JPEG files from being retrieved.

BlackBerry handhelds that are configured to operate through the BlackBerry Enterprise Server™ and Mobile Data Service (MDS) can handle far greater types of content.

MDS relies on a User Agent Profile to determine screen size, color depth and accepted image and content types before processing images for display by the handheld.

For example, the RIM 7230 v3.7.1 User Agent Profile has the following characteristics listed under the "BrowserHardware" section:

BitsPerPixel	16
ColorCapable	Yes
ImageCapable	Yes
PixelAspectRatio	1x1
ScreenSize	240x160
ScreenSizeChar	26x10
StandardFontProportional	Yes

The BlackBerry Browser can appear to directly handle a wider range of content through server-side MDS transcoding based on full knowledge of device profiles.

Images are also automatically scaled to better fit the display. Images that are wider than the display are scaled to match the display width less 5 pixels. If the image height is greater than twice the screen height, the image is scaled to be twice the display height. Proportional scaling is always employed. Use the trackwheel to scroll vertical images, and ALT trackwheel to scroll horizontally.

The syntax for displaying images is as follows:

```
<card id="Pix" title="A Fine Photo">
  <p align="center">
    <img src="mypix.png" alt="My Photo" />
    <br />
    Such a lovely photo!
    <br />
  </p>
</card>
```

Note that the BlackBerry Browser does not support built-in images as some cell phones do. If your WML page makes use of built-in images, the "alt" description will be displayed when viewing with the BlackBerry Browser.

Tables

Tables are easy to create but need to have the number of columns defined before creation:

```
<card id="Table" title="Table">
  <p>
    <table columns="1">
      <tr><td>One</td></tr>
    </table>
  </p>
  <p>
    <table columns="2">
      <tr><td>One</td><td>Two</td></tr>
    </table>
  </p>
  <p>
    <table columns="3">
      <tr>
        <td>One</td>
        <td>Two</td>
        <td>Three</td>
      </tr>
    </table>
  </p>
</card>
```

Images can be displayed within cells, and WML style tags can be used within cells and across entire tables.

Entities

WML supports all named, decimal, and hexadecimal character entities (in Unicode) including:

Named	Decimal	Hex	Description
&quot;	&#34;	&#x22;	Quotation mark
&amp;	&#38;	&#x26;	Ampersand
&apos;	&#39;	&#x27;	Apostrophe
&lt;	&#60;	&#x3C;	Less than
&gt;	&#62;	&#x3E;	Greater than
			Non-breaking space
&shy;	&#173;	&#xAD;	Soft (discretionary) hyphen

Including Telephone Numbers

WML pages can include linkable telephone numbers and email addresses:

```
<card id="Contact" title="Contact: DisIsMe">
  <p>
    Call:
    <a href="wtai://wp/mc;5195551212"
      title="Call">Work</a>
    <a href="wtai://wp/mc;5195551213"
      title="Call">Home</a><br />
    Email
    <a href="mailto:work@my_office.com">Work</a>
    <a href="mailto:home@my_home.com">Home</a>
  </p>
</card>
```

Multiple Cards

The true power of WML is shown by including several cards within a deck.

```
<!-- Multi-card phone and email example -->
<card id="PhoneList" title="Main Listing">
  <p align="center">
    My Phone and Mail Directory
  </p>
  <p>
    <a href="#Funny">Comedians</a>
    <br />
    <a href="#Tunes">Musicians</a>
  </p>
</card>

<card id="Funny" title="Comedians">
  <p align="center">
    Comedians
  </p>
  <p>
    Cheech Marin
    <br/>
    <a href="wtai://wp/mc;5195551234">
      &gt;Call
    </a>
    <br />
    <a href="mailto:info@cheechandchong.com">
      &gt;Email
    </a>
    <a href="http://www.cheechandchong.com">
      &gt;Web Site
    </a>
  </p>

  <p>
    Tommy Chong
    <br/>
    <a href="wtai://wp/mc;5195551234">
      &gt;Call</a><br />
    <a href="mailto:info@cheechandchong.com">
      &gt;Email</a>
    <a href="http://www.cheechandchong.com">
      &gt;Web Site</a>
    <do type="prev" label="Back">
      <prev/>
    </do>
  </p>
</card>

<card id="Tunes" title="Musicians">
  <p align="center">
    Musicians
  </p>
  <p>
    Alanis Morissette
    <br/>
    <a href="wtai://wp/mc;5195551234">
      &gt;Call
    </a><br />
```

```
    <a href="mailto:info@alanismorissette.com">
      &gt;Email
    </a>
    <a href="http://www.alanismorissette.com">
      &gt;Web Site
    </a>
  </p>
  <p>
    Barenaked Ladies
    <br/>
    <a href="wtai://wp/mc;5195551234x123">
      &gt;Call
    </a><br />
    <a href="mailto:info@bnlmusic.com">
      &gt;Email
    </a>
    <a href="http://www.bnlmusic.com">
      &gt;Web Site
    </a>
    <a href="ladies.wml#Ed">
      &gt;Ed's Edvice
    </a>
    <a href="ladies.wml#Steve">
      &gt;Steve's Sagacity
    </a>
    <do type="prev" label="Back">
      <prev/>
    </do>
  </p>
</card>
```

Unique things about this sample:

- A comment line was included at the start.
- The first card, “PhoneList”, displays a menu to access other cards within the script via:
  ```
  <a href="#Funny">Comedians</a>
  <br />
  <a href="#Tunes">Musicians</a>
  ```
- The remaining cards, “Funny” and “Tunes”, display the syntax used to incorporate telephone numbers, email addresses and URLs in user-selectable HREF links.
- The telephone number in the “Barenaked Ladies” card also includes a telephone extension (x123).
- Two trailing href's in the “Barenaked Ladies” card call an external WML file, positioning to cards with an id of “Ed” or “Steve”.
- A line near the base of the “Funny” and “Tunes” cards:
  `<do type="prev" label="Back"><prev/></do>`
  ... provides users with a method to return to the previous screen.

Note: *Many URLS shown within this article do not exist and have been used purely to demonstrate syntax and use.*

Variables

Variables can be used in WML pages by using the <setvar> and <input> tags, and can be accessed by preceding the variable name with '$'. The following example show how to use <setvar> and <input> to set variables, and how to display them thereafter.

```
<!-- An example that uses variables -->
<card id="Variables" title="Wonderful
Variables">
  <p align="center">
    Working with variables
  </p>
  <do type="accept" label="Want to see">
    <go href="#WhatIs">
      <setvar name="mListen"
        value="Jimi Hendrix" />
      <setvar name="mAge" value="44.93" />
      <p>
        You like to listen to?
        <input  name="yListen" />
        What is your telephone number?
        <input  name="yTelephone"
          format="\(NNN\)\ NNN\-NNNN" />
        Your age?
        <input  name="yAge" size="2"
          maxlength="2" format="NN" />
        Your password?
        <input  name="yPassword" size="12"
          maxlength="12" type="password"
          format="*x" />
      </p>
    </go>
  </do>
</card>

<!-- display variables -->
<card id="WhatIs" title="Here they are">
  <p align="center">
    My wonderful variables!
  </p>
  <p>
    I am $mAge years old and like to listen to
    $mListen<br />
    You are $yAge years old and like to listen to
    $yListen<br />
    Your telephone number is $yTelephone<br />
    Your "secret" password is $yPassword<br />
  </p>
</card>
```

Unique things about this sample:

- The opening "Variables" card declares two variables using <setvar>, then prompts the user to enter three additional variables through the use of the <input> tag.
- The first <input> tag does not perform any field limitation, validation or conversion. It simply lays down an input field that is the width of the current window.

- The second <input> tag masks the telephone entry field to automatically insert characters while the user is typing. The format="\(NNN\)\ NNN\-NNNN" will translate into "(999) 999-9999" if the user typed 9's.
- The third <input> tag limits the size of the field to 2 characters, and limits entry to numeric values. The BlackBerry Browser will automatically use numeric values when numeric keys are entered even if the Alt key is not depressed.
- The fourth <input> tag limits the field size to 12 characters, obfuscates input by setting type="password", and "wildcard" formats the data to allow alphanumeric and symbol characters converted to lower case.
- The final "WhatIs" card display all variables.

Note that variable names are case sensitive, must begin with an alphabetic or an underscore character, and can contain numeric characters after the first character.

The Input Element

Attribute	Required	Type	Description
name	Yes	Alphanumeric	The name of the variable used for the input field
type	No	Alphabetic	"text" (default) or "password"
value	No	Alphanumeric	Default value of the "name" variable
format	No	Alphanumeric	Sets a mask to allow, limit and alter data entered by the user
		Format	**Description**
		A	Allow upper-case alphabetic, punctuation and symbols
		a	Allow lower-case alphabetic, punctuation and symbols
		N	Allow numeric characters
		n	Allow numeric, punctuation and symbols
		X	Allow upper-case alphabetic, numeric, punctuation and symbols
		x	Allow lower-case alphabetic, numeric, punctuation and symbols
		M	Allow all supported characters
		m	Allow all supported characters
		*F	A wildcard mask, which applies the second character mask ('F', which represents a Format from the list above) to the entire input string from the placement position onwards. Note that the mask ('F') must be the last character in the masks
		#F	A repeating mask where '#' represent a value from 1 to 9, and 'F' represents a Format from the list above (except *F). Note that this must be positioned as the last entry in the mask.
		\C	Display the next escaped character 'C' in the input field.
emptyok	No	Boolean	If true, then the user does not have to enter a value in the field
size	No	Numeric	The size of the input field
maxlength	No	Numeric	The maximum number of characters that can be typed
tabindex	No	Numeric	Sets the tab order between fields. Ignored by the BlackBerry Browser.
title	No	Alphanumeric	Sets the field title. Ignored by the BlackBerry Browser.
accesskey	No	Numeric	Sets a key value to access the field. Ignored by the BlackBerry Browser.

The example to follow shows how to navigate to a fully formed user-entered URL by using field formatting, then using a "noesc" directive to stop the parser from translating the URL before use.

```
<!-- An example that navigates to a site -->
<!-- using a variable -->
<card id="Navigator"
  title="Variable
  Navigation">
  <p align="center">
    Let's set an URL
  </p>

  <input name="Troll"
    value="http://www.blackberry.net/go/mobile/"
    format="\h\t\t\p\:\/\/*x" />

  <do type="accept" label="Want to browse?">
    <go href="$(Troll:noesc)" />
  </do>
</card>
```

Other variable directives include “escape” to translate non-alphanumeric characters into their hexadecimal equivalents, and “unesc” to reverse the translation from hex to original form.

Give me an option

The best user interface for WAP forms is usually one that requires the least amount of typing and navigating from one screen to another. To minimize keyboard activity, use menus of options where the user can scroll through a list of options, and select one or more options from the list.

The first method in the “MyOptions” group allows the user to select a single option from the list. The selected option string is returned in $MyOptions, with the index number of the selected option (range 0-n) returned in $MyOptionsIndex where a return of zero indicates that no option was selected.

The second method in the “MoreOptions” group allows the user to select zero or more options from the list. The selected option strings are returned in $MoreOptions.

Note that none of the attributes available for use with the select tag are mandatory. A single <select> will suffice to start an options list.

```
<!-- An example that uses a list of options -->
<card id="Options" title="Our Options">
  <p align="center">
    Let's display our options
  </p>

  <p>
    <select name="MyOptions"
      iname="MyOptionsIndex">
      <option value="Life">
        Living well
      </option>
      <option value="Death">
        Not living at all
      </option>
      <option value="Tween">
        Something in between
      </option>
    </select>

    <select name="MoreOptions" multiple="true">
      <option value="Happy">
        Feeling fine
      </option>
      <option value="Sad">
        Feeling awful
      </option>
      <option value="Normal">
        Getting by
      </option>
    </select>

    <do type="accept" label="Want to see">
      <go href="#TheOption" />
    </do>
  </p>
</card>

<card id="TheOption" title="Selected Option">
  <p align="center">
    You selected $MyOptions which was option
    #$MyOptionsIndex<br />
    You also selected $MoreOptions
  </p>
</card>
```

The Select Element

Attribute	Required	Type	Description
title	No	Alphanumeric	Sets the set title. Ignored by the BlackBerry Browser.
name	No	Alphanumeric	Returns string value of selected option(s). The first letter in the variable name must be alphabetic.
iname	No	Numeric	Name token. Returns numeric value of selected option. The first letter in the variable name must be alphabetic.
ivalue	No	Numeric	Index value of default <option> when the script is first run.
multiple	No	Boolean	Set to "true" to permit use to select multiple <option> items. The default is "false"

Attribute	Required	Type	Description
tabindex	No	Numeric	Sets the tab order between select groups. Ignored by the BlackBerry Browser.
xml:lang	No	Alphabetic	Language code. See: **http://lcweb.loc.gov/standards/iso639-2/englangn.html**

Event Handling

Events are triggered when cards are loaded, cards are acted upon, and timers are set.

For example, the following code will cause the second card to be displayed as soon as the first card is loaded due to the way events are triggered. It will then load the third card when the <prev /> event is used to navigate back to the first card. When a <go> is used in the third card to directly navigate to the first card, it will automatically load the second card again.

```
<card id="One" title="Card #1"
  onenterforward="#Two"
  onenterbackward="#Three">
  <p align="center">
    This is card #1
  </p>
</card>

<card id="Two" title="Card #2">
  <p align="center">
    This is card #2
  </p>

  <do type="accept" label="To Card #1">
    <prev />
  </do>
</card>

<card id="Three" title="Card #3">
  <p align="center">
    This is card #3
  </p>

  <do type="accept" label="To Card #1">
    <go href="#One" />
  </do>
</card>
```

This code works because of the way the “onenterforward” and “onenterbackward” events are triggered.

An “onenterforward” event is triggered when a card is first loaded, or when it is called via <go> or <anchor>.

An “onenterbackward” event is triggered when a card is called via <prev />.

Card one is destined never to display by setting it to handle both events.

As shown in the following code, timers are enabled in the card by setting the timer name and duration in the code of a card, and setting the action within the card declaration using the “ontimer” attribute.

```
<!-- An example that deals with timers -->
<card id="First" title="First card"
  ontimer="#Second">
  <p align="center">
    My first card
  </p>
  <timer name="t_1" value=20 />
</card>

<card id="Second" title="Second card"
  ontimer="#Third">
  <p align="center">
    My second card
  </p>
  <timer name="t_2" value=20 />
</card>

<card id="Third" title="Third card">
  <p align="center">
    My third card
  </p>
</card>
```

In this example, the “First” card displays for 2 seconds then loads the “Second” card. The “Second” card does the same then calls the “Third” card.

You will run into a problem if you call a card that already has an expired timer. For example, changing the card “Third” as shown below will not work as expected:

```
<card id="Third" title="Third card"
  ontimer="#First">
  <p align="center">
    My third card
  </p>
  <timer name="t_3" value=20 />
</card>
```

On initial execution of the script with the modified “Third” card above, cards “First”, “Second”, “Third” and “First” will be called in sequence every 2 seconds. Unfortunately, the state of expired timers will remain cached for the duration of script execution. As such, this script will leave “First” on display after it is called by “Third”. Refreshing the deck will not reset the expired timers. The only workaround at present is to flush the cache to fully reload and initialize the deck.

Selecting or deselecting an option from a <select> list triggers an "onpick" event as shown below:

```
<!-- An example that captures an -->
<!-- onpick event -->
<card id="Options" title="Our Options">
  <p align="center">
    Let's display our options
  </p>

  <p>
    <select name="TheSelection" multiple="true">
      <option value="Option1">Option #1</option>
      <option value="Option2" onpick="#Option2">
        Option #2
      </option>
      <option value="Option3">Option #3</option>
    </select>
  </p>

  <do type="accept" label="View Selection(s)?">
    <go href="#Selections" />
  </do>
</card>

<card id="Selections" title="Selected Option">
  <p align="center">
    You selected $TheSelection
  </p>
</card>

<card id="Option2" title="Triggered Card">
  <p align="center">
    You're in triggered Option #2 and you've
    selected $TheSelection
  </p>
</card>
```

When first executed, the user will be presented with a list of three items, any of which can be selected or deselected as required. Nothing of particular interest occurs if "Option1" or "Option3" are modified, but if any modification is made to "Option2", then the "Option2" card will be called.

There are four main tasks that help direct event handling within a WML script:

1. <go>
2. <prev />
3. <refresh>
4. <noop>

The <go> Task

As demonstrated in earlier scripts, <go> will load a specific card in the current deck, or from an external WML deck, WMLScript file, or URL. The table to follow details all attributes that can be used with <go>

Attribute	Required	Type	Description
href	Yes	Alphanumeric	A card in the current deck, a WML deck or WMLScript file, or an URL.
sendreferer	No	Boolean	If "true", passes the referring deck (URI). The default is "false".
method	No	Alphabetic	HTTP submission method: "post" or "get". The default is "get".
enctype	No	ContentType	Content type to submit to the server. Default "application/x-www-form-urlencoded" is used when method ="get". When method="post", can be default or "multipart/ form-data".
cache-control	No	"no-cache"	Must be set to "no-cache" if the attribute is used. This forces server to reload the deck on execution.
accept-charset	No	Alphanumeric	List of character encodings in comma or space delimited format for data that is acceptable to the processing server. Examples include: utf-8, UTF-8, us-ascii, iso-8859-1, utf-16. The default is "unknown".

The <prev /> Task

This task will navigate to the previous card in the history list. In doing so, it triggers an "onenterbackward" event in the destination card. The syntax is as follows:

```
<do type="accept" label="Previous">
  <prev />
</do>
```

The <refresh> Task

This task is used to refresh the display and variable settings, where <refresh /> will simply refresh the display. Use the following syntax to refresh variable settings that may be accessed within another card.

```
<!-- An example that deals with <refresh> -->
<card id="Main" title="Main entry point">
  <p align="center">
    Enter a string value:
    <input name="ChangeMe" />

    CARD: Main<br />
    The string value is set to $ChangeMe<br />

    <do type="accept" label="Modify">
      <go href="#ChangeIt" />
    </do>
  </p>
</card>

<card id="ChangeIt" title="Time for a change">
  <p align="center">
    CARD: ChangeIt<br />
    ChangeMe is currently set to $ChangeMe<br />

    <do type="accept" label="Update">
      <refresh>
        Enter a NEW string value:
        <input name="ChangeMe" />
      </refresh>

      <prev />
    </do>
  </p>
</card>
```

The <noop> Task

At first glance, inserting a task that performs "No Operation" within a deck seems a little odd. After all, why insert code to do nothing?

The answer is simple: to disable default functionality on some cell phones (but not on the BlackBerry handheld).

The code to follow would disable default functionality on many cell phones to return to the previous card:

```
<!-- An example that deals with <noop> -->
<card id="Main" title="Main entry point">
  <p align="center">
    You're in the main card!!

    <do type="accept" label="Next Card">
      <go href="#Next" />
    </do>
  </p>
</card>

<card id="Next" title="Bypassing previous">
  <p align="center">
    You're in the NEXT card!!

    <do type="prev">
      <noop />
    </do>
  </p>
</card>
```

Please let us know if you can find other uses for <noop> within a BlackBerry WML script.

Deck-Level Event Handlers

Default event handling can be specified for an entire deck through the use of the <template> tag as shown below:

```
<template>
  <do type="options" label="Back">
    <prev />
  </do>

  <do type="accept" label="Next Card">
    <go href="#Next" />
  </do>
</template>

<!-- An example that deals with <template> -->
<card id="Main" title="Main card">
  <p align="center">
    You're in the main card!!
  </p>
</card>

<card id="Next" title="NEXT card">
  <p align="center">
    You're in the NEXT card!!
  </p>
</card>
```

Note: *This handling can be overridden at a card level.*

The <meta> Tag

The most common use for meta tags within a WML deck is to set caching control. In the example to follow, the browser is directed to cache the deck for one minute, and then automatically reload the deck:

```
<head>
  <meta forua="true" http-equiv="Cache-Control"
content="max-age=60" />
```

```
</head>

<card id="IsCached">
  <p>
    This deck will be cached for one minute
  </p>
</card>
```

Set “max-age=0” to prevent deck caching.

Access Control

Deck access can be restricted to specific IP address(es) or domain(s) through the use of the <*access*> tag:

```
<wml>
  <head>
    <access domain="DOMAIN" [path="PATH"]/>
  </head>
</wml>
```

... where DOMAIN can be an IP address or an authenticated domain, and optional PATH can be a specific path off of the domain or IP such as “/secret”.

Creating Client-Server Applications

The most vital component of a dynamic WML site is to interact with server-side applications.

The WML example below prompts the user to input two fields, then submits the fields to a Perl script:

```
<!-- An example that submits a form -->
<card id="Submit"
  title="Form Submission Script">
  <p align="center">
    Form Submission<br />
    Enter Contact:
    <input name="Contact" />
    Enter Company:
    <input name="Company" />
  </p>

  <do type="accept">
    <go href="ProcessForm.pl">
      <postfield name="formContact"
        value="$Contact" />
      <postfield name="formCompany"
        value="$Company" />
    </go>
  </do>
</card>
```

This method of form submission cleanly declares fields with variables before form submission.

The Perl script below extracts the passed fields, then creates and displays a WML deck containing the passed fields:

```
#!/bin/perl
#
# Filename: ProcessForm.pl
# Purpose : to extract passed form fields
#
use CGI qw(:standard);

use CGI;

$query = new CGI;

$Contact = $query->param('formContact');
$Company = $query->param('formCompany');

$Deck = "Content-type: text/vnd.wap.wml

<?xml version=\"1.0\"?>

<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML
  1.2//EN\"
  \"http://www.wapforum.org/DTD/wml_1.1.xml\">

<wml>

<!-- after submission -->
<card id=\"Returned\"
  title=\"Form Returned Script\">
  <p align=\"center\">
    Returned Form<br />
    You passed $Contact and $Company
  </p>
</card>

</wml>";

print $Deck;
```

As you can see, it's not very difficult to get form submissions working.

An alternate method of accomplishing the same goal within the passed URL can be done by changing the code as follows:

```
<do type="accept">
  <go href="ProcessForm.pl?formContact=
    $(Contact)&amp;formCompany=$(Company)"/>
  </go>
</do>
```

It's far less clear than using the <postfield> tags, and far more visible to the user, but it will work.

That's it!

WML really is an easy to learn language that most people can program in without a huge learning curve.

If you want to expand your skills with WML, I strongly suggest that you read my article “WMLScript Compendium”, which has also been published this issue. You will quickly discover that WMLScript is an easy to learn subset of JavaScript that provides math functions, string handling, extended URL and WMLBrowser handling, and dialog boxes.

Understanding MDS: A Developer's Perspective

Tyler Lessard, Research In Motion

Some of the key challenges with developing and deploying wireless applications for the enterprise have traditionally been security, connectivity, manageability and extensibility.

How can you build a wireless extension to an existing application that resides behind the corporate firewall?

How do you ensure that it will be highly secure and remotely manageable?

How will that application make an end-to-end connection to the server that hosts the valuable corporate data?

Will this application be extensible to a number of wireless networks and how will it support roaming between countries?

And perhaps, most importantly, how much will it cost, and how long will it take to deploy and validate a solution that meets these requirements?

It is these problems that the BlackBerry Enterprise Server and Mobile Data Service (MDS) are designed to solve. By enabling wireless enterprise applications to be developed and deployed quickly and securely in a manageable fashion.

MDS is a feature of the BlackBerry Enterprise Server that enables applications beyond email and Personal Information Management (PIM) to securely establish http network connections behind the corporate firewall. By using the existing secure connection, the BlackBerry Browser and 3rd party Java applications for BlackBerry, can easily communicate with corporate web and application servers. This eliminates the need to worry about data encryption, connectivity behind the firewall or the underlying wireless network technology or carrier.

Although MDS can be thought of as a “secure pipe” for wireless applications to get behind the firewall, it is much more than that. MDS can play many passive and active roles in managing push- and pull-based connections to and from wireless applications, as well as user management, authentication and data optimization.

This article discusses many of the key features of MDS and its various roles in proxying and managing connections to and from wireless enterprise applications for BlackBerry.

BlackBerry Architecture and MDS Overview

The BlackBerry wireless solution consists of several main components:

- BlackBerry wireless handhelds
- Wireless networks
- BlackBerry Enterprise Server with MDS

BlackBerry Enterprise Server is installed and run behind the corporate firewall. It is integrated with the enterprise email system to enable wireless email and collaboration. BlackBerry Enterprise Server maintains a secure, authenticated connection to all handhelds within that organization via the BlackBerry wireless data center. The BlackBerry wireless data center provides a reliable, consolidated connection point to all wireless networks that BlackBerry operates on, including GSM/GPRS, CDMA, iDEN, Mobitex and DataTAC networks.

Wireless applications for BlackBerry handhelds use the MDS feature of the BlackBerry Enterprise Server to securely establish HTTP connections behind the firewall over the same secure channel as is used for wireless email. As wireless applications are typically deployed on top of an existing email/PIM deployment, this model enables the re-use of the existing BlackBerry infrastructure. MDS also provides the necessary interfaces to enable server side applications to proactively push content to the BlackBerry handheld in a secure and reliable manner.

MDS acts as a secure gateway that can be used by the following types of applications:

- BlackBerry Browser
- Custom Java/J2ME Applications developed by Enterprise IT or Systems Integrator (SI)
- Custom Java/J2ME Applications developed by 3rd party Independent Software Vendors (ISVs)

Conceptually, MDS provides a persistent, bi-directional Virtual Private Network (VPN) connection for the BlackBerry Browser and J2ME applications for BlackBerry.

Value to the Customer

BlackBerry Enterprise Server with MDS provides significant value to enterprise customers that are interested in extending their wireless strategy beyond email and PIM. MDS enables new wireless enterprise applications to be deployed within the enterprise without the need to:

- Purchase or manage new wireless connections or new pricing plans
- Worry about security or encryption beyond the firewall
- Worry about remote connectivity and support for multiple networks (including roaming!)
- Worry about data security on the handheld itself

Value to the Developer

BlackBerry Enterprise Server with MDS also provides significant value to wireless application developers, enabling quick and inexpensive wireless application development and deployment:

- Out-of-the-box secure connectivity to the corporate intranet: Inherent security, quicker time to launch, simpler deployment
- Leverage PUSH technology to improve the value of your solution ... get users addicted to your always-on applications!
- Network-independence:
  - Develop one application that serves all wireless networks and BlackBerry handhelds
  - Managed roaming: No concerns about connectivity while roaming, no re-connecting and dropping data

What Does MDS Do?

Connectivity

At its core, MDS acts as a proxy server or wireless gateway for applications on the handheld to connect to the wired intranet. In its role as a wireless gateway for handheld applications, MDS provides the following features:

- Protocol conversion
- Encryption and compression
- Queuing and flow control for Push-based connections

The following table details what protocols are used through each stage of transmission:

	BlackBerry Handheld - Wireless Network	Wireless Network-MDS (Over Internet, through firewall)	At MDS (Role of MDS)	MDS-Web or Application Server
Pull Request	Wireless Protocols	BlackBerry/IP(Port 3101 through firewall, Outbound Initiated)	Data Decryption Decompression Protocol Conversion DNS/Routing	HTTP (GET/POST) or TCP/IP Request
Pull Response	Wireless Protocols	BlackBerry/IP(Port 3101 through firewall, Outbound Initiated)	Data Encryption Conversion Compression Protocol	HTTP or TCP/IP Response

	Web/Application Server-MDS	At MDS	MDS-Wireless Network	Wireless Network-BlackBerry Handheld
Push (Server Initiated)	HTTP (POST) to MDS	Protocol Conversion, Compression, Data Encryption, Queuing and Flow Control	BlackBerry Protocols over IP (Port 3101 through firewall, outbound initiated)	Wireless Protocols

Data is compressed and encrypted for all communications between the handheld and the MDS.

Also worth noting is the additional queuing and flow control during a push-based connection

If a server-side application initiates a data push to a handheld application, but the handheld is out of coverage, MDS will queue the data in hopes of delivering the data once the user returns to coverage. This is discussed in more detail in the section on Push Management.

What Does MDS Do?

Security and Authentication

As previously discussed, MDS plays a key role in data encryption and security. MDS provides inherent data encryption from the enterprise to the wireless handheld application. MDS also supports additional encryption between MDS and the destination web or application server via HTTPS. As well, MDS provides support for standard enterprise authentication schemes for those situations where the enterprise system requires a username/password authentication.

MDS supports TLS/SSL for HTTPS connections, and Kerberos, HTTP Basic, and NT LAN Manager (NTLM) enterprise authentication. In the case of NTLM or Kerberos authentication, the authentication server is specified in the MDS configuration.

The following table details what security and authentication schemes can be used through all stages of transmission:

	BlackBerry Handheld - Wireless Network	Wireless Network-MDS (Over Internet, through firewall)	At MDS (Role of MDS)	MDS-Web or Application Server
Security	3DES encryption	3DES encryption	3DES De/Encryption Optional TLS/SSL De/Encryption	TLS/SSL Handshaking
Authentication	(User credentials included in payload)	(User credentials included in payload)	Username/Password submitted as response to challenge	Authentication (Kerberos, Basic, NTLM)

What Does MDS Do?

Cache User Information

MDS also acts as a proxy to manage cookies and user credentials on behalf of the handheld applications. Storing frequently used cookies and user information on MDS preserves wireless bandwidth, as this information is not sent unnecessarily over the wireless link.

Cookies

Cookies are generally used for session management during HTTP communication. When server-side applications respond to HTTP requests with cookies, MDS will store the cookie locally on the server. MDS will support all standard cookie directives such as timeouts.

User Information

As discussed in the previous section, MDS supports a variety of authentication schemes. In the situation where authentication is required, MDS will cache user credentials after a user is prompted for a username and password. The next time that user logs into the server and is challenged for a username and password, MDS will automatically authenticate the user with the cached credentials, removing the need for them to re-enter their credentials each time.

Note: *This feature can be turned off in the MDS configuration.*

PUSH Management: What Does it Do?

Server-side applications can securely push content to handheld applications via MDS. BlackBerry provides the ability to push content to the BlackBerry Browser or to custom Java applications that have been designed to listen for incoming pushes. For more information on building push-based applications, refer to the BlackBerry developer guides.

With regards to MDS, what happens during a push? The following is a step-by-step list of what MDS does after receiving a push request from a server-side application:

1. MDS receives HTTP POST with data to be pushed
2. MDS checks the local BlackBerry user database to confirm the existence of the destination user
3. If the user exists, MDS responds to the server application with an HTTP confirmation (HTTP 200 OK) and closes the connection
4. MDS now determines the status of the destination handheld to confirm whether it is online or offline
   - If the handheld is in coverage, the data is pushed immediately through the appropriate wireless network
   - If the handheld is out of coverage, the push request is queued within MDS (currently the data is queued in RAM on the server)

Push requests that are queued in MDS are then subject to Flow Control and Timeouts. MDS implements Timeouts to expire stale push requests and to preserve memory space on the server. If the handheld is out of coverage, the initial push requests are subject to Flow Control.

Flow Control:

- 5 push requests will be queued in MDS for the flow control timeout period (default: 10 minutes)
- No further requests will be pushed until these 5 are received by the handheld

If more data is pushed to the disconnected handheld while the Flow Control bucket is full, they are subject to a different Timeout.

Time-out:

- Push requests received while the Flow Control bucket is full (5 requests that have not been delivered) are subject to this timeout
- These push requests will be discarded after the period specified by the timeout parameter (default: 3 minutes)

Therefore, if the MDS configuration values are left at their default values and 12 requests are pushed to a handheld that is out of coverage, requests 6-12 will be discarded after 3 minutes and requests 1-5 will be discarded after 10 minutes.

Note: *Increasing the Flow Control and Timeout parameters will increase the length of time that push requests can be queued. This would have the effect of increasing reliability of pushes, at the expense of RAM usage. Also note that these timeout settings are global for all pushes and all users.*

MDS Configuration and Policy Settings

MDS includes a number of configurable parameters that can impact how applications behave. If you are deploying applications that will use MDS as a gateway, it is worth your time to become familiar with these parameters. To follow are some of the key MDS configuration parameters that are relevant to developers or administrators deploying wireless applications for BlackBerry:

Push Listen Port

- Port that MDS listens on for incoming pushes from server-side applications. Keep in mind that this is configurable, so try not to hard code the destination server port in push applications

FlowControl Timeout and Push Queue Timeout

- As described above, these parameters defines the expiry time for pushes to a handheld that is out of coverage

Max Payload

- Maximum response size that MDS will allow for a single request

HTTP Logging

- Enable / Disable detailed logging of HTTP requests, typically used for debugging only

TLS

- Specifies whether or not HTTPS connections will be allowed to "untrusted" servers. To become a trusted server for HTTPS connections, the server certificate must be injected into MDS

Description	Parameter	Default
FlowControl Timeout	IPPP.queue.flowcontrol. timeout	10 min.
Push Queue Timeout	IPPP.connection.timeout	3 min.
Max Payload	IPPP.connection. MaxNumberOfKBytesToSend	128k
HTTP Logging	Application.handler. http.logging	False
TLS	Application.hander.tls. allowUntrustedServer	False

Note: *These parameters are defined in a text file called "rimpublic.property" which can be found in the /MDS/config/ folder.*

BlackBerry Enterprise Server also includes a set of IT Policies that are configurable for each user's handheld. These policies dictate what the handheld and local applications can and cannot do. The following policies are relevant to developers:

- AllowBrowser: Dictates whether or not users can use browser (default: Yes)
- DisallowThirdPartyAppDownloads and AllowRunThirdPartyApps: Controls use of Java apps on handheld (default: Allow applications to be loaded and run)
- AllowThirdPartyUseSerialPort: Dictates whether or not applications can communicate through the serial or USB port (default: Yes)
- AllowThirdPartyUsePersistentStore: Dictates whether or not applications can use the persistent store API to write to flash (default: Yes)
- AllowInternalConnections: Dictates whether or not applications can connect via MDS (default: Yes)
- AllowExternalConnections: Dictates whether or not applications can connect via public gateways, such as WAP gateways (default: Yes)
- AllowSplitPipeConnections: Dictates whether or not a single application can connect via both MDS and an public gateway (such as WAP) (default: Yes)

WMLScript Compendium

Richard Evers, Editor

A new plague hit the world in 1995 in the form of hack generalists who could copy and paste together awful HTML sites.

As the years trickled by, and dot com bubbles burst, bandwidth improved along with the languages, techniques and software used to create really decent web sites.

While many conventional sites are pretty good now, the wireless world of WAP is still wallowing in an awkward stage of existence. Unlike traditional sites, WAP sites look basic at best because they are constrained to work within the confines of wireless devices with small screens, modest memory and limited bandwidth.

The Wireless Markup Language (WML) is a broad, fairly easy to use language that permits site developers to create reasonable content that can be viewed on all WAP browsers. Unfortunately, it lacks many of the useful things found in a true scripting language.

The solution is WMLScript, a wireless scripting language that can co-exist with WML decks. It is similar to JavaScript, and is a modified subset of ECMAScript.

WMLScripts are independent files called as external references within WML decks. They are compiled into byte code at run time on the server before being sent to the WAP browser. Like C, C++ and Java, the language is case sensitive and has a construct that is similar to C.

The format of a WMLScript function is as follows:

```
extern function NAME( [PARAMETERS(S)] )
{
  // body of function
}
```

The 'extern' keyword is used to make the function public to external files. Do not include ‘extern’ on functions that are only called from within the script.

The following example consists of a WML deck that calls an external WMLScript function:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD
  WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="page1" title="Execute Script">
    <do type="options" label="GetNews">
      <go href="go_mobile.wmls#surf('news')"/>
    </do>
    <do type="options" label="GetFlightStat">
      <go href="go_mobile.wmls#
        surf('flightstat')"/>
    </do>
  </card>
</wml>
```

The highlighted lines in the earlier example contain references to the external WMLScript to follow:

```
/*
 * Filename: go_mobile.wmls
 * Function: surf()
 * Purpose : selectively surf the mobile web
 */
extern function surf(the_url)
{
  if (the_url == "news")
  {
    WMLBrowser.go
      ("http://mobile.globeandmail.com");
  }
  else if (the_url == "flightstat")
  {
    WMLBrowser.go("http://mobile.aircanada.ca/
      aircanada/flstatus.wml");
  }
}
```

Parameter passing is fairly lax where type is not specified in the parameter list. The caller of the function is responsible for making sure that parameters passed to a function are in the expected format and sequence.

Reserved Words

access	domain	http	struct
agent	else	if	super
break	equiv	import	switch
catch	enum	isvalid	throw
class	export	meta	try
const	extends	name	typeof
continue	extern	path	url
debugger	finally	private	use
default	for	public	user
div	function	return	var
do	header	sizeof	while

Reserved words cannot be used to name functions or variables.

Statements

WMLScript statements are as follows:

break	for	return
continue	if/else	while

Operators and Expressions

Arithmetic operators:

+	plus	-	minus
*	multiply	/	divide
%	remainder	div	perform integer division

Bitwise operators:

<<	left shift	>>	right shift
&	AND	|	OR
^	Exclusive OR	>>>	bitwise right shift with zero fill

Assignment operators:

=	assignment	+=	add and assign
-=	subtract and assign	*=	multiply and assign
/=	divide and assign	div=	integer divide and assign
%=	remainder and assign	<<=	bitwise left shift and assign
>>=	bitwise right shift and assign	>>>=	bitwise right shift zero fill and assign
&=	bitwise AND and assign	|=	bitwise OR and assign
^=	bitwise XOR and assign		

Unary operators:

+	plus	-	minus
--	post or pre-decrement	++	post or pre-increment
~	bitwise NOT		

Logical operators:

&&	AND	||	OR
!	NOT		

String operators:

+	concatenate	+=	concatenate

Comparison operators:

<	less than	<=	less than or equal
==	equal	>	greater than
>=	greater than or equal	!=	not equal

Conditional operator

- ?: Example:

```
var IsOkay = ( Want == "Food" ) ? 1 : 0;
```

Other operators

WLMScript supports comma-operators:

```
for (el = 0, id = 100; el < 10; el++, id+=10)
    // do something
```

While WMLScript is weakly typed, it does support boolean, integer, floating-point, string and invalid data types. The **typeof** operator will return an integer value that identifies the data type as shown below:

1. integer
2. floating-point
3. string
4. boolean
5. invalid

Example:

```
var data = "Mares eat oats";
var result = typeof data; // result equals 2
```

An 'isvalid' operator is provided to safely test whether an expression is valid. It will return true if the passed expression is valid, else returns false. The syntax is as follows:

```
var IsOk_1 = isvalid (99/0); // false
var IsOk_2 = isvalid (99/1); // true
```

Libraries

The strength of WMLScript is largely contained with the function libraries. The libraries include functions to deal with numeric values, dialog and alerts, strings, relative and absolute URLs, and the browser.

Library: Lang

Lang contains the core library functions:

abort	isInt	minInt
abs	max	parseInt
characterSet	maxInt	random
exit	min	seed

Lang.abort()

Description:

- Aborts execution of WMLScript and returns passed string to caller.

Syntax:

- Lang.abort(ErrorMessage);

Parameters:

- An error message

Returns:

- Does not return

Example:

```
Lang.abort("Script failure on line 123");
```

Lang.abs()

Description:

- Returns the absolute value of the passed number.

Syntax:

- Value = Lang.abs(Number);

Parameters:

- An integer or float value

Returns:

- The absolute value returned as an integer or float value

Example:

```
// will return positive 98765
var i_ret = Lang.abs(-98765);

// will return positive 987.65
var f_ret = Lang.abs(-987.65);
```

Lang.characterSet()

Description:

- Returns a value that identifies the supported character set.
- Visit: **http://www.iana.org/assignments/character-sets** ... for an up-to-date list of character sets.

Syntax:

- charset = Lang.characterSet();

Parameters:

- void

Returns:

- Numeric character-set identifier

Example:

```
// Assigned MIB enum Numbers
//--------------------------
//0-2       Reserved
//3-999     Set By Standards Organizations
//1000-1999 Unicode and ISO/IEC 10646
//2000-2999 Vendor
// for example, returns 3 for US-ASCII
var charset = Lang.characterSet();
```

Lang.exit()

Description:

- Terminates script and returns passed message to caller.

Syntax:

- Lang.exit(Message);

Parameters:

- A message to pass back to the caller

Returns:

- Does not return

Example:

```
Lang.exit("Exit stage left");
```

Lang.isInt()

Description:

- Tests if the passed string can be converted into an integer value using parseInt()

Syntax:

- isOkay = Lang.isInt(StringValue);

Parameters:

- A string representation of an integer value.

Returns:

- Boolean true if string will convert to integer form, else false.

Example:

```
isOkay1 = Lang.isInt("98765"); // true
isOkay2 = Lang.isInt("-98765"); // true
isOkay3 = Lang.isInt("9.8e2"); // true
isOkay4 = Lang.isInt("intni"); // false
```

Lang.max()

Description:

- Determines the maximum value of two passed values in either integer or floating-point form.

Syntax:

- maxValue = Lang.max(Value1, Value2);

Parameters:

- Two integer or floating-point values to compare

Returns:

- The highest value passed

Example:

```
// returns 13
var maxValue = Lang.max(12, 13);
```

Lang.maxInt()

Description:

- Returns the maximum value of an integer.

Syntax:

- theMax = Lang.maxInt();

Parameters:

- void

Returns:

- Maximum integer value

Example:

```
// returns 2147483647
var theMax = Lang.maxInt();
```

Lang.min()

Description:

- Returns the minimum value of two passed values in either integer or floating-point form.

Syntax:

- theMin = Lang.max(Value1, Value2);

Parameters:

- Two integer or floating-point values to compare

Returns:

- The smallest value passed

Example:

```
// returns 12
var theMin = Lang.max(12, 13);
```

Lang.minInt()

Description:

- Returns the minimum value of an integer.

Syntax:

- theMin = Lang.minInt();

Parameters:

- void

Returns:

- Minimum integer value

Example:

```
// returns -2147483648
var theMin = Lang.maxInt();
```

Lang.parseInt()

Description:

- Converts a string into an integer value.

Syntax:

- theInt = Lang.parseInt(StringInt);

Parameters:

- Integer value in string form

Returns:

- Integer value

Example:

```
// returns 9876
var theInt1 = Lang.parseInt("9876");

// returns 987
// stops parsing on first error
var theInt2 = Lang.parseInt("9876Hi!!");
```

Lang.random()

Description:

- Returns a random number between 0 and the passed value

Syntax:

- rndValue = Land.random(MaxRange);

Parameters:

- Maximum integer value to draw from

Returns:

- A random integer within the specified range

Example:

```
var rndValue = Land.random(9867);
```

Lang.seed()

Description:

- Initializes the random number generator

Syntax:

- ret = Lang.seed(SeedValue)

Parameters:

- An integer seed value

Returns:

- An empty string

Example:

```
var ret = Lang.seed(98765)
```

Library: Dialogs

Dialogs contains three dialog handlers:

alert	confirm	prompt

Dialogs.alert()

Description:

- Displays passed message and waits for a response.

Syntax:

- var ret = Dialogs.alert(Message);

Parameters:

- Message to display

Returns:

- An empty string

Example:

```
var ret = Dialogs.alert("Wake up now!");
```

Dialogs.confirm()

Description:

- Displays passed message, waits for a response, then returns a boolean value that corresponds to the selected option.

Syntax:

- var isOkay = Dialogs.confirm(Message, Okay, Cancel);

Parameters:

- All parameters are string or string literals
- #1: the confirmation message
- #2: user option #1/2
- #3: user option #2/2

Returns:

- Boolean true if parameter 2 is selected else false

Example:

```
var isOkay = Dialogs.confirm
    ("Exit", "Yes", "No");
```

Dialogs.prompt()

Description:

- Displays passed message, waits for a response, then returns the user's response, or the default response if nothing had been entered.

Syntax:

- var gotIt = Dialogs.prompt(Message, Default);

Parameters:

- All parameters are string or string literals
- #1: the prompt message
- #2: default response if nothing entered

Returns:

- String response

Example:

```
var Age = Dialogs.prompt("Your age", "30");
```

Library: String

String contains 16 functions:

charAt	find	length	squeeze
compare	format	removeAt	subString
elementAt	insertAt	replace	toString
elements	isEmpty	replaceAt	trim

String.charAt()

Description:

- Returns a single character located at the passed offset position within the passed string

Syntax:

- var theChar = String.charAt(Buffer,Offset);

Parameters:

- #1: source string buffer
- #2: offset position within source buffer

Returns:

- A single character located at the offset position

Example:

```
// return "c"
var theChar1 = String.charAt("abcdef",3);

// returns ""
var theChar2 = String.charAt("abcdef",8);
```

String.compare()

Description:

- String comparision where ranking is performed based on the ASCII value of each character within the string

Syntax:

- var theRes = String.compare(String1, String2);

Parameters:

- #1: first string to compare
- #2: second string to compare

Returns:

- 0 if the strings are identical
- -1 if the first string is less than the second
- 1 if the second string is less than the first

Example:

```
// return 0
var theRes1 = String.compare("ABC", "ABC");

// returns 1
var theRes2 = String.compare("abc", "ABC");

// returns -1
var theRes3 = String.compare("ABC", "abc");
```

String.elementAt()

Description:

- Locates a single element within passed string buffer.

Syntax:

- var Field = String.elementAt(Buffer, Element, Delim);

Parameters:

- #1: string buffer containing delimited fields
- #2: numeric value set to the desired field number
- #3: character(s) used to delimit fields

Returns:

- The requested field, or the first field if a negative element is passed, or the last field if the element exceeds the total number of fields.

Example:

```
// returns "transformation"
var Field = String.elementAt("In an
extraordinary transformation of heat to light,
Gibbon rested.", 4, " ");
```

String.elements()

Description:

- Counts how many times a delimiter occurs within a passed buffer

Syntax:

- var howMany = String.elements(Buffer, Delimiter);

Parameters:

- #1: string buffer
- #2: field delimiter

Returns:

- The total count of Delimiter within the Buffer

Example:

```
// returns 20
var howMany = String.elements("This descent from
unity into multiplicity recalled Constantine's
timid policy of 'dividing whatever is united',
but its effects were far different", " ");
```

String.find()

Description:

- Searches for the first occurrence of the passed substring with the passed buffer

Syntax:

- var theFirst = String.find(Buffer, SubString);

Parameters:

- #1: source string buffer
- #2: substring to search for within passed buffer

Returns:

- Offset value of first occurence of substring (0-n), or -1 if substring is not found

Example:

```
// returns 2
var theFirst = String.find( "Waterloo", "ter" );
```

String.format()

Description:

- Formats the passed numeric value as a string

Syntax:

- var looksNice = String.format(FormatString, Value);

Parameters:

- #1: Formatting string configured as follows:
  - "%[width][.precision] type" where "**%**" and **type** are mandatory

width	minimum number of characters that must be returned in the string
.precision	required decimal precision that is set based on the setting of "**type**"
type = 'd'	Source is treated as a positive or negative Integer value in the form of [-]9999 where 9999 is one or more decimal digits.. If ".precision" is set, then the output value is padded on the left side with up to the ".precision" number of zeroes.

type = 'f'	Source is treated as a positive or negative Floating Point value in the form of [-]9999.9999 where 9999 is one or more decimal digits. If ".precision" is set, then is used to set the number of digits after the decimal point, with at least one digit appearing before the decimal point. The default precision is 6. If a 0 or nothing has been specified after the '.' then the decimal component is truncated.
type = 's'	Source is treated as a String. The "width" argument can be used to set the minimum string size. The ".precision" argument can be used to set the maximum string size.

Returns:

- A formatted string

Example:

```
// returns "9876"
var s1 = String.format("%8d", 9876);

// returns "009876"
var s2 = String.format("%8.6d", 9876);

// returns "9876.543"
var s3 = String.format("%8.3f", 9876.54321);

// returns "NCC-1701"
var s4 = String.format("NCC-%4d", 1701);

// returns "Hobbits rule!"
var s5 = String.format("Hobbits %s", "rule!");

// returns "        98.765%"
var s6 = String.format("%10.3F%%", 98.7654);
```

String.insertAt()

Description:

- Creates a new string from the passed buffer that includes the passed field and field delimiter, inserted at the passed field element number.

Syntax:

- var nStr = String.insertAt(Buff, Field, Element, Delim);

Parameters:

- #1: string buffer containing delimited fields
- #2: string field to insert
- #3: Numeric element (0-n) where the field is to be inserted. If less than 0 then is 0 is used. If greater than maximum number of elements, then the new field is appended to the buffer.
- #4: Character delimited to insert after the Field

Returns:

- Resulting string

Example:

```
// results: "1|99|2|"
var nStr = String.insertAt("1|2|","99",1,"|");
```

String.isEmpty()

Description:

- Determines is the passed string is empty

Syntax:

- var IsNULL = String.isEmpty(Buffer);

Parameters:

- A string buffer

Returns:

- Boolean true if the string is empty else false.

Example:

```
// returns true
var NULLString = "";
var IsNULL = String.isEmpty(NULLString);
```

String.length()

Description:

- Returns the length of a passed string

Syntax:

- var theLength = String.length(Buffer);

Parameters:

- A string buffer

Returns:

- The string length (0-n)

Example:

```
// returns 6
var theLength = String.length("yellow");

// returns 0
var theLength = String.length("");
```

String.removeAt()

Description:

- Removes a field from passed buffer at a specific element position.

Syntax:

- var Res = String.removeAt(Buffer,Element,Delim);

Parameters:

- #1: source string buffer
- #2: field element to remove from buffer
- #3: character delimiter used to separate fields

Returns:

- String buffer without the requested element

Example:

```
// returns  "1|3|"
var Res = String.removeAt("1|2|3|",1,"|");
```

String.squeeze()

Description:

- Creates a string where all repeat white spaces in the passed string buffer are reduced to single spaces.

Syntax:

- var SqzMe = String.squeeze(Buffer);

Parameters:

- A string buffer

Returns:

- The string buffer with “squeezed” spaces

Example:

```
// return "Will B Good"
var SqzMe = String.squeeze("Will  B  Good");
```

String.subString()

Description:

- Returns a portion of the passed string

Syntax:

- var SS = String.subSTring(Buffer, Start, Size);

Parameters:

- #1: source string buffer
- #2: starting offset into the source string (0-n)
- #3: the number of characters to extract

Returns:

- The requested substring

Example:

```
// returns "ffe"
var SS = String.subString("Coffee", 2, 3);
```

String.toString()

Description:

- Returns a string representation of the passed parameter.

Syntax:

- var theString = String.toString(theValue);

Parameters:

- Anything

Returns:

- A string

Example:

```
var theString = String.toString(98.76);
```

String.trim()

Description:

- Returns passed string without leading and trailing spaces

Syntax:

- var isTrimmed = String.trim(Buffer);

Parameters:

- A string buffer

Returns:

- The string buffer without leading and trailing spaces

Example:

```
// returns "Well Padded"
var isTrimmed = String.trim(" Well Padded "
```

Library: URL

URL contains 14 functions:

escapeString	getPath	isValid
getBase	getPort	loadString
getFragment	getQuery	resolve
getHost	getReferer	unescapeString
getParameters	getScheme	

URL.escapeString()

Description:

- Returns a string where special characters are changed into hexadecimal escape sequences. The escaped characters are as follows:
  - Control Characters (ASCII %00 to %1F) and %7F
  - Space (ASCII %20)
  - Upper range (ASCII %8F to %FF)
- Reserved Characters:

;	**/**	**?**	**:**	**@**
&	**=**	**+**	**$**	

- Not Recommended Characters

{	}	|	\
^	[	]	‘

- Delimiters:

<	>	#	%	“

Syntax:

- var newStr = URL.escapeString(theURL);

Parameters:

- String buffer containing unescaped URL

Returns:

- String buffer containing escaped URL

Example:

```
// results: "http%3a%2f%2frim.com%2f"
URL.escapeString("http://rim.com/");
```

URL.getBase()

Description:

- Returns the absolute URL (without fragment) of the current WMLScript

Syntax:

- var absURL = URL.getBase();

Parameters:

- void

Returns:

- String of absolute URL

Example:

```
// if URL = "http://rim.com/script.wmls#frag"
// then returns "http://rim.com/script.wmls"
var absURL = URL.getBase();
```

URL.getFragment()

Description:

- Returns the fragment portion of the passed URL

Syntax:

- var theFrag = URL.getFragment(theURL)

Parameters:

- An URL

Returns:

- URL fragment

Example:

```
// returns "frag"
var theURL = "http://rim.com/script.wmls#frag"
var theFrag = URL.getFragment(theURL);
```

URL.getHost()

Description:

- Returns the host specified within the passed URL

Syntax:

- var theHost = URL.getHost(theURL);

Parameters:

- An URL

Returns:

- Host component

Example:

```
// returns "www.rim.com"
var theURL = "http://www.rim.com/script.wmls";
var theHost = URL.getHost(theURL);

// returns ""
theURL = "script.wmls";
theHost = URL.getHost(theURL);
```

URL.getParameters()

Description:

- Returns the parameters within the last path segment of the passed URL.

Syntax:

- var parms = URL.getParameters(theURL);

Parameters:

- An URL

Returns:

- The parameters

Example:

```
// returns "foo;bar"
var theURL = "http://rim.com/foo.php;foo;bar";
var parms = URL.getParameters(theURL);
// returns ""
theURL = "http://www.rim.com/script.wmls";
parms = URL.getParameters(theURL);
```

URL.getPath()

Description:

- Returns the path specified within the passed URL

Syntax:

- var thePath = URL.getPath(theURL);

Parameters:

- An URL

Returns:

- The path component

Example:

```
// returns "/foo/bar.php"
var theURL = "http://rim.com/foo/bar.php";
var thePath = URL.getPath(theURL);

// returns ""
theURL = "http://rim.com/";
thePath = URL.getPath(theURL);
```

URL.getPort()

Description:

- Returns the port specified within the passed URL

Syntax:

- var thePort = URL.getPort(theURL);

Parameters:

- An URL

Returns:

- The port component as a string

Example:

```
// returns "80"
var theURL = "http://www.rim.com:80";
var thePort = URL.getPort(theURL);

// returns ""
theURL = "http://www.rim.com";
thePort = URL.getPort(theURL);
```

URL.getQuery()

Description:

- Returns the query portion of the passed URL

Syntax:

- var theQ = URL.Query(theURL);

Parameters:

- An URL

Returns:

- The query

Example:

```
// returns "bar"
var theURL = "http://rim.com/ok.php?foo=bar";
var theQ = URL.getQuery(theURL);

// returns ""
thePort = URL.getPort("http://www.rim.com");
```

URL.getReferer()

Description:

- Returns the smallest relative URL for the page, deck or script that called the current script.

Syntax:

- var whoCalled = URL.getReferer();

Parameters:

- void

Returns:

- The referer

Example:

```
// might return full URL
// or something relative such as "mydeck.wml"
// it will return "" if there is no referer
var whoCalled = URL.getReferer();
```

URL.getScheme()

Description:

- Returns the scheme within the passed URL

Syntax:

- var theScheme = URL.getScheme(theURL);

Parameters:

- An URL

Returns:

- The scheme

Example:

```
// returns "http"
var theURL = "http://www.rim.com/";
var theScheme = URL.getScheme(theURL);

// returns ""
var theURL = "www.rim.com/";
var theScheme = URL.getScheme(theURL);
```

URL.isValid()

Description:

- Validates the syntax of the passed URL

Syntax:

- var isOkay = URL.isValid(theURL);

Parameters:

- An URL

Returns:

- Boolean true if syntax is correct else returns false

Example:

```
// returns true
var theURL = "http://www.rim.com/";
var isOkay = URL.isValid(theURL);

// returns false
theURL = "http:/www.rim.com/";
isOkay = URL.isValid(theURL);
```

URL.loadString()

Description:

- Returns the content referred by the passed absolute URL and content type

Syntax:

- var theContent = URL.loadString(theURL, theCT);

Parameters:

- #1: string containing an absolute
- #2: string containing the Content Type that must prefix with “text/”

Returns:

- A string buffer containing the requested page, deck or script.

Example:

```
// returns the page contents
var theURL = "http://www.rim.com/index.shtml";
var theCT  = "text/plain" );
var theContent = URL.loadString(theURL, theCT);
```

URL.resolve()

Description:

- Combines the passed base and relative URLs to return an absolute URL

Syntax:

- var absURL = URL.resolve(baseURL, relURL);

Parameters:

- #1: base URL (e.g. “http://www.rim.com/”)
- #2: relative URL (e.g. “index.shtml”)

Returns:

- The resulting absolute URL

Example:

```
// returns "http://www.rim.com/index.shtml"
var baseURL = "http://www.rim.com/";
var relURL = "index.shtml" );
var absURL = URL.resolve(baseURL, relURL);
```

URL.unescapeString()

Description:

- Returns a string where escaped characters have been restored to original form

Syntax:

- var newURL = URL.unescapeString(escURL);

Parameters:

- An escaped URL

Returns:

- An unescaped URL

Example:

```
// results: "http://rim.com/"
var escURL = "http%3a%2f%2frim.com%2f";
var newURL = URL.unescapeString(escURL);
```

Library: Browser

Browser contains 7 functions:

getCurrentCard	prev
getVar	refresh
go	setVar
newContext	

Browser.getCurrentCard()

Description:

- Returns the smallest relative URL of the current card being processed by the browser. If the current card has a different base than the current script, will return the absolute URL of the card.

Syntax:

- var relURL = Browser.getCurrentCard();

Parameters:

- void

Returns:

- A relative or absolute URL

Example:

```
// results: e.g. "validate#doit"
var relURL = Browser.getCurrentCard();
```

Browser.getVar()

Description:

- Returns the value of the passed variable name within the current browser context

Syntax:

- var varVal = Browser.getVar(strName);

Parameters:

- Variable name

Returns:

- String value or invalid if not found

Example:

```
// results: e.g. "jdoe"
var varVal = Browser.getVar("userid");
```

Browser.go()

Description:

- Navigates browser to an URL

Syntax:

- var ret = Browser.go(navURL);

Parameters:

- A relative or absolute URL

Returns:

- An empty string

Example:

```
// relative navigation
var ret = Browser.go("newpage.wml");

// absolute navigation
var ret = Browser.go("http://www.xyzzy.com/");
```

Browser.newContext()

Description:

- Resets browser context thus clearing all variables

Syntax:

- var ret = Browser.newContext();

Parameters:

- void

Returns:

- An empty string

Example:

```
var ret = Browser.newContext();
```

Browser.prev()

Description:

- Navigate to the previous card

Syntax:

- var ret = Browser.prev();

Parameters:

- void

Returns:

- An empty string

Example:

```
var ret = Browser.prev();
```

Browser.refresh()

Description:

- Refreshes current page by pulling it from the server

Syntax:

- var ret = Browser.refresh();

Parameters:

- void

Returns:

- An empty string

Example:

```
var ret = Browser.refresh();
```

Browser.setVar

Description:

- Sets the value of a variable

Syntax:

- var isSet = Browser.setVar(varName, varValue);

Parameters:

- #1: variable name
- #2: value to assign to variable

Returns:

- Boolean true on success else false

Example:

```
var isSet = Browser.setVar("password", "plugh");
```

Creating a RSS/RDF Push Service for BlackBerry

Mark Sohm, Research In Motion

These days, more and more web sites are offering *Really Simple Syndication*[1] (*RSS*) or *Resource Description Framework*[2] (*RDF*) services. These services allow RSS and RDF readers to connect to a website and retrieve a list of headlines that can be presented to the user. Combining feeds from multiple sites gives a reader a quick, one-stop location where they can browse the latest information from all of their favorite web sites. More information on RSS and RDF can be found on their official home pages. You can find links to these pages in "*Appendix A - Site References*" below.

Selecting the Technology

BlackBerry is an ideal platform for easily taking advantage of these services. I thought that having a small, always-available list of updated headlines of favorite web sites would be a popular BlackBerry push service. I decided to create a browser channel push, which involves pushing an HTML or WML page to a BlackBerry handheld through the *BlackBerry Mobile Data Service (MDS)*. A new icon will appear on the home screen of the BlackBerry handheld once the push arrives. The HTML or WML page will open in the BlackBerry Browser when a user clicks on the icon.

Since this service would not require an application to reside on the BlackBerry handheld, I was free to choose from a variety of technologies to build the application. The service as a whole is required to perform the following tasks:

- manage a list of subscribers and web sites to which they have subscribed
- download and parse RSS/RDF pages
- send a customized HTML page to the user using the BlackBerry MDS

After considering my options, I chose to make use of the following technology. *MySQL*[3] as the database server to store all of the user, server, site and subscription information. For the user front end, I chose to make use of *Apache*[4] and *php*[5] to create a web site where users can manage their subscriptions, and add new RSS/RDF sites. To handle the download, RSS/RDF parsing, subscriber page generation and pushing to the MDS, I decided to create a server application using Java (J2EE).

Please read on for a journal-like description of the process behind the creation of this application. Full source code is included in the Appendices at the end of this document. I suggest jumping to the appendix and reading through the source files as you work your way through this article. I have refrained from including code samples inside this article to conserve space; most samples would be quite large and I believe it is more beneficial to view the source files in their entirety to see how the application flows and works together.

The Database Design

The first step in creating this system was to design a database that would hold all of the data used by the pieces of this application. The first and most obvious table required would hold the list of users or subscribers to the service. This would contain the following:

- email address used to push the HTML page to their BlackBerry handheld
- personalized HTML page that is pushed to them
- hash of their personalized page
- flag that is set when a user unsubscribes from the service
- timestamps of when their personalized page was last updated
- time of the last successful push to MDS

The next required table would be one that holds information about the RSS/RDF sites on the system. This table would store:

- the site name
- its URL
- user number of the user who added the site to the system
- a cache of the RSS/RDF page itself
- hash of the RSS/RDF page
- timestamp of when this page was last updated
- last modified date/time
- ETag header information from the web server that houses the RSS/RDF page

The ETag and last modified HTTP header information from the last HTTP request are sent with every subsequent request to access the RSS/RDF page. This allows the web server to determine whether the page has been updated since last retrieval, and not to send the whole page every time (it will simply reply with a Not Modified: 304 code instead). This is a feature that should be implemented with every RSS/RDF reader as it greatly reduces the load placed on a web server when many RSS/RDF readers connect simultaneously.

I also needed a table to store subscription information to know what sites users would want pushed to their BlackBerry handheld. The basic requirements of this table called for two columns:

- the site number
- the user number of the subscriber

Expanding on this, I also included some columns to store user preference information about the sites they subscribe to:

- an integer field to store the maximum number of articles to return from the site. This is useful for sites that include a large number of headlines in their RSS/RDF page.
- a flag that allows subscribers to select whether they want the article description to be sent along with the headline - for sites that include a description
- an integer field to allow a subscriber to specify the order they wish their subscribed sites to appear in on their BlackBerry handheld

The final table contains a list of BlackBerry Mobile Data Service (MDS) servers that will be used to push data to the subscribers. The hostname and port of each BlackBerry MDS server are stored here. This allows MDS servers to be added and updated as the BlackBerry environment grows.

With all the tables set up to hold the data, it was time to create something that could be used to populate them. I was now ready to begin creating the PHP and HTML pages that would be used by potential subscribers to sign up for the RSS/RDF push service. For the complete SQL commands used to create the tables described above, please see "*Appendix B - RSS/RDF Push Service MySQL Table Structures*".

The Front End

My decision to design the front end using web-based technologies was based on a number of factors. Almost every computer user is familiar with the web, and connecting via a web browser wouldn't require any additional software to be loaded on a subscriber's PC. In this way, they could even manage their subscription using their BlackBerry handheld! The RSS/RDF push service front end is made up of a very small number of files. There is a main index HTML page and four PHP pages.

The main index.html page simply prompts the user for their email address, and submits it to the rssmain.php page after verifying that the user has entered a properly formatted email address. Most of the work of the front end is done on the rssmain.php page. When a user first loads this page, it verifies whether their email address is already associated with a user entry in the user table. If it is not, one is added. It will then list all RSS/RDF pages that have been added to the server. This allows the user to manage subscriptions. From this the user can set the order they will be received, the number of headlines to be shown for each site, and whether or not the description will be shown with the headline if it is available. Users are also able to add additional RSS/RDF sites to the system from the rssmain.php page.

The remaining three pages have a much simpler function. The rssunsubscribe.php page is posted to from rssmain.php, and supplies the user number for the subscriber that is unsubscribing. The user is then removed from all entries of the subscriptions table, and the SendDelete flag in the users table is set to 1. The showpage.php also accepts the user number as a parameter and will retrieve and display the user's customized HTML page from the users table that is sent to them via the channel push. The last page, header.php, is included by the other pages and creates the connection to the MySQL database server.

Now that the front end was complete and I had a mechanism to enter subscriber and site information, I was ready to start building the push service application. To view the complete source code for the PHP pages, please see "*Appendix C - RSS/RDF Push Service php & HTML Pages*".

The Push Application

The next step would be to create the application that would take care of all the dirty work. It would need to:

- download the RSS/RDF pages
- parse them
- create a customized HTML page based on a user's subscription
- push that page out to their BlackBerry handheld through MDS

I created four Java classes to handle this operation:

- a database connector
- an RSS/RDF XML parser
- a push class
- a main controlling class that ties it all together

The first Java class I created, RSSDBConnector, was one that connects to the MySQL database server. Making a separate class for the database class made sense because if I need to change the database server, I just have to create a new dbconnector class. I made use of the *MySQL Connector/J*[6] JDBC driver when connecting to MySQL. The RSSDBConnector class allows connections to be created and closed to MySQL, as well as executing a query passed to it. I created two separate methods to handle queries, one for select statements that return a result set, and one for insert and update statements that return the number of rows affected by the query.

The next class on the list, RSSParser, needed to parse the xml RSS/RDF documents and break them apart, extracting the site name and titles, links and descriptions for every item. The *dom4j*[7] framework was used to handle the XML. This class is fairly simple in that you pass it a string of XML and get back a Vector containing the extracted information to be used in creating a subscriber's customized HTML page.

The RSSPusher class handles the actual push of data to MDS has been modelled after the BrowserPushDemo application that is included as a sample application with the *BlackBerry Java Development Environment* (*JDE*). Since the main framework was already in place, I was able to quickly make

some minor changes to customize the application to suit my needs. This class allows me to set all the parameters for a push and send it out to the specified MDS Server.

With all of the individual pieces in place, it was time to build the main controlling class that would utilize the database connector, RSS/RDF parser and push classes. The RSSPush class will create a connection to MySQL through RSSDBConnector and retrieve a listing of all RSS/RDF pages on the system. It will then go through each page to see if has been updated since the last check, then updates the page in the database if needed. Once that has been completed, the HTML pages for all subscribers are updated if pages they subscribe to have been updated. After all of the customized HTML pages are up to date they are pushed out to the subscribers. It then disconnects from MySQL and sleeps for 45 minutes before starting the process all over again.

With the Java push application in place, my BlackBerry RSS/RDF push service was almost ready to go. To see the full Java source code for the Java push application, please see "*Appendix D - RSS/RDF Push Service Java Source Code*". The final pieces of the puzzle are the icons that will be shown on the home screen of the BlackBerry after the push has taken place.

Graphics

Since I was using a browser channel push, the BlackBerry Browser would take care of rendering the HTML page that is pushed out to the subscribers. This means I didn't have to create an application to reside on the BlackBerry handheld. However, this push would place an icon on the BlackBerry handheld home screen. There are default icons used for channel pushes if an icon is not specified, but I wanted something that would easily identify my push page. I created a pair of 28x28 pixel images in .png format that are used for the read and unread icons. The icons for a channel push change between two states; read and unread, depending on whether or not a new page has been pushed since the user last viewed it. These images are placed on the web server and links to them are included with the push itself.

Finishing Up

With all of the pieces of the RSS/RDF push service ready to go, all that remained was to populate some data. I added the MDS servers and ports of the machines that are set to be the MDS Push Servers in my environment and added a handful of RSS/RDF sites I thought would be popular in attracting subscribers. As a matter of convenience, I set up the Java push application to run as a Windows Service using the *Java Service Wrapper*[8], allowing it to run at startup without requiring someone logged into the computer. With that in place I had a service setup to allow me, and any other BlackBerry users, to read the latest headlines from popular web sites, updated automatically, on our BlackBerry handhelds.

Appendix A - Site References

1. Really Simple Syndication (RSS): **http://blogs.law.harvard.edu/tech/rss**

2. Resource Description Framework (RDF): **http://www.w3.org/RDF/**

3. MySQL: **http://www.mysql.com/**

4. Apache: **http://www.apache.org/**

5. php: **http://www.php.net/**

6. MySQL Connector/J: **http://www.mysql.com/products/connector/j/**

7. dom4j: **http://www.dom4j.org/**

8. Java Service Wrapper: **http://wrapper.tanukisoftware.org/doc/english/introduction.html**

Appendix B - RSS/RDF Push Service MySQL Table Structures

```
#
# Table structure for table `users`
#

CREATE TABLE 'users' (
  'UserNo' int(11) NOT NULL auto_increment,
  'Email' varchar(64) NOT NULL default '',
  'LastPush' datetime NOT NULL default '0000-00-00 00:00:00',
  'Page' mediumtext,
  'Hash' bigint(20) NOT NULL default '0',
  'SendDelete' tinyint(4) NOT NULL default '0',
  'LastUpdated' datetime NOT NULL default '0000-00-00 00:00:00',
  PRIMARY KEY  ('UserNo'),
  UNIQUE KEY 'Unique_Email' ('Email')
) TYPE=MyISAM;

#
# Table structure for table `rsssites`
#

CREATE TABLE 'rsssites' (
  'SiteNo' int(11) NOT NULL auto_increment,
  'SiteName' varchar(100) NOT NULL default '',
  'AddedBy' int(11) NOT NULL default '0',
  'URL' varchar(200) NOT NULL default '',
  'Hash' bigint(20) default NULL,
  'Page' mediumtext,
  'LastUpdated' datetime NOT NULL default '0000-00-00 00:00:00',
  'LastModified' bigint(20) NOT NULL default '0',
  'ETag' varchar(32) NOT NULL default '""',
  PRIMARY KEY  ('SiteNo')
) TYPE=MyISAM;

#
# Table structure for table `subscriptions`
#

CREATE TABLE 'subscriptions' (
  'UserNo' int(11) NOT NULL default '0',
  'SiteNo' int(11) NOT NULL default '0',
  'NoOfArticles' tinyint(4) NOT NULL default '0',
  'ListOrder' tinyint(4) NOT NULL default '0',
  'ShowDescription' tinyint(4) NOT NULL default '0',
  PRIMARY KEY  (`UserNo`,`SiteNo`)
) TYPE=MyISAM;

#
# Table structure for table `mdsservers`
#

CREATE TABLE 'mdsservers' (
  'MDSServerNo' int(11) NOT NULL auto_increment,
  'Hostname' varchar(64) NOT NULL default '',
  'Port' smallint(6) NOT NULL default '0',
  'Notes' varchar(255) NOT NULL default '',
  PRIMARY KEY  ('MDSServerNo')
) TYPE=MyISAM;
```

Appendix C - RSS/RDF Push Service php & HTML Pages

Filename: index.html

```
<html>
<head>
<title>RSS/RDF Push</title>

<script language = "Javascript">

function echeck(str)
{
  var at="@";
  var dot=".";
  var lat=str.indexOf(at);
  var lstr=str.length;
  var ldot=str.indexOf(dot);

  if (str.indexOf(at)==-1)
  {
    alert("Invalid E-mail Address");
    return false;
  }

  if (str.indexOf(at)==-1 || str.indexOf(at)==0 || str.indexOf(at)==lstr)
  {
    alert("Invalid E-mail Address");
    return false;
  }

  if (str.indexOf(dot)==-1 || str.indexOf(dot)==0 || str.indexOf(dot)==lstr)
  {
     alert("Invalid E-mail Address");
     return false;
  }

   if (str.indexOf(at,(lat+1))!=-1)
  {
     alert("Invalid E-mail Address");
     return false;
   }

   if (str.substring(lat-1,lat)==dot || str.substring(lat+1,lat+2)==dot)
  {
     alert("Invalid E-mail Address");
     return false;
   }

   if (str.indexOf(dot,(lat+2))==-1)
  {
     alert("Invalid E-mail Address");
     return false;
   }

   if (str.indexOf(" ")!=-1)
  {
     alert("Invalid E-mail Address");
     return false;
   }

   return true;
}
```

```
function ValidateForm()
{
  var emailID=document.login.email;

  if ((emailID.value==null)||(emailID.value==""))
  {
    alert("Please Enter your Email Address");
    emailID.focus();
    return false;
  }

  if (echeck(emailID.value)==false)
  {
    emailID.focus();
    return false;
  }
  return true
 }
</script>

</head>
<body>
<h2>BlackBerry RSS/RDF Push</h2>
<form name="login" method="post" action="rssmain.php" onSubmit="return ValidateForm()">
<b>Enter your email address: </b><input type="text" name="email" size="20">
<input type="submit" name="submit" value="Enter"><br/>
</form>
Questions?  Email <a href=mailto:you@yourhost.com>you@yourhost.com</a>.
</body>
</html>
```

Filename: rssmain.php

```
<html>
<head>
<title>RSS/RDF Push Selection</title>
</head>
<body>
<?
include "header.php";

//Verifies a URL exists when a user attempts to add a new RSS/RDF site.
function urlExists($link)
{
  $link = ereg_replace("http://", "", $link);
  @list($domain, $file) = explode("/", $link, 2);
  $fid=@fsockopen($domain,80);
  @fputs($fid,"GET /$file HTTP/1.0\r\nHost: $domain\r\n\r\n");
  $gets = @fgets($fid, 1024);
  @fclose($fid);
  if (ereg("HTTP/1.1 200 OK", $gets))
  {
    return TRUE;
  }
  else
  {
    return FALSE;
  }
}

$messageline = "";
```

```
//Retrieve the user's number.
$select_query = "SELECT UserNo FROM Users WHERE Email='$email'";
$select_user = mysql_db_query($dbname, $select_query, $id_link) or die ("Select user failed.");

if (mysql_num_rows($select_user) == 0)
{
  $insert_query = "INSERT INTO Users (Email) VALUES('$email');";
  $insert_user = mysql_db_query($dbname, $insert_query, $id_link) or die ("Insert user failed.");

  $userNo = mysql_insert_id();
}
else
{
  $user_arr = mysql_fetch_array($select_user);
  $userNo = $user_arr['UserNo'];
}

//User is trying to add a new RSS/RDF site.
if (isset($addsite))
{
  $select_query = "SELECT SiteName FROM rssSites WHERE URL='$url';";
  $select_site = mysql_db_query($dbname, $select_query, $id_link) or die ("Select site failed.");

  //Ensure that a site with the same URL does not already exist.
  if (mysql_num_rows($select_site) == 0)
  {
    $select_query = "SELECT URL FROM rssSites WHERE SiteName='$sitename';";
    $select_site = mysql_db_query($dbname, $select_query, $id_link) or die ("Select site failed.");

    //Ensure that a site with the same name does not already exist.
    if (mysql_num_rows($select_site) == 0)
    {
      if (urlExists($url))
      {
        $insert_query = "INSERT INTO rssSites (SiteName, AddedBy, URL)
          VALUES('$sitename', $userNo, '$url');";
        $insert_site = mysql_db_query($dbname, $insert_query, $id_link)
          or die ("Insert site failed.");

        $messageline .= "New site was added.<br>";
      }
      else
      {
        $messageline .= "The server was unable to locate the specified URL. Site not added.<br>";
      }
    }
    else
    {
      $site_arr = mysql_fetch_array($select_site);
      $messageline .= "A site has already been added with the name $sitename.  It points to " .
        $site_arr['URL'] . ".  Site not added.<br>";
    }
  }
  else
  {
    $site_arr = mysql_fetch_array($select_site);
    $messageline .= "The URL $url has already been added under site name " . $site_arr['SiteName'] .
      ".  Site not added.<br>";
  }
}
```

```
//User is updating their site subscriptions.
if (isset($updatesubscription))
{
  //Delete all current subscriptions.
  $delete_query = "DELETE FROM Subscriptions WHERE UserNo='$userNo';";
  $delete_subscriptions = mysql_db_query($dbname, $delete_query, $id_link)
    or die ("Delete subscriptions failed.");

  //Insert all subscriptions and site preferences.
  for($count = 0; $count <= $total; $count++)
  {
    if (isset($siteselected[$count]))
    {
      if (isset($showdescription[$count]))
        $showdesc = 1;
      else
        $showdesc = 0;

      $insert_query = "INSERT INTO Subscriptions (UserNo, SiteNo, ListOrder, NoOfArticles,
        ShowDescription) VALUES('$userNo', '" . $siteno[$count] . "', '" . $siteorder[$count] . "',
        '" . $noofarticles[$count] . "', '$showdesc');";
      $insert_subscription = mysql_db_query($dbname, $insert_query, $id_link)
        or die ("Insert subscription $count failed.");

      //Reset the user's personal page and last updated date/time.
      $update_query = "UPDATE Users SET LastUpdated='0000-00-00 00:00:00' WHERE UserNo='$userNo'";
      $update_user = mysql_db_query($dbname, $update_query, $id_link)
        or die ("Insert subscription $count failed.");
    }
  }

}
?>

<h3>RSS/RDF push configuration for <?echo $email;?></h3>
Please allow up to 45 minutes for any changes to appear on your BlackBerry.<br/><br/>
<font color="red"><?echo $messageline;?></font><br>
<b>Add new RSS/RDF site to the server.</b>
<form name="new_rss" method="post" action="rssmain.php">
Name of website: <input type="text" name="sitename" size="50"><br>
URL to rss or rdf page: <input type="text" name="url" size="50"><br>
<input type="submit" name="submit" value="Add Site">
<input type="hidden" name="email" value="<?echo $email;?>">
<input type="hidden" name="addsite" value="1">
</form>
You can search <a href="http://www.syndic8.com/feedlist.php" target="_new">here</a> for rss news
feeds.<br><br>

Please select which rss feeds you would like pushed to your BlackBerry.

<table border="0">
<tr>
  <td valign="top"><b>Selected</b></td>
  <td valign="top"><b>List Order</b></td>
  <td valign="top"><b>Site</b></td>
  <td valign="top"><b>No. of Articles</b><br>(Enter 0 for all)</td>
  <td valign="top"><b>Show Description</b></td>
</tr>
<form name="rss_subscriptions" method="post" action="rssmain.php">
<?
```

```
$select_query = "SELECT SiteNo, SiteName, URL FROM rssSites ORDER BY SiteName;";
$select_sites = mysql_db_query($dbname, $select_query, $id_link) or die ("Select user failed.");

$total_sites = mysql_num_rows($select_sites);
$siteCount = 0;
$colourCount = 0;


//Display all RSS/RDF sites.
while ($sites_arr = mysql_fetch_array($select_sites)):
  $select_query = "SELECT ListOrder, NoOfArticles, ShowDescription FROM Subscriptions WHERE
UserNo='$userNo' AND SiteNo='" . $sites_arr['SiteNo'] . "';";
  $select_subscription = mysql_db_query($dbname, $select_query, $id_link) or die ("Select
subscriptions failed.");
  $subscription_arr = mysql_fetch_array($select_subscription);

  if ($colourCount == 1)
  {
    echo "<tr>\n";
    $colourCount = 0;
  }
  else
  {
    echo "<tr bgcolor='#DCDCDC'>\n";
    $colourCount++;
  }
  echo "<td><input type='checkbox' name='siteselected[$siteCount]' value='1'";
  if (mysql_num_rows($select_subscription) != 0)
    echo " checked ";

  echo "></td>\n";
  echo "<td><select name='siteorder[$siteCount]'>\n";

  for($count = 1; $count <= $total_sites; $count++)
  {
    echo "<option";
    if ($subscription_arr['ListOrder'] == $count)
      echo " selected ";

    echo ">$count</option>\n";
  }
  echo "</select></td>\n";
  echo "<td><a href='" . $sites_arr['URL'] . "' target='_new'>" . $sites_arr['SiteName'] .
    "</a></td>\n";
  echo "<td><input type='text' name='noofarticles[$siteCount]' size='5' value='" .
    $subscription_arr['NoOfArticles'] . "'></td>\n";
  echo "<td><input type='checkbox' name='showdescription[$siteCount]' value='1'";

  if ($subscription_arr['ShowDescription'] == 1)
    echo " checked ";

  echo "></td>\n";
  echo "</tr>\n";
  echo "<input type='hidden' name='siteno[$siteCount]' value='" . $sites_arr['SiteNo'] . "'>\n";

  $siteCount++;

endwhile;
?>
</table>
<br>
```

```
<input type="hidden" name="email" value="<?echo $email;?>">
<input type="hidden" name="updatesubscription" value="1">
<input type="hidden" name="total" value="<?echo $siteCount;?>">
<input type="submit" name="submit" value="Update Subscriptions">
</form>
<br><br>

<h3>Unsubscribe From All</h3>
<form name="unsubscribe" method="post" action="rssunsubscribe.php">
<input type="submit" name="submit" value="Unsubscribe">
<input type="hidden" name="userno" value="<?echo $userNo;?>">
<input type="hidden" name="unsubscribe" value="1">
</form>
Questions?  Email <a href=mailto:you@yourhost.com>you@yourhost.com</a>.

</body>
</html>
```

Filename: rssunsubscribe.php

```
<html>
<head>
<title>RSS Push Selection</title>
</head>
<body>
<?
include "header.php";

//Delete all of the users subscriptions.
$delete_query = "DELETE FROM Subscriptions WHERE UserNo='$userno'";
$delete_subscriptions = mysql_db_query($dbname, $delete_query, $id_link)
  or die ("Delete subscriptions failed.");

//Set the delete flag so a channel delete is pushed to the user on the next update.
$update_query = "UPDATE Users SET SendDelete='1' WHERE UserNo='$userno'";
$update_user = mysql_db_query($dbname, $update_query, $id_link) or die ("Update user failed.");

?>
<b>You have been unsubscribed from all rss feeds and the icon will be removed from your
BlackBerry.</b>
</body>
</html>
```

Filename: showpage.php

```
<?
include "header.php";
//This page will display a user's personalized RSS/RDF news page.

$select_query = "SELECT Page FROM Users WHERE UserNo='$userNo';";
$select_page = mysql_db_query($dbname, $select_query, $id_link) or die ("Select page failed.");
$page_arr = mysql_fetch_array($select_page);
echo $page_arr['Page'];
?>
```

Filename: header.php

```
<?
$dbname   =  'YourDatabaseName';
$hostname =  'localhost';
$username =  'UserName';
$password =  'Password';

$id_link = mysql_connect($hostname, $username, $password);

if (! $id_link):
  echo  'Connection to PHP has failed.';
  exit();
endif;
?>
```

Appendix D - RSS/RDF Push Service Java Source Code

Filename: RSSDBConnector.java

```
/*
 * RSSDBConnector.java
 *
 * Author:  Mark Sohm
 *
 * Handles connections to the MySQL database server.
 */


package com.msohm.rssPush;

import java.util.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import com.mysql.jdbc.*;


public class RSSDBConnector
{
  private String dbHost;
  private Connection conn;

  public RSSDBConnector()
  {
  }

  //Connect to MySQL
  public void dbConnect(String database, String host, String username, String password)
  {
    dbHost = "jdbc:mysql://" + host + "/" + database + "?user=" + username + "&password=" +
      password + "&autoReconnect=true";

    //Register the MySQL Driver With the DriverManager
    try
    {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
    } catch (Exception e)
    {
      throw new RuntimeException("Exception registering MySQL driver: " + e.getMessage());
```

```
  }

  //Connect to the database server.
  try
  {
    conn = DriverManager.getConnection(dbHost);
  } catch (SQLException e)
  {
    throw new RuntimeException("SQL Exception: " + e.getMessage() + "\n" + "SQLState: " +
      e.getSQLState() + "\n" + "VendorError: " + e.getErrorCode());
  }
}

//Insert or update a record in MySQL.
//Returns the number of rows affected by the query.
public int insertOrUpdate(String query)
{
  java.sql.Statement stmt = null;
  int rows = 0;

  try
  {
    stmt = conn.createStatement();
    rows = stmt.executeUpdate(query);

  } catch (Exception e)
  {
    throw new RuntimeException("Exception executing insert/update query: " + e.getMessage());
  } finally
  {
    //Clean up resources
    if (stmt != null)
    {
      try
      {
        stmt.close();
      } catch (SQLException sqlEx)
      {
        /* Ignore any exceptions */
      }
      stmt = null;
    }
  }

  return rows;
}

//Retrieve records via a select statement
public java.sql.ResultSet select(String query)
{
  java.sql.Statement stmt = null;
  java.sql.ResultSet rs = null;

  try
  {
    stmt = conn.createStatement();
    rs = stmt.executeQuery(query);

    return rs;

  } catch (Exception e)
```

```
    {
      throw new RuntimeException("Exception executing select query: " + e.getMessage());
    }
  }

  //Disconnect from MySQL
  public void dbDisconnect()
  {
    try
    {
      conn.close();
    } catch (Exception e)
    {
      /* Ignore any exceptions */
    }

    conn = null;
  }
}
```

Filename: RSSParser.java

```
/*
 * RSSParser.java
 *
 * Author:  Mark Sohm
 *
 * Parses rss and rdf XML, extracting titles, descriptions and links.
 */

package com.msohm.rssPush;

import org.dom4j.*;
import java.util.*;

public class RSSParser
{
  private Document xmlDoc;

  public RSSParser()
  {
  }

  //Set the XML string that will be parsed.
  public void setXMLString(String xmlString)
  {
    try
    {

      xmlDoc = DocumentHelper.parseText(xmlString);
    } catch (Exception e)
    {
      throw new RuntimeException("Error parsing string: " + e.getMessage());
    }
  }

  //Get the current XML string.
  public String getXmlString()
  {
    return xmlDoc.asXML();
  }
```

```
//Parses the XML data, extracting title, link and description.
public Vector parseXML(int articleLimit)
{
  Node node;
  Document itemDoc;
  String itemInfo[] = new String[3];
  Vector xmlData = new Vector(10);
  List list = null;
  boolean rdfNode;
  String nodeName;
  String nodePath;

  node = xmlDoc.selectSingleNode("/*");
  nodeName = node.getName();

  //Different format for rdf vs rss pages.
  if (nodeName.compareToIgnoreCase("rdf") == 0)
  {
    //Get the title, description and link for the web site
    rdfNode = true;

    node = xmlDoc.selectSingleNode("/rdf:RDF/*[name()='channel']/*[name()='title']");

    if (node != null)
      itemInfo[0] = node.getText();
    else
      itemInfo[0] = "";

    node = xmlDoc.selectSingleNode("/rdf:RDF/*[name()='channel']/*[name()='link']");

    if (node != null)
      itemInfo[1] = node.getText();
    else
      itemInfo[1] = "";

    node = xmlDoc.selectSingleNode("/rdf:RDF/*[name()='channel']/*[name()='description']");

    if (node != null)
      itemInfo[2] = node.getText();
    else
      itemInfo[2] = "";
  }
  else
  {
    //Get the title, description and link for the web site

    rdfNode = false;

    node = xmlDoc.selectSingleNode("/*/channel/title");

    if (node != null)
      itemInfo[0] = node.getText();
    else
      itemInfo[0] = "";

    node = xmlDoc.selectSingleNode("/*/channel/link");

    if (node != null)
      itemInfo[1] = node.getText();
    else
      itemInfo[1] = "";
```

```
  node = xmlDoc.selectSingleNode("/*/channel/description");

  if (node != null)
    itemInfo[2] = node.getText();
  else
    itemInfo[2] = "";
}

xmlData.add(itemInfo);

//Re-initialize string array so Vector doesn't get confused.
itemInfo = new String[3];


//Get the list of news articles
if (rdfNode)
{
  //RDF Format
  list = xmlDoc.selectNodes("/rdf:RDF/*[name()='item']");
}
else
{
  //RSS Format
  list = xmlDoc.selectNodes("/*/channel//item");
}

int count = 0;

for (Iterator iter = list.iterator(); iter.hasNext();)
{
  //Check to see if we have reached the subscribers max article limit.
  if ((articleLimit > 0) && (count == articleLimit))
    break;

  node = (Node)iter.next();

  try
  {
    itemDoc = DocumentHelper.parseText(node.asXML());

  } catch (Exception e)
  {
    throw new RuntimeException("Error parsing string: " + e.getMessage());
  }

  if (rdfNode)
  {
    node = itemDoc.selectSingleNode("/*[name()='item']/*[name()='title']");

    if (node != null)
      itemInfo[0] = node.getText();
    else
      itemInfo[0] = "";

    node = itemDoc.selectSingleNode("/*[name()='item']/*[name()='link']");

    if (node != null)
      itemInfo[1] = node.getText();
    else
      itemInfo[1] = "";
```

```
        node = itemDoc.selectSingleNode("/*[name()='item']/*[name()='description']");

        if (node != null)
          itemInfo[2] = node.getText();
      else
        itemInfo[2] = "";
    }
    else
    {
      node = itemDoc.selectSingleNode("/item/title");

      if (node != null)
        itemInfo[0] = node.getText();
      else
        itemInfo[0] = "";

      node = itemDoc.selectSingleNode("/item/link");

      if (node != null)
        itemInfo[1] = node.getText();
      else
        itemInfo[1] = "";

      node = itemDoc.selectSingleNode("/item/description");

      if (node != null)
        itemInfo[2] = node.getText();
      else
        itemInfo[2] = "";
    }

    xmlData.add(itemInfo);

    //Re-initialize string array so Vector doesn't get confused.
    itemInfo = new String[3];

    count++;
   }
   return xmlData;
  }
}
```

Filename: RSSPusher.java

```
/*
 * RSSPusher.java
 *
 * Author:  Mark Sohm
 *
 * Pushes specified URL to a BlackBerry user via an MDS Channel Push.
 */


package com.msohm.rssPush;

import java.net.*;
import java.io.*;
import java.util.*;

/**
 * Pushes the supplied information to the specified user on the specified
```

```
 * BlackBerry Mobile Data Server.
 *
 */

public class RSSPusher
{
  //Constants
  public static final String CHANNEL = "Browser-Channel";
  public static final String MESSAGE = "Browser-Message";
  public static final String CONTENT = "Browser-Content";
  public static final String CHANNEL_DELETE = "Browser-Channel-Delete";

  private String besHostname;
  private int besPort;
  private String email;
  private String pushURLString;
  private String pushType;
  private String pushTitle;
  private String unreadIconURL;
  private String readIconURL;

  public RSSPusher()
  {
  }

  //Define get/set methods.

  public void setBesHostname(String rssBesHostname)
  {
    besHostname = rssBesHostname;
  }

  public String getBesHostname()
  {
    return besHostname;
  }

  public void setBesPort(int rssBesPort)
  {
    besPort = rssBesPort;
  }

  public int getBesPort()
  {
    return besPort;
  }

  public void setEmail(String rssEmail)
  {
    email = rssEmail;
  }

  public String getEmail()
  {
    return email;
  }

  public void setPushURLString(String rssPushURL)
  {
    pushURLString = rssPushURL;
  }
```

```
public String getPushURLString()
{
  return pushURLString;
}

public void setPushType(String rssPushType)
{
  pushType = rssPushType;
}

public String getPushType()
{
  return pushType;
}

public void setPushTitle(String rssPushTitle)
{
  pushTitle = rssPushTitle;
}

public String getPushTitle()
{
  return pushTitle;
}

public void setUnreadIconURL(String rssUnreadIconURL)
{
  unreadIconURL = rssUnreadIconURL;
}

public String getUnreadIconURL()
{
  return unreadIconURL;
}

public void setReadIconURL(String rssReadIconURL)
{
  readIconURL = rssReadIconURL;
}

public String getReadIconURL()
{
  return readIconURL;
}
//End define get/set methods.

/**
 * Pushes the page to the BlackBerry user.  Returns true if it was delivered
 * to the MDS Server, false if it was not.
 */

public int pushPage()
{

  /*
   * Two HttpURLConnections are used.  One connects to the URL of the
   * page to push and reads the page from there.  The other
   * connects to the MDS Server and writes the page to it.
   */
```

```
boolean pushSucceeded = true;
HttpURLConnection pushConn, besConn;
URL pushUrl, besUrl;
int rescode = 0;

try
{
  /*
   * Push listener thread on the device listens to port 7874 for
   * pushes from the BlackBerry Mobile Data Server.
   */

  besUrl = new URL("http", besHostname, besPort, "/push?DESTINATION=" + email +
    "&PORT=7874&REQUESTURI=/");

  besConn = (HttpURLConnection)besUrl.openConnection();

  // Set header properties for the push.
  besConn.setRequestProperty("Content-Location", pushURLString);
  besConn.setRequestProperty("X-RIM-Push-Title", pushTitle);
  besConn.setRequestProperty("X-RIM-Push-Type", pushType);

  if (pushType.equals(CHANNEL) || pushType.equals(CHANNEL_DELETE))
  {
    besConn.setRequestProperty("X-RIM-Push-Channel-ID", pushURLString);

    if (pushType.equals(CHANNEL))
    {
      besConn.setRequestProperty("X-RIM-Push-UnRead-Icon-URL", unreadIconURL);
      besConn.setRequestProperty("X-RIM-Push-Read-Icon-URL", readIconURL);
    }
  }

  try
  {
    besConn.setRequestMethod("POST");
  } catch (ProtocolException e)
  {
    throw new RuntimeException("Exception calling setRequestMethod(): " + e.getMessage());
  }

  besConn.setAllowUserInteraction(false);
  besConn.setDoInput(true);
  besConn.setDoOutput(true);

  if (!pushType.equals(CHANNEL_DELETE))
  {
    // No need to connect to the pushed page for a delete.
    try
    {
      pushUrl = new URL(pushURLString);
    } catch (MalformedURLException e)
    {
      throw new RuntimeException("Invalid push URL: " + e.getMessage());
    }

    pushConn = (HttpURLConnection)pushUrl.openConnection();
    pushConn.setAllowUserInteraction(false);
    pushConn.setDoInput(true);
    pushConn.setDoOutput(false);
    pushConn.setRequestMethod("GET");
```

```
      pushConn.connect();

      /*
       * Read the header properties from the push connection and write
       * them to the BlackBerry Mobile Data Server connection.
       */

      String name, value;

      for (int i = 0; true; i++) {
        name = pushConn.getHeaderFieldKey(i);
        value = pushConn.getHeaderField(i);

        if ((name ==null) && (value == null)) { break; }
        if ((name ==null) || (value == null)) { continue; }
        if (name.equals("X-RIM-Push-Type")) { continue; }
        if (name.equals("Transfer-Encoding")) { continue; }

        besConn.setRequestProperty(name, value);
      }

      /*
       * Read the content from the push connection and write it to
       * the BlackBerry Mobile Data connection.
       */

      InputStream ins = pushConn.getInputStream();
      OutputStream outs = besConn.getOutputStream();
      copyStreams(ins, outs);
    }

    besConn.connect();
    rescode = besConn.getResponseCode();
  } catch (IOException e)
  {
    throw new RuntimeException("Unable to push page:" + e.toString());
  }

  return rescode;
 }

 /**
  * Reads data from the input stream and copies it to the output stream
  */

 private static void copyStreams(InputStream ins, OutputStream outs) throws IOException
 {
   int maxRead = 1024;
   byte [] buffer = new byte[1024];
   int bytesRead;

   for(;;)
   {
     bytesRead = ins.read(buffer);

     if (bytesRead <= 0) {break;}
       outs.write(buffer, 0, bytesRead);
   }
 }
}
```

Filename: RSSPush.java

```
/*
 * RSSPush.java
 *
 * Retrieves, parses and pushes rss and rdf feeds to BlackBerry users.
 *
 * Author:  Mark Sohm
 */

package com.msohm.rssPush;

import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.*;
import java.sql.ResultSet;
import java.util.logging.*;

/**
 * Monitors user selected rss news feeds and pushes updated article information
 * to a users BlackBerry.
 */

public class RSSPush extends Thread
{
  public static void main(String[] args)
  {
    RSSPush rssPush = new RSSPush();
    rssPush.start();
  }

  public void run()
  {

    //Shared variables
    RSSDBConnector dbConnector = new RSSDBConnector();
    RSSDBConnector dbConnector2 = new RSSDBConnector();
    RSSPusher pusher = new RSSPusher();
    RSSParser parser = new RSSParser();
    Logger logger = null;
    java.sql.ResultSet rs;
    java.sql.ResultSet rs2;
    boolean dbFailed, moreRecords;
    int count;
    int updateFailed;
    String query = null;

    //Update rss sites variables
    StringBuffer sb = new StringBuffer();
    URL rssURL = null;
    HttpURLConnection rssConnection = null;
    boolean urlFailed;
    String rssHash;
    String rssPage = null;
    long pageLastModified = 0;
    String eTag = "";
    boolean urlNotFound;
    InputStream ins = null;
    BufferedReader rdr = null;
```

```
//Parse variables
boolean moreSites;
StringBuffer header = new StringBuffer();
StringBuffer body = new StringBuffer();
boolean parseFailed;
String usersPage = new String("");
String xmlElements[];
Vector parsedXMLData = new Vector();

//Push variables
String mdsHosts[] = new String[10];
int mdsPorts[] = new int[10];
int hostCount = 0;
boolean pushDone;
int responce;

//Initialize the log file
try
{
  //Create an appending file handler
  boolean append = true;
  FileHandler handler = new FileHandler("C:\\YourPath\\rssFeed.xml", append);

  //Add to the desired logger
  logger = Logger.getLogger("com.msohm.rssPush");
  logger.addHandler(handler);
} catch (IOException e)
{
  System.out.println("Error initializing logger: " + e.toString());
  System.out.println("Exiting program.");
  System.exit(0);
}

//Loop continuously.
for(;;)
{
  dbFailed = true;
  try
  {
    //Initialize the two DBConnectors.
    dbConnector.dbConnect("DataBaseName", "databaseServerIP", "DatabaseLogin",
      "DatabasePassword");
    dbConnector2.dbConnect("DatabaseName", "databaseServerIP", "DatabaseLogin",
      "DatabasePassword");

    dbFailed = false;
  } catch (Exception e)
  {
    logger.log(Level.SEVERE, "Error establishing database connection!: " + e.toString());
  }

  //Start check rss pages for updates.

  if (!dbFailed)
  {
    logger.log(Level.WARNING, "Starting check rss pages for updates.");

    try
    {
      //Retrieve all rss sites.
```

```
rs = dbConnector.select("SELECT SiteNo, URL, Hash, LastModified, ETag FROM rssSites");

moreRecords = rs.next();

//Loop through all rss sites.
while(moreRecords)
{
  urlFailed = true;

  try
  {
    rssURL = new URL(rs.getString(2));
    urlFailed = false;
  } catch (MalformedURLException e)
  {
    //Could not open URL.
    urlFailed = true;
    logger.log(Level.WARNING, "\nFailed to open URL: " + rs.getString(2));
  }

  //Ensure that the URL is valid.
  if(!urlFailed)
  {
    try
    {
      rssConnection = (HttpURLConnection)rssURL.openConnection();
      //Send the date the page was last updated along with the ETag to allow
      //for support for 304 response codes.
      rssConnection.setRequestProperty("If-None-Match", rs.getString(5));
      rssConnection.setIfModifiedSince(rs.getLong(4));
      rssConnection.setAllowUserInteraction(false);
      rssConnection.setDoInput(true);
      rssConnection.setDoOutput(false);
      rssConnection.setRequestMethod("GET");
      rssConnection.connect();

      if (rssConnection.getResponseCode() == 200)
      {
        //New page that was successfully retrieved.

        ins = rssConnection.getInputStream();

        //Get the last updated date anad ETag for the page.
        pageLastModified = rssConnection.getHeaderFieldDate("Last-Modified", 0);
        eTag = rssConnection.getHeaderField("ETag");
        urlNotFound = false;
      }
      else
      {
        urlNotFound = true;
        logger.log(Level.WARNING, "Page not available for site " + rs.getString(1) +
          ".  Response code: " + rssConnection.getResponseCode());
      }
    } catch (Exception e)
    {
      //File was not found.
      logger.log(Level.WARNING, "Unable connect to URL: " + rs.getString(2));
      urlNotFound = true;
    }

    if (urlNotFound)
```

```
{
  rssPage = "";
}
else
{
  //Read in the html page.
  try
  {
    rdr = new BufferedReader(new InputStreamReader(ins));

    String line;

    sb.delete(0, sb.length());

    while ((line = rdr.readLine()) != null)
    {
      sb.append(line);
      sb.append("\n");
    }

    rssPage = sb.toString();
    rssPage = rssPage.replaceAll("'", "''");

    ins.close();
    rdr.close();

  } catch (Exception e)
  {
    urlNotFound = true;
    logger.log(Level.WARNING, "Error reading input for URL: " + rs.getString(2));
  }
}

//Close the connection.
try
{
  rssConnection.disconnect();
} catch (Exception e)
{
  logger.log(Level.WARNING, "Exception disconnecting from site: " + rs.getString(2)
    + " Exception: " + e.toString());
}

if (rssPage.hashCode() != rs.getInt(3) && !urlNotFound)
{
  //The page has been changed since last check.
  //Store the new page in the database.

  try
  {
    rssPage = rssPage.replaceAll("'", "''");

    query = "Update rssSites SET Page='" + rssPage + "', Hash='" + rssPage.hashCode()
      + "', LastUpdated=NOW(), LastModified='" + pageLastModified + "', ETag='" + eTag
      + "' WHERE SiteNo='" + rs.getInt(1) + "';";

    updateFailed = dbConnector.insertOrUpdate(query);

    if (updateFailed == 0)
    {
      //Update failed
```

```
            logger.log(Level.WARNING, "\n0 rows updated: " + query);
          }
          else
          {
            logger.log(Level.INFO, "Updated site number " + rs.getString(1));
          }
        } catch (Exception e)
        {
          //Update query failed
          logger.log(Level.WARNING, "Update rss page failed:" + e.toString());
        }
      }
    }

    moreRecords = rs.next();
  }

  rs.close();
 } catch (Exception e)
 {
  //Select query failed
  logger.log(Level.WARNING, "Exception during rss page update process: " + e.toString());
 }

 logger.log(Level.WARNING, "Completed check rss pages for updates.");
}

//End check rss pages for updates.


//Start update personal pages.

if (!dbFailed)
{
 logger.log(Level.WARNING, "Starting update personal pages.");

 try
 {

  //Select the users who have subscribed to pages that have updated
  //since their own last update.

  rs = dbConnector.select("SELECT DISTINCT Users.UserNo, Users.Hash FROM Users,
    Subscriptions, rssSites WHERE Users.UserNo=Subscriptions.UserNo AND
    Subscriptions.SiteNo=rssSites.SiteNo AND Users.LastUpdated<=rssSites.LastUpdated;");
  moreRecords = rs.next();

  //Loop through all records.
  while(moreRecords)
  {
    //Clear contents from previous page.
    header.delete(0, header.length());
    body.delete(0, body.length());

    //Begin creation of the users customized HTML page.
    header.append("<html><head><title>BlackBerry RSS News Feed</title></head><body>");

    try
    {
      //Retrieve a list of sites and site settings the user has subscribed to.
      rs2 = dbConnector2.select("SELECT Subscriptions.NoOfArticles, rssSites.SiteNo,
```

```
  rssSites.Page, Subscriptions.ShowDescription FROM Subscriptions, rssSites
  WHERE Subscriptions.UserNo='" + rs.getInt(1) + "' AND
  Subscriptions.SiteNo=rssSites.SiteNo ORDER BY Subscriptions.ListOrder;");

moreSites = rs2.next();

//Loop through all sites.
while(moreSites)
{
  parseFailed = true;

  try
  {
    parser.setXMLString(rs2.getString(3));
    parsedXMLData = parser.parseXML(rs2.getInt(1));
    parseFailed = false;

  } catch (Exception e)
  {
    //RSS feed has invalid XML data.
    logger.log(Level.WARNING, "Unable to parse XML for site number " + rs2.getInt(2) +
      ". : " + e.toString());
  }

  //If the RSS feed was processed.
  if (!parseFailed)
  {
    int numOfElements = parsedXMLData.size();

    boolean newSite = true;

    for (count = 0; count < numOfElements; count++)
    {
      //0 = Title, 1 = Link, 2 = Description

      xmlElements = (String[])parsedXMLData.get(count);

      //Is this a new web site?
      if (newSite)
      {
        //Add heading information for the new web site.
        newSite = false;
        header.append("<h3><a href=\"#" + rs2.getInt(2) + "\">" +
          xmlElements[0].replaceAll("'", "'") + "</a></h3>");
        header.append("<hr/>");
        body.append("<hr/>");
        body.append("<a name=\"" + rs2.getInt(2) + "\"><!-- --></a><a href=\"" +
          xmlElements[1].replaceAll("'", "'") + "\"><h4>" +
          xmlElements[0].replaceAll("'", "'") + "</h4></a>");
      }
      else
      {
        //Add the description if the users has chosen to receive them.
        if(rs2.getInt(4) == 1)
        {
          body.append("<a href=\"" + xmlElements[1].replaceAll("'", "'") + "\">" +
            xmlElements[0].replaceAll("'", "'") + "</a><br><font size='-2'>" +
            xmlElements[2].replaceAll("'", "'") + "</font><br><br>");
        }
        else
        {
```

```
                body.append("<a href=\"" + xmlElements[1].replaceAll("'", "'") + "\">" +
                  xmlElements[0].replaceAll("'", "'") + "</a><br><br>");
              }
            }
          }
        }

        moreSites = rs2.next();
      }

      rs2.close();

    } catch (Exception e)
    {
      //Select query failed
      logger.log(Level.WARNING, "Exception during subscription select for user " +
        rs.getInt(1) + " : " + e.toString());
    }

    //Append the closing tags to the user's html page.
    body.append("<hr/>");
    body.append("Questions?  Email <a href=you@yourhost.com>you@yourhost.com</a>.");
    body.append("</body></html>");
    header.append(body.toString());

    usersPage = header.toString();
    //Duplicate all ' for the SQL statement.
    usersPage = usersPage.replaceAll("'", "''");

    if (usersPage.hashCode() != rs.getInt(2))
    {
      query = "UPDATE Users SET Page='" + usersPage + "',LastUpdated=NOW(), Hash='" +
        usersPage.hashCode() +"' WHERE UserNo='" + rs.getInt(1) + "';";

      try
      {
        updateFailed = dbConnector2.insertOrUpdate(query);

        if (updateFailed == 0)
        {
          //Update failed
          logger.log(Level.WARNING, "\n0 rows updated during user page update for user " +
            rs.getInt(1) + ":" + query);
        }
        else
        {
          logger.log(Level.INFO, "Updated personal page for user " + rs.getInt(1));
        }
      } catch (Exception e)
      {
        //Update query failed
        logger.log(Level.WARNING, "Exception during update user page: " + e.toString());
      }
    }

    moreRecords = rs.next();
  }

  rs.close();

} catch (Exception e)
```

```
  {
    //Select query failed
    logger.log(Level.WARNING, "Exception during user update process: " + e.toString());
  }

  logger.log(Level.WARNING, "Completed check rss pages for updates.");
}

//End update personal pages if subscribed news pages have changed

//Start push updates to subscribers.

if (!dbFailed)
{
  logger.log(Level.WARNING, "Starting push updates to subscribers.");

  //Setup standard push variables
  pusher.setPushTitle("RSS News");
  pusher.setUnreadIconURL("http://YourWebServerURL/rssPush/images/unread.png");
  pusher.setReadIconURL("http://YourWebServerURL/rssPush/images/read.png");

  //Get MDS Server list.
  try
  {
    rs = dbConnector.select("SELECT Hostname, Port FROM MDSServers ORDER BY MDSServerNo;");

    moreRecords = true;
    moreRecords = rs.next();

    hostCount = 0;

    while(moreRecords)
    {

      mdsHosts[hostCount] = rs.getString(1);
      mdsPorts[hostCount] = rs.getInt(2);

      moreRecords = rs.next();
      hostCount++;

    }

    rs.close();
  } catch (Exception e)
  {
    logger.log(Level.WARNING, "Select MDS Server list failed: " + e.toString());
  }

  //Get a list of users who's page has been updated since their last push.
  try
  {
    rs = dbConnector.select("SELECT UserNo, Email, SendDelete FROM Users
      WHERE LastPush<LastUpdated OR SendDelete='1';");

    moreRecords = true;
    moreRecords = rs.next();

    while(moreRecords)
    {

      pusher.setEmail(rs.getString(2));
```

```
pusher.setPushURLString("http://YourWebServerURL/rssPush/showpage.php?userNo=" +
  rs.getString(1));

count = 0;
pushDone = false;

//Push a channel delete if the user has unsubscribed from the service
//to remove the icon from their ribbon.
if (rs.getInt(3) == 1)
{
  pusher.setPushType(RSSPusher.CHANNEL_DELETE);
}
else
{
  pusher.setPushType(RSSPusher.CHANNEL);
}

//Push to each MDS Server in the organization until we
//push to the one the user is on.
while(!pushDone && count < hostCount)
{
  pusher.setBesHostname(mdsHosts[count]);
  pusher.setBesPort(mdsPorts[count]);
  responce = 0;

  try
  {
    responce = pusher.pushPage();

  }catch(Exception e)
  {
  logger.log(Level.INFO, "Exception during push try # " + count + " to " +
    pusher.getBesHostname() +  ":" + pusher.getBesPort() + " for user: " +
    rs.getString(2) + ". Exception: " + e.toString());
  }

  if (responce == 200)
  {
    //Push was successfull.
    pushDone = true;

    try
    {
      if (rs.getInt(3) == 1)
      {
        //Successfully pushed a channel delete.  Reset the flag
        //so we don't send it again.
        dbConnector2.insertOrUpdate("UPDATE Users SET SendDelete='0' WHERE UserNo='" +
          rs.getString(1) + "'");
      }
      else
      {
        //Update the users's last push date/time.
        dbConnector2.insertOrUpdate("UPDATE Users SET LastPush=NOW() WHERE UserNo='" +
          rs.getString(1) + "'");
      }

    } catch (Exception e)
    {
      logger.log(Level.WARNING, "Update LastPush/SendDelete Failed: " + e.toString());
    }
```

```
            }

            logger.log(Level.INFO, "Push try # " + count + " to " + pusher.getBesHostname() +
              ":" + pusher.getBesPort() +  " Responce=: " + responce + ". For user: " +
              rs.getString(2));

            count++;
           }

           moreRecords = rs.next();

         }

         rs.close();
        } catch (Exception e)
        {
         logger.log(Level.WARNING, "Select users failed (push updates): " + e.toString());
        }

        logger.log(Level.WARNING, "Completed push updates to subscribers.");
      }


      //End push updates to subscribers.


      //Disconnect from the database server since we won't need the connection
      //for another 45 minutes.
      try
      {
        dbConnector.dbDisconnect();
        dbConnector2.dbDisconnect();

        dbFailed = false;
      } catch (Exception e)
      {
        logger.log(Level.SEVERE, "Error closing database connection!: " +
          e.toString());
      }


      //Sleep for 45 minutes before starting the process again.
      try
      {
        sleep(2700000);

      } catch (Exception e)
       {
        logger.log(Level.SEVERE, "I couldn't sleep! : " + e.toString());
        System.exit(0);
      }
    }
  }
}
```