Virtual Machinery
B+Tree Implementation – Demos

Table of Contents

Demos

Introduction

There are two types of demo available –

1. Demos that can be downloaded from the website
2. Demos that come with the product

All of the demos that are downloadable from the website are also included with full code in the product. Most of the demos on the website come with code but use cut down versions of the B+Tree libraries.

These demos are designed to demonstrate the power of the Virtual Machinery BTree in a number of different ways.

Website Demos

There a number of demos in the demo packages that you can download from our website. These demos are intended as a 'taster' of what Virtual Machinery's B+Tree can do.

These are restricted demos as they generally only handle the Read-Only aspects of the B+Tree implementation. The versions of the B+Tree read only libraries provided with these applications are 'cut-down' versions of the originals. In the case of the iOS library it is only compiled for the Simulator.

There are 4 demo packages available from the B+Tree demos download page. These packages reflect the main platforms that Virtual Machinery's B+Tree implementation is available on.

1. Java B+Tree
2. Java J2ME B+Tree
3. Android B+Tree
4. iOS B+Tree

There is also a compendium package that includes all of the demos.

Product demos

The Base License of the Virtual Machinery B+Tree implementation includes full versions of these demos – these use the standard B+Tree libraries and source code is included for all of the demos.

Standard datasets

To illustrate the power and platform independence of our B+Tree implementation two standard datasets are included. The same datasets are used for multiple platforms. In other words you can create a dataset on one platform and use it on another. The datasets provided are –

- VMPHONETEST
- MEDIA

Each dataset is made up of an index (.IND) and a data (.DAT) file. In the case of the Android example the dataset names have had to be changed as a consequence of the Android file handling mechanism. The Android filenames have been changed from XXXX.DAT to XXXX_DAT.WMV and from XXXX.IND to XXXX_IND.WMV.

Standard Read only Demos

There are two standard read only demos that are available from the Web and are also included in the Base License package. In most cases the code for these applications is provided (or else illustrative code samples are included in this document). Each of the demos is implemented on each of the platforms. In the case of the mobile environments the examples are designed to be run in the simulator/development environments for the platforms. The mobile demos have all been tested in a limited number of 'real' environments and in the case of the J2ME and Android demos (both the web versions, and the versions that come with the full product) these can be deployed on real devices. In the case of the web versions of the iOS demos these can only be deployed in the XCode simulator as the library has only been compiled for the simulator. The versions of the iOS demos that come as part of the full product run using the full library and as such can be deployed to real devices.

There are some differences in the implementations for different platforms but the principles remain the same.

The Address demo

This demo allows you to use a phone number to look up a name and address in a B+Tree which contains details for 10000 Phone accounts. There are a number of options available on the demo that illustrate the use of Virtual Machinery's B+Tree library –

1. Enter a number and retrieve an address
2. Enter a number of addresses to retrieve – this will create a sequence of random phone numbers and retrieved the address details for each. At the end of the test the number of milliseconds that it took to retrieve the details is shown.
3. Switch cacheing off or on. This shows the effect of using the Virtual Machinery B+Tree's cacheing features on the test in 2. Above

The Media demo

This demo shows the ability of the B+Tree to store non-textual items. Three screens are shown in the demo that illustratefeatures of the Virtual Machinery B+Tree implementation:-

1. Storage of text data
2. Storage of an image
3. Storage of sound

In the case of the sound and image the sizes of the data are greater than the maximum page size allowed by the Virtual Machinery B+Tree implementation. The demos provide an implementation of a strategy to overcome this limitation.

Java SE BTree Demos

These demos are designed to be run in a standard BTree environment (Java SE or J2EE). They have been packaged for use with the Eclipse development environment and are therefore Eclipse projects. It should be easy to use the code in any environment. The code is in the src directory and the B+Tree library is in the root directory of the project. The code is found in the package com.virtualmachinery.btreedemo.

The following classes are provided in the website demo and in the base license of the product –

- BTreeAddressDemo.java
- BTreeMediaDemo.java

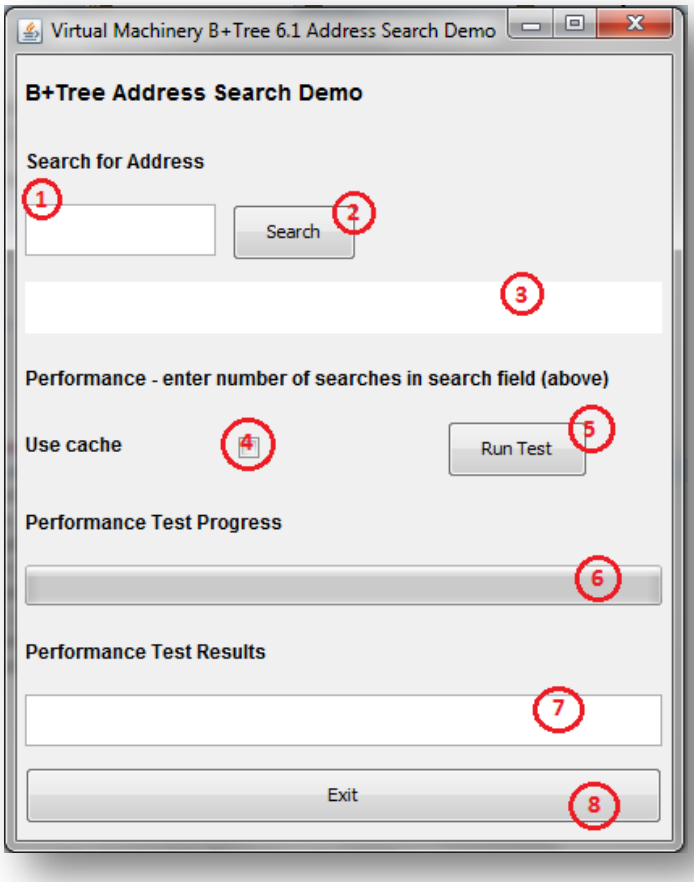
And the following additional classes are included with base license of the product –

- BTreeDemo.java
- BtreeEntryDemo.java

Full source code is supplied for the demos in both cases. There are some differences in the code supplied between the web demo and the demo supplied with the base license but the principles are the same.

The BTreeAddressDemo demo (Web and Base License)

You can start the demo by double clicking on the BTreeAddressDemo.jar in the …\demo\javabtree directory in the distribution or by running the BTreeAddressDemo main method in an IDE. In either case you will see the following screen –



The various numbered parts of the screen are described below –

1. The address search entry field.
2. The 'Search' button – used to initiate the search on the B+Tree using the contents of the search entry field
3. The details field which will show the name and address details relating to the number entered in the address search field
4. The 'Use cache' button which switches the cacheing feature of the B+Tree off or on.
5. The 'Run Test' button which initiates the performance test
6. The progress bar that illustrates the current progress of the test
7. The 'Performance Test Results' field which will show the results of running the performance test.
8. The 'Exit' button that is used to exit the test

The address search: enter a value in the address search entry field. There are 10000 entries in the B+Tree starting at 1111111 so you can enter a number between 1111111 and 1121110 here. Then

press the 'Search' button. The details associated with the search value will be displayed in the details field.

The performance test (cache off): enter the number of addresses that you want to search for. This will generate that number of random searches. Leave the 'Use cache' check box empty. You should choose a small number of entries to begin with (say 1000) until you have some sense of the power of your machine. The time taken to carry out the searches will be displayed in the 'Performance Test Results' field.

The performance test (cache on): enter the number of addresses that you want to search for. This will generate that number of random searches. Check the 'Use cache' check box. You should choose a small nummber of entries to begin with (say 1000) until you have some sense of the power of your machine. The time taken to carry out the searches will be displayed in the 'Performance Test Results' field.

The Code behind the B+Tree Address Demo

Note that any call to a method on the BTree must be prepared to handle the occurrence of a BTreeException –

Opening the tree: The BTree addressBTree is opened in the initialize() window using the following code :

```
try {
	addressBtree = BTreeRO.openExistingTree("VMPHONETEST");
} catch (BTreeException e) {
	e.printStackTrace();
}
```

Searching for an entry: To search for a key in the BTree the get method is used. Since the key is passed as a String the get method that takes a String as a parameter is used. This also returns the result as a String.

```
try {
	result = addressBtree.get(key);
} catch (BTreeException e) {
	System.out.println("Error fetching data from BTree in search "+e);
	e.printStackTrace();
}
```

Switching the cache on and off: When the status of the cache (i.e. whether it is on or off) is changed the BTree is reopened with cacheing either enabled or disabled. Since there are 3 index pages and 132 data pages in the tree the tree is opened with all of these pages cached if the cache has been switched on.

```
try {
	if (cache) {
		addressBtree = BTreeRO.openExistingTree("VMPHONETEST",3,132);
	} else {
		addressBtree = BTreeRO.openExistingTree("VMPHONETEST");
	}
```

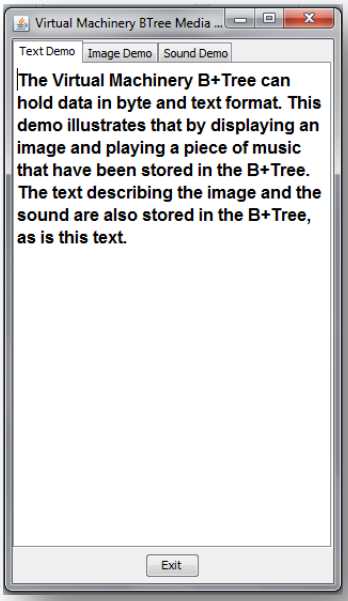
```
} catch (BTreeException e) {
	e.printStackTrace();
}
```

Running a performance test: The performance test finds the number of iterations to run from the number entry field on the dialog. For each iteration it creates a valid random number to look up in the B+Tree and updates the progress bar to show the proportion of entries that it has looked up.

```
int count=0;
int entries = Integer.parseInt(numberEntry.getText().toString());
Random random = new Random();
int randNo = -1;
long startTime = System.currentTimeMillis();
try {
	for (int i=0;i<entries;i++) {
		randNo = ((int) (random.nextFloat()*10000)) + 1111111;
		if (addressBtree.get(randNo+"") != null) {
			count++;
		}
		if (count != 0) {
			final int progress = (int) ((count/(entries *1.0))*100);
			progressBar.setValue(progress);
		}
	}
} catch (BTreeException e) {
	System.out.println("Error fetching data from BTree in performance test "+e);
	e.printStackTrace();
} finally {
	performanceText.setText(count+" Searches took "+(System.currentTimeMillis() -
startTime)+" ms");
	progressBar.setValue(0);
}
```

The Media demo (Web and Base License)

You can start the demo by double clicking on the BTreeMediaDemo.jar in the …\demo\javabtree directory in the distribution or by running the BTreeMediaDemo main method in an IDE. In either case you will see the following screen –

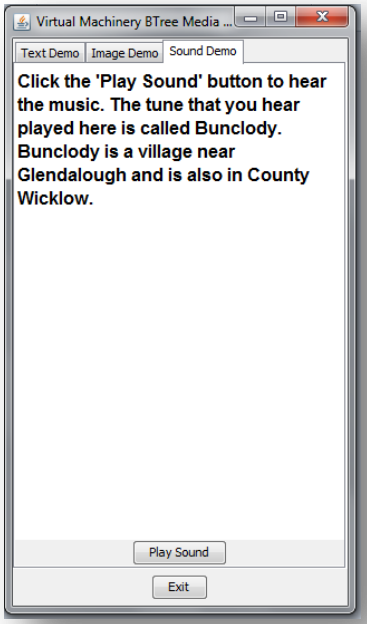


There are three tabs on the demo marked as 'Text Demo', 'Image Demo' and 'Sound Demo'. As you can see above the screen opens on the 'Text Demo' tab which displays some information about the demo. All of the text on the tabs is stored in the B+Tree.

Clicking on the 'Image Demo' tab opens the following screen -

This shows a photographic image that has been stored in the B+Tree.

Clicking on the 'Sound Demo' tab opens the following screen -



If you click the 'Play sound' button you will hear a piece of music that has been stored in the B+Tree.

The code behind the B+Tree Media demo

The source code for the B+Tree Media demo can be found in the BTreeMediaDemo class in the com.virtualmachinery.btreedemo package.

The B+Tree is opened by the following code in the initialize() method –

```
mediaBtree = BTreeRO.openExistingTree("MEDIA");
```

This opens a read only BTree using the default setting i.e. no cache and using the files MEDIA.IND and MEDIA.DAT.

The opening page is the text page so the text is fetched from the BTree using the getTextAreaFromKey method and the string "TEXT_KEY" –

```
public JTextArea getTextAreaFromKey(String key) {
        JTextArea tPane = new JTextArea();
        tPane.setFont(new Font(Font.SANS_SERIF,Font.BOLD,16));
        tPane.setLineWrap(true);
        tPane.setWrapStyleWord(true);
        try {
                tPane.setText(mediaBtree.get(key));
        } catch (BTreeException e) {
                e.printStackTrace();
        }
        return tPane;
}
```

The important method here is the get method called on the mediaBtree (the BTreeRO instance).

If we click on the Image tab the method getImagePanel causes this to be populated with two items – the text (created using using the getTextAreaFromKey method and the string “IMAGE_TEXT” as the key) and the image which is created using the restoreImage method with the string “PICTURE_KEY” as the key.

The restoreImage method gets the bytes for the image from the BTree using the getBytesFromTree method and then reconstitues the image from the bytes –

```
	public BufferedImage restoreImage(String index) throws IOException {
		byte[] pixels = getBytesFromTree(index);
		ByteArrayInputStream bs = new ByteArrayInputStream(pixels);
		return ImageIO.read(ImageIO.createImageInputStream(bs));
	}
```

In this case the getBytesFromTree method uses the “PICTURE_KEY” index to retrieve all the records associated with the index and concatenate them back into a byte array:

```
	public byte[] getBytesFromTree(String index) {
		String iString;
		try {
			iString = (String) mediaBtree.get(index);
			if (iString != null) {
				String[] base = ((String) mediaBtree.get(index)).split("\\^");
				int numRecords = Integer.parseInt(base[0]);
				int pixLength = Integer.parseInt(base[1]);
				byte[] pixels = new byte[pixLength];
				int start = 0;
				for (int i=0;i<numRecords;i++) {
					byte[] thisRec =
mediaBtree.get(BTreeROFactory.getInstance().createRecord(index+"_"+i)).getValue();
					start+=thisRec.length;
				}
				return pixels;
			}
			else {
				return null;
			}
		} catch (BTreeException e) {
			e.printStackTrace();
		}
	}
```

The important lines here are:

```
	iString = (String) mediaBtree.get(index);
```

which retrieves the root record that indicates how many records and bytes there are and the line :

```
	byte[] thisRec =
mediaBtree.get(BTreeROFactory.getInstance().createRecord(index+"_"+i)).getValue();
```

which retrieves each record as an array of bytes which can then be added to the array.

The Sound tab works in a similar way to the image tab and is built by the getSoundPanel() method. It is populated with two items – the text (created using using the getTextAreaFromKey method and the

string “SOUND_TEXT” as the key) and the sound, which is retrived as an array of bytes retrieved using the getBytesFromTree method with the string “SOUNDWAV_KEY” as the key :

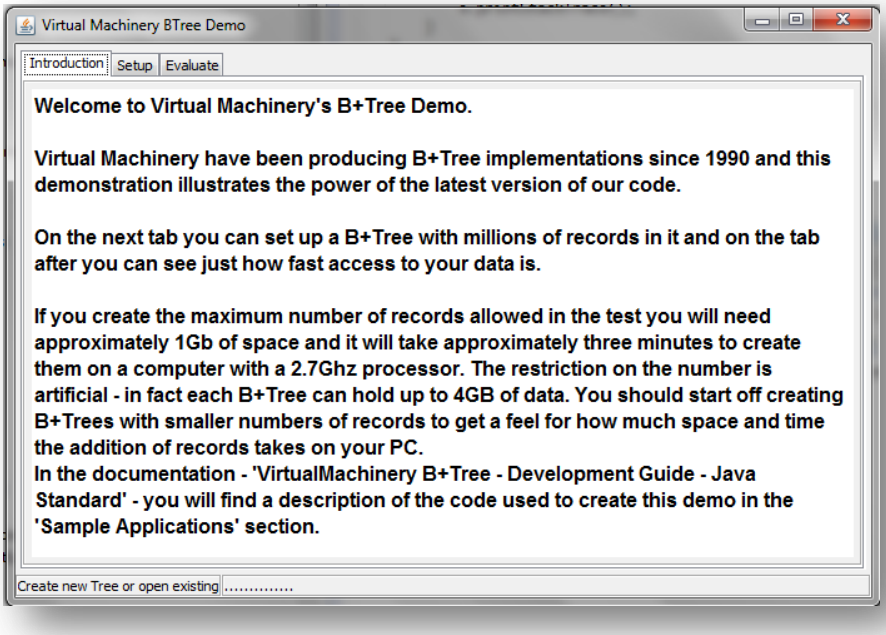
```
        private JPanel getSoundPanel() {
                JPanel soundPanel = new JPanel();
                soundPanel.setLayout(new GridBagLayout());
                JTextArea tPane = this.getTextAreaFromKey("SOUND_TEXT");
                …….
                sound = getBytesFromTree("SOUNDWAV_KEY");
                …….
                return soundPanel;
        }
```

When the ‘Play Sound’ button is clicked the array of bytes is played as a WAV sound by the playSound() method –

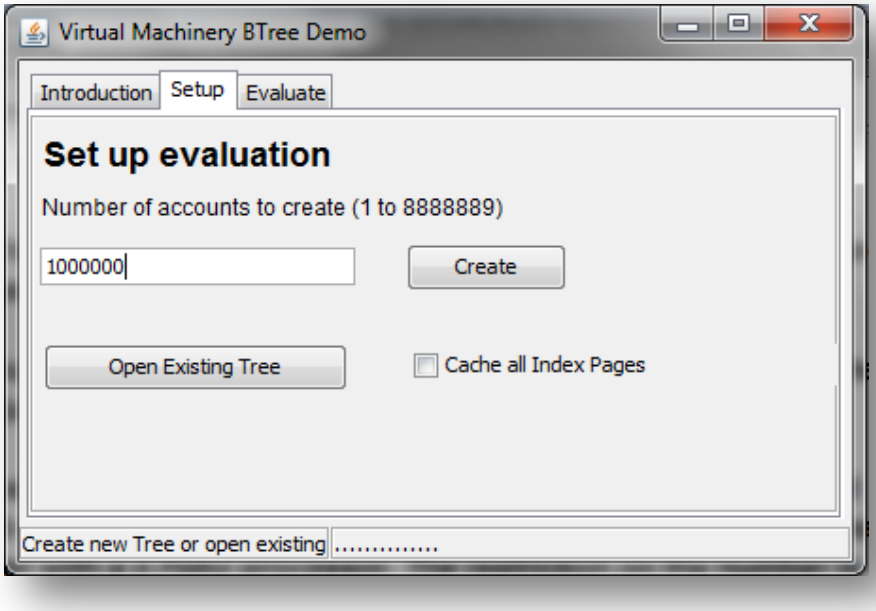
```
private void playSound() {
         ByteArrayInputStream bais = new ByteArrayInputStream(sound);
         AudioInputStream ais = new AudioInputStream(bais, new AudioFormat(44100f, 16, 1,
true, false), sound.length);
                Clip clip;
        try {
                clip = AudioSystem.getClip();
                        clip.open(ais);
                        clip.start();
        } catch (LineUnavailableException e) {
                e.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
     }
```

The BTreeDemo demo (Base License only)

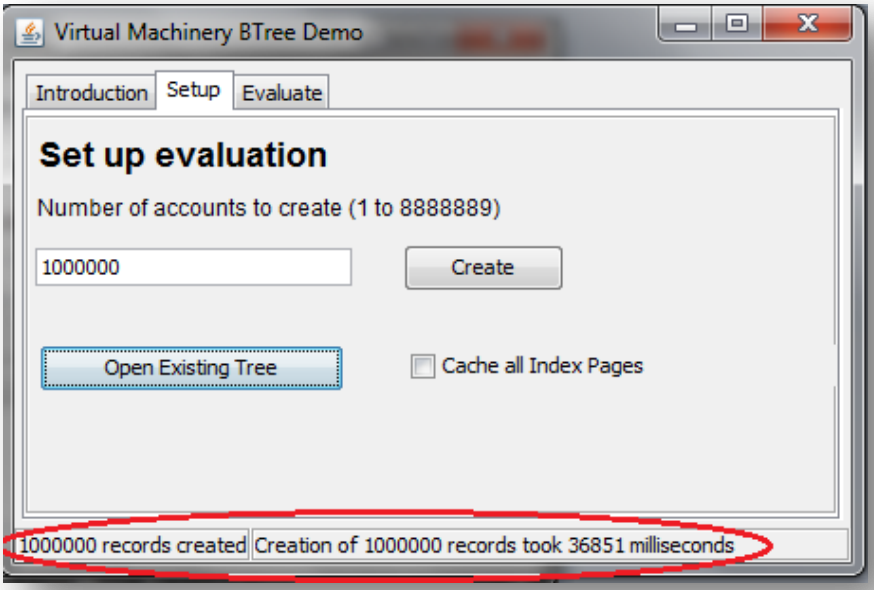
After starting the BTReeDemo demo the opening screen appears as follows -



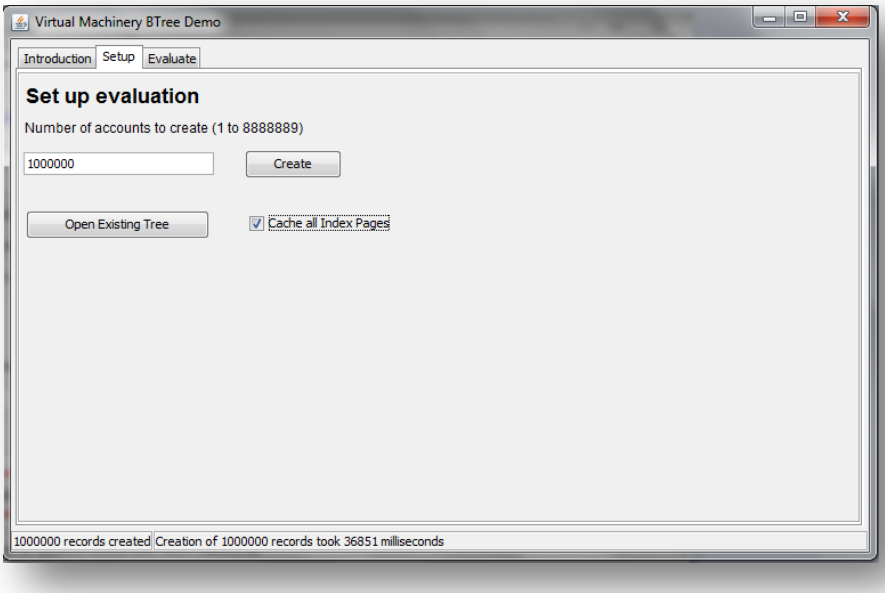
The first thing you have to do is to create a new B+Tree with as many entries as you require. To do this you need to click on the 'Setup' tab and select the number of entries to create –



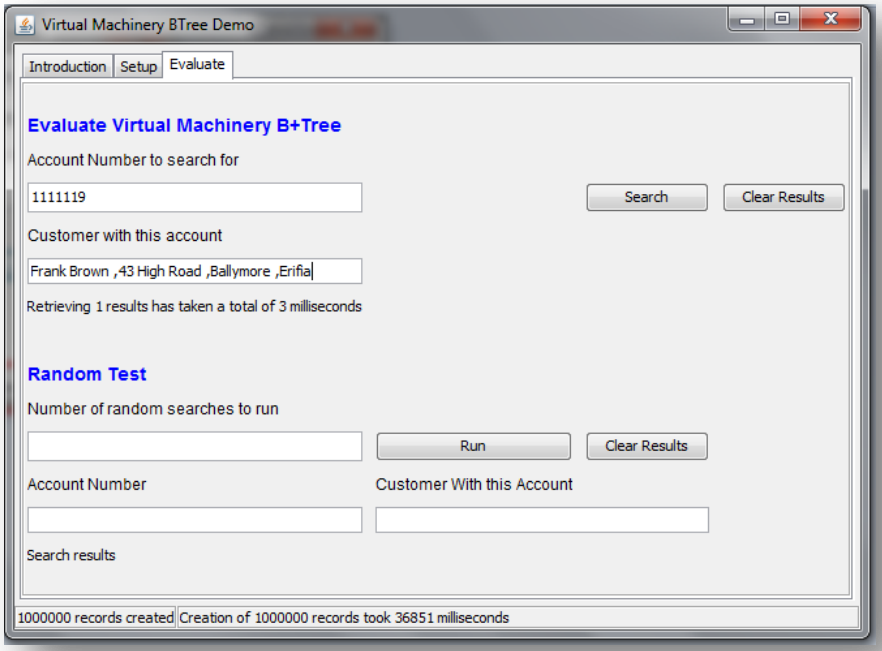
In this case we are creating 100000 entries. After the entries are created the time to create them will be indicated in the status bar at the bottom of the screen –



Note that in this case we have chosen gto open the tree without cacheing the index pages. You can also open the tree with all of the index pages cached – this will greatly increase performance as we will see later -

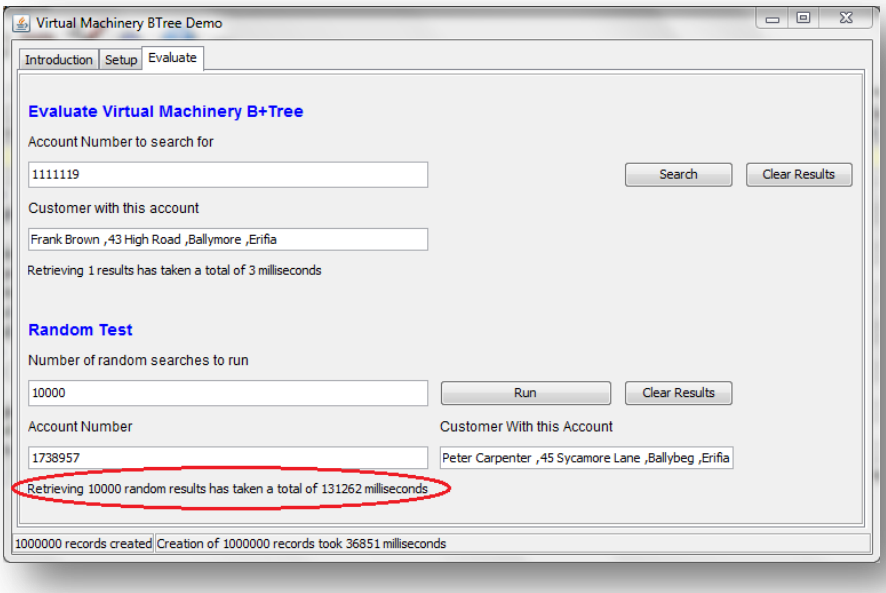


Having created the tree we can now switch to the 'Evaluate' tab to look up the entries in the tree and run performance tests in much the same way as we did in the BTreeAddressDemo above.

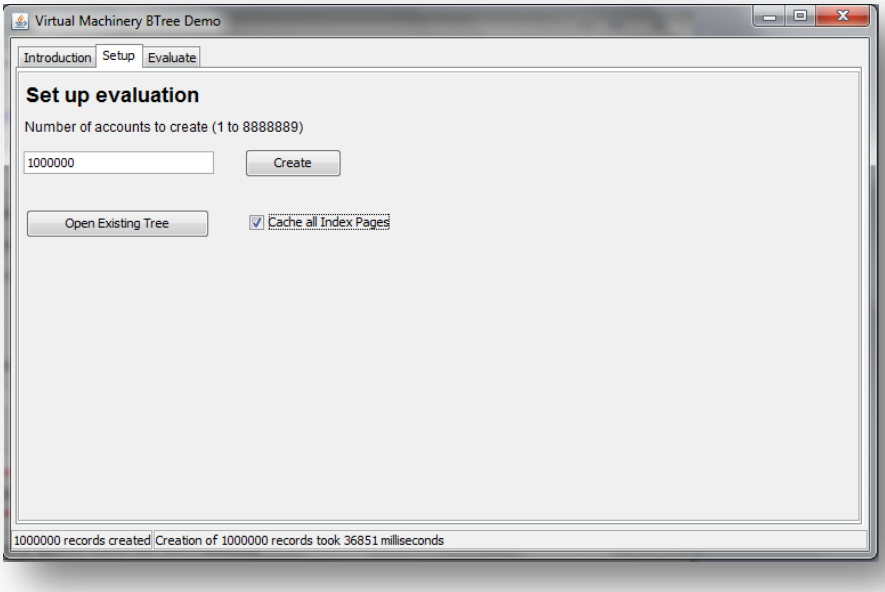


The range of values of the Account number that we can use to search for will be limited by the number of entries that we asked to be created in the 'Setup' options when we created the B+Tree. When we do a search for an account number the associated details and the time taken to fetch them are displayed.

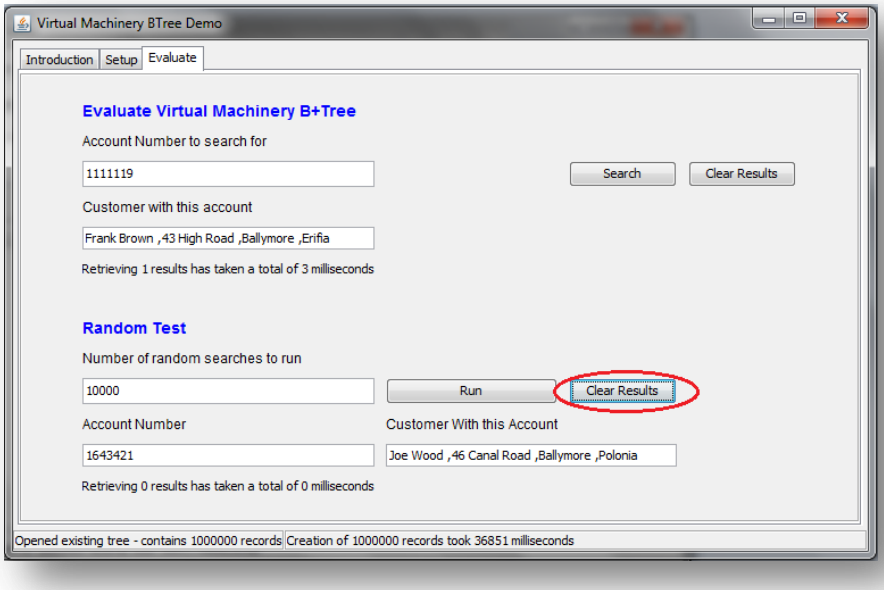
To run a performance test go down to the 'Random Test section of the 'Evaluate' tab and enter the number of random searches to run and press the 'Run' button. The results of search and the overall result will then be displayed.

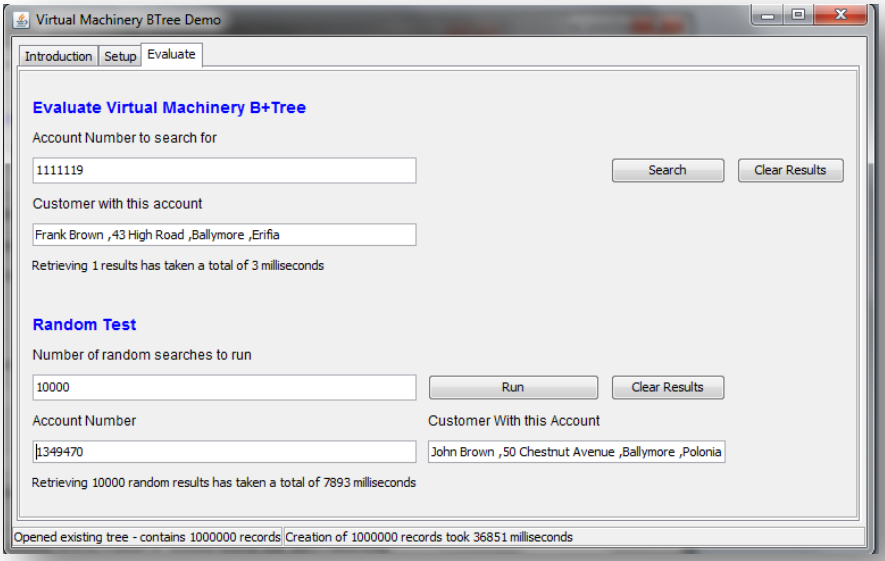


In this case we can see that the test took 131262 milliseconds. If you go back to the Setup page and reopen the test with the caching of index pages switched on you can see the effect of cacheing by rerunning the test after the cache has been switched on –



As the results displayed are cumulative you will have to clear the results before running the test -





After rerunning the test you can see that the time taken has dropped from 121 seconds to 8 seconds – quite a difference!

The code behind the BTreeDemo demo

The BTreeDemo class in the com.virtualmachinery.btreedemo package contains the logic for the creation of the screens and the handling of the action buttons. The logic for the interaction of the GUI with the BTree is contained in the BtreeDemoService class in the com.virtualmachinery.btreedemosupport package. The tree is created createPhoneTree method in the Generatetrial class in the com.virtualmachinery.btreedemosupport package.

Creating the tree is relatively simple –

- The Tree is created, (if there is a particularly large number of entries the data page size is increased from 4k to 16k to accommodate them)
- The random records are created and put in the tree
- The tree is flushed and closed

```
public void createPhoneTree(long max, JLabel status) throws BTreeException {
        if (max > 4500000) {
                ipsize= 512;
                dpsize=16384;
        }
        aTree = BTree.createNewTree("VMTEST.DAT","VMTEST.IND",2000,100, ipsize, dpsize);
        long startphone = 1111110;
        String index;
        String record;
        for (long i=0;i<max;i++) {
                startphone++;
                index = startphone+"";
                record = generateRecord(); //Generate a random name and address
                aTree.put(index, record);
                if (i%1000 == 0) {status.setText(i+" records created");}
        }
        aTree.flushBTree();
        aTree.closeBTree();
        status.setText(max+" records created");
}
```

The openExistingTree method in the BtreeDemoService uses the value of the cached flag (based on whether the checkbox has been checked on the 'Setup' tab) to open the tree with or without the cache on. If the cache is on it caches up to 1000 index pages – this is large enough to cache all of the index pages no matter what size the tree is.

```
	if (cached) {
		theTree = new BTreeRO("VMPHONETEST.DAT","VMPHONETEST.IND",1000,0);
	} else {
		theTree = new BTreeRO("VMPHONETEST.DAT","VMPHONETEST.IND",0,0);
	}
```

The openExistingTree method then uses the getPreviousAsString method of the BTree to find the last entry in the B+Tree and therefore the maximum number of records that can be accessed –

```
	theTree.initPrevious();
	String lastIndex = theTree.getPreviousAsString()[0];
	recordsCreated = Long.parseLong(lastIndex) - 1111111 +1;
```

The search method in the BtreeDemoService just takes the search key entered into the text box and looks for it in the B+Tree using the get method –

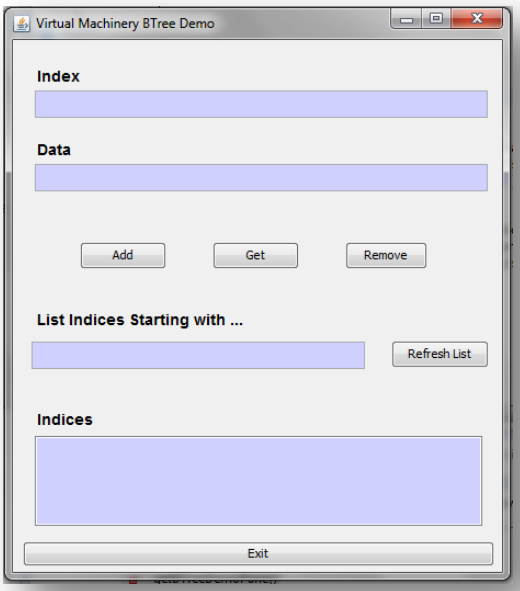
```
	standardTime -= System.currentTimeMillis();
	answer=theTree.get(index);
	standardTime += System.currentTimeMillis();
	standardResults++;
	person.setText(answer);
	status.setText("Retrieving "+standardResults+" results has taken a total of
"+standardTime+" milliseconds");
```

The runRandom method in the BtreeDemoService generates a random index in the range of available indices and uses the get method to display the value retrieved and increments the time counter –

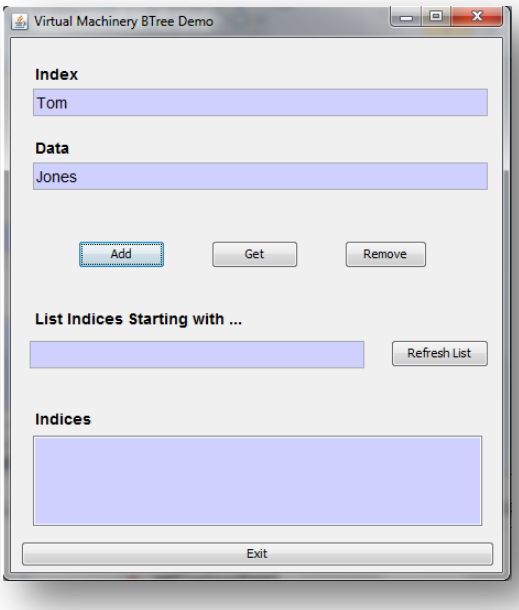
```
	for (long i = 0;i<max; i++) {
		index = ""+(randomGenerator.nextInt((int) recordsCreated)+1111111);
		phone.setText(index);
		randomTime -= System.currentTimeMillis();
		answer=theTree.get(index);
		randomTime += System.currentTimeMillis();
		randomResults++;
		details.setText(answer);
		status.setText("Retrieving "+randomResults+" random results has taken a
total of "+randomTime+" milliseconds");
	}
```

The BTreeEntryDemo demo (Base License only)

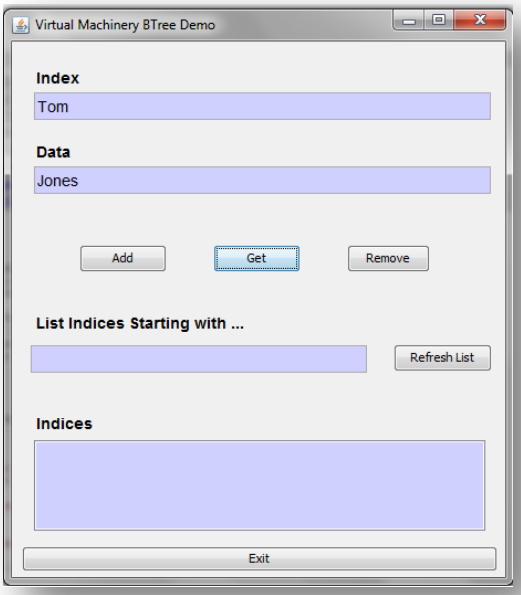
When you start the BtreeEntryDemo demo the following screen is displayed –



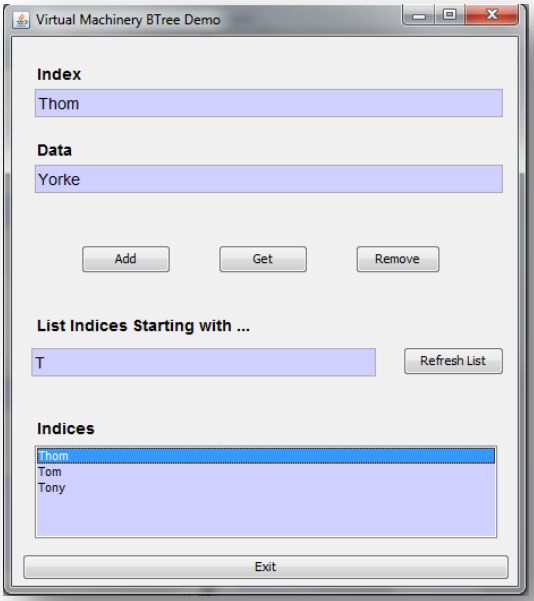
We can now add an entry to the BTree. We enter the index 'Tom' and the data 'Jones' and then press the 'Add' button –



Clear the index and data fields and the enter 'Tom' and press the 'Get' button. The data field will be populated with the name 'Jones' –

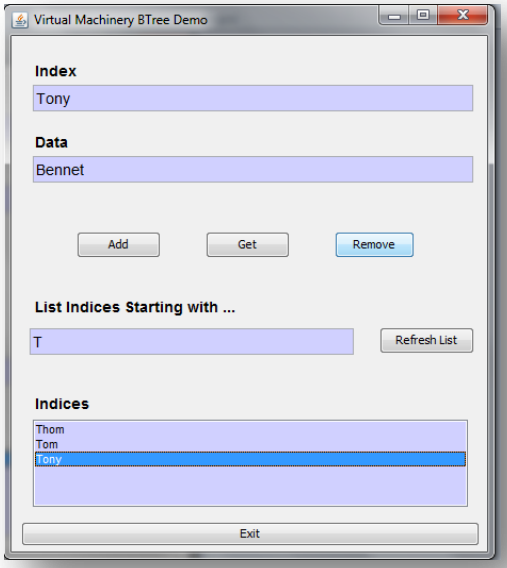


Now add two more entries ‘Thom’ (index) , ‘Yorke’ (data) and ‘Tony’ (index), ‘Bennett’ (data). Then put the letter ‘T’ in the ‘List indices starting with …’ field and press ‘Refresh List’. This will display a list of all the indices starting with ‘T’ –

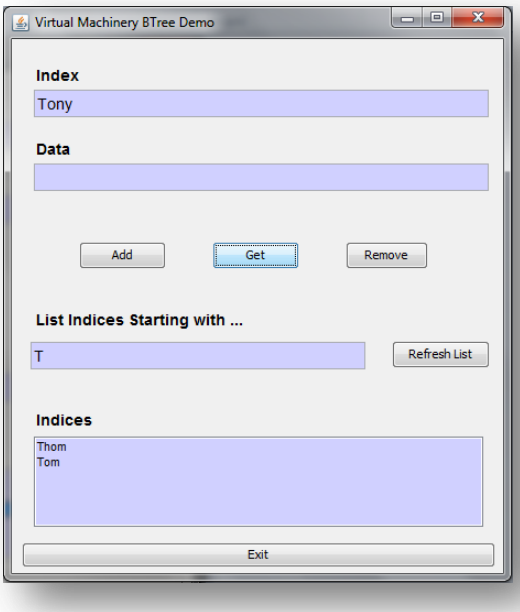


By selecting an index from the list of indices the index and data fields in the top half of the screen will be updated with the index and data for that entry.

If we want to remove an entry from the BTree(in this case ‘Tony’) we simply enter the index (or select it from the ‘Indices’ list) and press the ‘remove’ button –



Now if we list the indices we will see that the entry for ‘Tony’ has been removed. Using ‘Get’ and refreshing the list of indices starting with ‘T’ show that the entry has been removed from the B+Tree –



Note that each index must be unique. So there can only be one entry with the index ‘Tom’. If you add ‘Tom’ (index) ‘Waits’ (data) the data for ‘Tom’ (which was ‘Jones’) will be replaced with ‘Waits’.

The code behind the BTreeEntryDemo demo

All of the code for this demo is in the class BTreeEntryDemo.java in the com.virtualmachinery.btreedemo package.

The BTree instance used in this demo is created each time the demo is started – i.e. it only lasts as long as the demo. The BTree is always accessed through the getBTree() method and is created if it does not already exist.

```
private com.virtualmachinery.btree.interfaces.BTreeInterface getBTree() {
  if (aBTree == null) {
        try {
               aBTree = com.virtualmachinery.btree.btree.BTree.createNewTree("Test");
        } catch (com.virtualmachinery.btree.exceptions.BTreeException be) {
               handleException(be);
        }
  }
  return aBTree;
}
```

When the 'Add' button is clicked the following code is called. The main point of interest here is the 'put' method of the BTree – which takes the two strings from the 'Index' and 'Data' fields of the screen.

```
public void addButtonClicked(String index, String data) {
        try {
               getBTree().put(index,data);
        }
        catch (com.virtualmachinery.btree.exceptions.BTreeException be) {
               handleException(be);
        }
}
```

When the 'Get' button is clicked the following code is called. The main point of interest here is the 'get' method of the BTree – which uses the string from the 'Index' field of the screen and uses the result to populate the 'Data' field.

```
public void getButtonClicked(String index) {
        String dat="";
        try {
               dat = getBTree().get(index);
        }
        catch (com.virtualmachinery.btree.exceptions.BTreeException be) {
               handleException(be);
        }
        if (dat !=null) { getdataText().setText(dat);}
        else { getdataText().setText("");} //clear the text to avoid confusion
}
```

When the 'Refresh List' button is clicked the following code is called. The main point of interest here is the 'getIndicesStartingWith' method of the BTree – which uses the string from the 'Start' field of the screen as a parameter then uses the result to populate the 'Indices' field. Note the second parameter to the 'getIndicesStartingWith' method – 1000 – this is the maximum number of entries that should be returned.

```
public void refreshButtonClicked(String start) {
        try {
```

```
            String[] result = getBTree().getIndicesStartingWith(start, 1000);
            this.addItems(result);
        }
        catch (com.virtualmachinery.btree.exceptions.BTreeException be) {
            handleException(be);
        }
```

Finally we have the method that is called when the 'Remove' button is clicked. In this case the remove method of the BTree is called using the String in the 'Index' field on the screen.

```
    public void removeButtonClicked(String index) {
        try {
                getBTree().remove(index);
        }
        catch (com.virtualmachinery.btree.exceptions.BTreeException be) {
                handleException(be);
        }
    }
```

Java J2ME SDK B+Tree Demos

These demos have been packaged for use with the J2ME SDK and development environment and are therefore Eclipse projects. It should be easy to use the code in any environment that supports J2ME development. The code is in the src directory and the B+Tree library is in the root directory of the project.

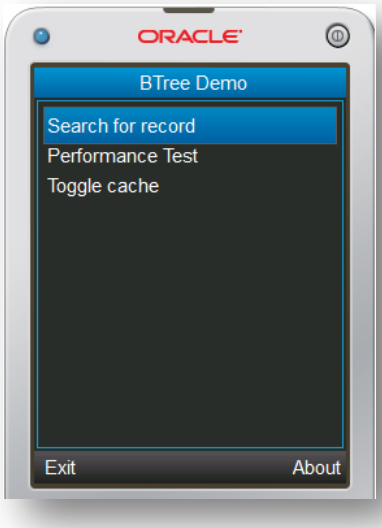
Due to the limitations of the J2ME environment the libraries have been compiled using JDK 1.4. This makes no difference to the running of the code. The other libraries are compiled using JDK 1.5

The following projects are provided in the demo –

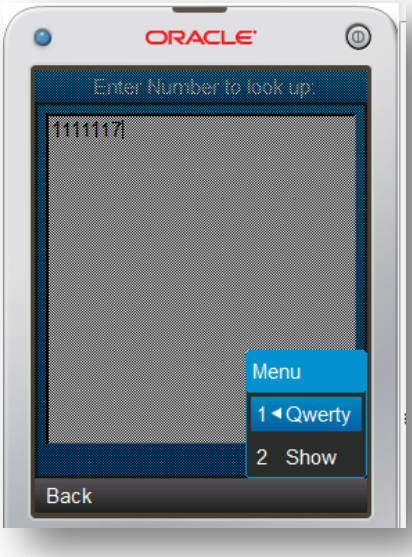
- BTreeAddressDemo (Web and Base License)
- BTreeMediaDemo (Web and Base License)
- BTreeMEDemo (Base License only)
- BTreeJSR75Demo (Base License only)
- BTreeMEROInMemDemo (Base License only)

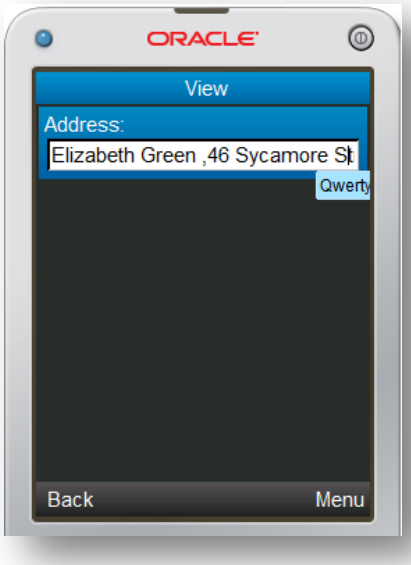
The BTreeAddressDemo (Web and Base License)

In the J2ME development environment right-click on BtreeAddressDemo project and choose the 'Run As' option then select 'Java Emulated ME Midlet' option in the Run menu. The following screen will be displayed:



Select the first option 'Search for record'. The following screen will be displayed and you should enter a number between 1111111 and 1121110. Choosing the 'Show'option will display the details associated with that number:

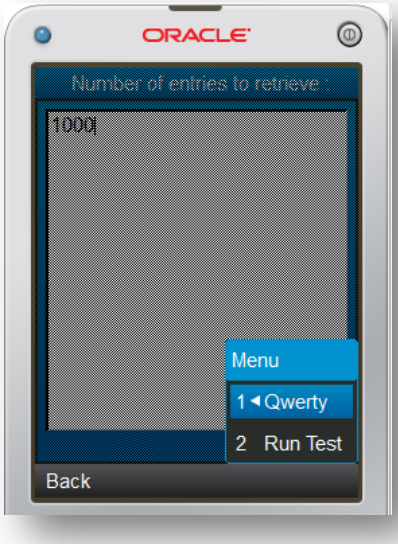


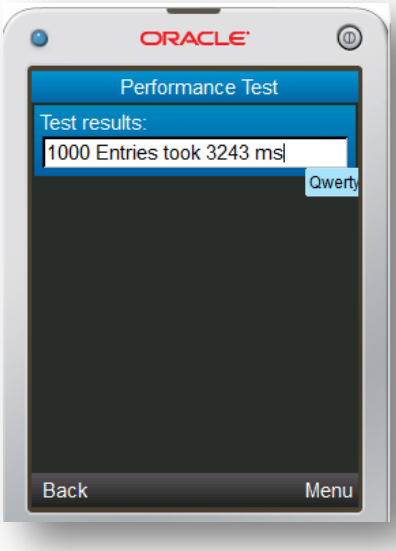


You can return to the main menu by selecting the 'Menu' option in the bottom right of the screen.

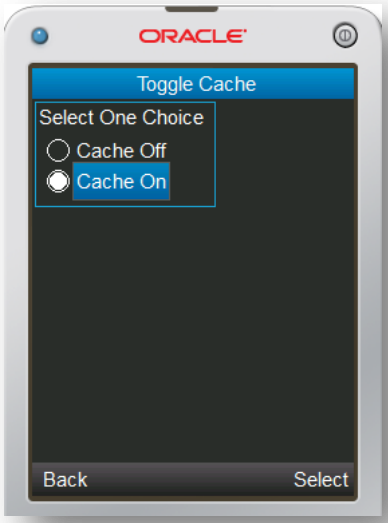
To run the performance test select the 'Performance Test' option from the menu and enter the number of entries that you want to retrieve. You can select any number that you like but you should choose a relatively small number at first as the test may take some time to run. After entering the number select the 'Run Test' option. The test will then run and the results will be displayed.

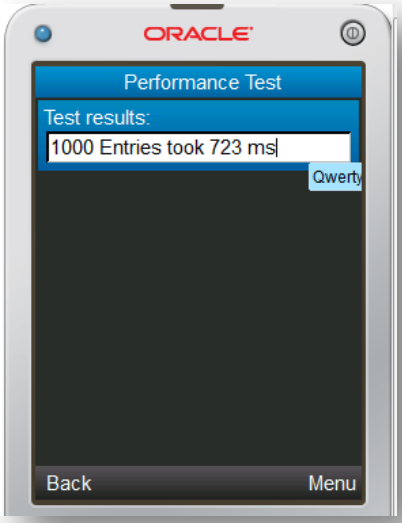
You can return to the main menu by selecting the 'Menu' option in the bottom right of the screen.





You can see how much cacheing the B+Tree can improve the performance of the test by choosing the ‘Toggle Cache’ menu and selecting the ‘Cache On’ option. Press Select and rerun the test that you ran above. You can see that the Test now takes 723ms compared to the 3243ms that the first test took (obviously these figures wil vary depending on the power of the environment that you are using).





The code behind the BTree Address demo

The code for the demo is in the BTreeAddressDemoMIDlet class in the com.virtualmachinery.btree.demo package in the BTreeAddressDemo project. This demo uses the Virtual Machinery BTree read only library for J2ME which is implemented using the JSR75 file handling functionality. This functionality is not necessarily available in all handsets so developers should check before using this library. The use of these libraries is covered in more detail in the developers guide that comes with the product.

The Tree is opened using the openBTree(..) method of the BTreeMEJSR75RO class:

```
private void openBTree(String resource) {
    System.out.println("Opening tree with cache "+cacheOn);
       try {
              if (cacheOn) {
                     addressBTree = BTreeMEJSR75RO.openExistingTree(resource,3,100);
              } else {
                     addressBTree = BTreeMEJSR75RO.openExistingTree(resource);
              }
       }
    catch(BTreeFileException bfe) {
              error("Error Creating default Tree "+bfe,10000);
       } catch (BTreeException e) {
              error("Error Creating default Tree "+e,10000);
       }
    }
}
```

Since this is a read-only application the BTree must already exist and can therefore be opened using the openExistingTree(..) method which is called with the name of the dataset being used. If the BTree is being cached then two additional parameters are required the number of index pages being cached (3) and the number of data pages being cached (100). Since these figures are greater than, or equal to, the number of pages of each type then we know that the entire BTree will be cached.

The code to view the data related to an index is in the viewForm() method –

```
String key = lookupKeyBox.getString();
if (key != null) {
        try {
              String result = addressBTree.get(key);
              if (result != null) {
                     addressField.setString(result);
                     display.setCurrent(viewForm);
                     currentMenu = "ViewForm";
              }
        }
        catch (BTreeException be) {
              System.out.println(be.toString());
       }
}
```

The important code here is the addressBTree.get(key) method. This retrieves a record from the BTree using a String (key) as the index. The value is returned as a String.

The code to run the performance test is in the runTest() method:

```
public void runTest () {
       final Thread testThread = new Thread() {
```

```
        public void run() {
                int count=0;
                int entries = Integer.parseInt(runNumberBox.getString());
                System.out.println("Running test - "+entries+" entries");
                int start=1111111;
                Random random = new Random();
                int max=10000;
                int randNo = -1;
                long startTime = System.currentTimeMillis();
                try {
                        for (int i=0;i<entries;i++) {
                                randNo = ((int) (random.nextFloat()*max)) + start;
                                if (addressBTree.get(randNo+"") != null) {
                                        count++;
                                }
                                if (count != 0) {
                                        final int progress = (int) ((count/(entries
                                *1.0))*100);
                                }
                        }
                } catch (BTreeException e) {
                        e.printStackTrace();
                } finally {
                        if (count != entries) count = -1*(count-entries);
                                String resultString = count+" Entries took
                        "+(System.currentTimeMillis() - startTime)+" ms";
                                System.out.println(resultString);
                                resultsField.setString(resultString);
                                display.setCurrent(resultsForm);
                                currentMenu = "ResultsForm";
                        };
                }
        };
        testThread.start();
}                                                       }
```

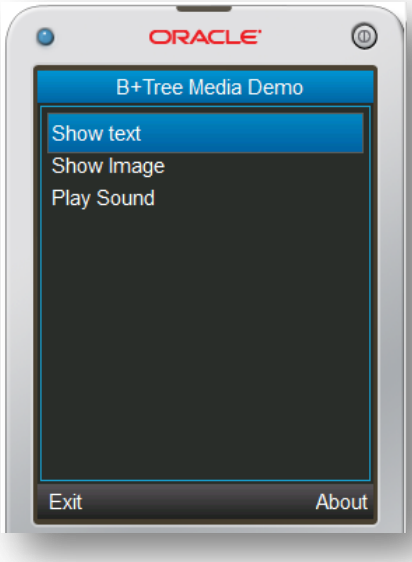
When the user selects ‘’Toggle cache’’ the value of cacheOn is set according to the option selected. The following code in CommandAction in the inner class BTreeDemoCommandListener is then called :

```
        if (cacheToGoOn != cacheOn) {
                cacheOn = cacheToGoOn;
                try {
                        addressBTree.closeBTree();
                        openBTree(filePath);
                } catch (BTreeException e) {
                        error("Error toggling cache "+e,10000);
                }
        }
```

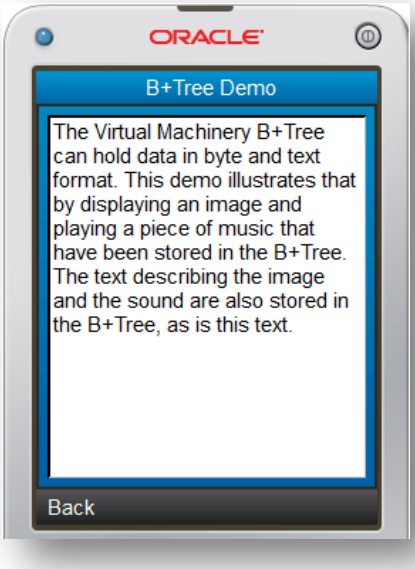
This closes the BTree and reopens it. When the BTree is reopened the value of the cacheOn flag will be taken into consideration and the BTree opened appropriately (see the description of the openBTree() code earlier.

The BTreeMedia Demo (Web and Base License)

In the J2ME development environment right-click on BTreeMediaDemo project and choose the 'Run As' option then select 'Java Emulated ME Midlet' option in the Run menu. The following screen will be displayed:



The start menu shows the three options available. Select the 'Show text' option to show the following screen:

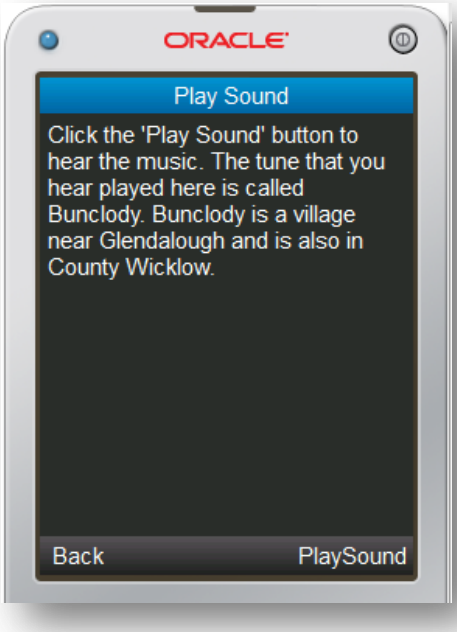


As the text on the screen describes all of the text used on this tab has been stored in a B+Tree. Press the 'Back' button in the bottom left hand corner to return to the main menu.

Select the 'Show Image' option to show the following screen:

All of the text shown on the screen and the image has been stored in the B+Tree .Press the ‘Back’ button in the bottom left hand corner to return to the main menu.

Select the ‘Play sound’ option to show the following screen:



Click the ‘Play Sound’ button to hear the music. All of the text shown on the screen and the data used to create the sound have been stored in the B+Tree. Press the ‘Back’ button in the bottom left hand corner to return to the main menu.

Pressing the 'About' button in the bottom right hand corner of the menu page will display the above screen. Press 'Done' to return to the menu page.

The code behind the BTree Media Demo

The BTree Media demo uses a different approach to data storage compared to that in the BTree Address demo. In the case of the Web version of the demo the data is stored as part of the MIDlet application and is distributed with it. This makes referencing the data set easily as it is simply a case of using the relative addresses of the files in the jar:

```
protected void startApp() throws MIDletStateChangeException {
        display.setCurrent(fmMain);
        try {
                mediaBTree = BTreeMEROJar.openExistingTree("/MEDIA.DAT","/MEDIA.IND");
        } catch (BTreeException e) {
              displayErrorAlert( "Error opening B+Tree",e);
        }
}
```

In the case of the version distributed with the product a number of alternative storage approaches are offered – you can find out more about these in the section following this ('Alternative storage approaches offered in the Product media demo').

When we are building the 'Show Text' screen we call the 'showText()' method:

```
public void showText() {
        String answer = getString("TEXT_KEY");
        TextBox textBox = new TextBox("B+Tree Demo",answer, answer.length(),
TextField.UNEDITABLE);
        textBox.addCommand(BACK_COMMAND);
        textBox.setCommandListener(new BTreeDemoCommandListener());
        display.setCurrent(textBox);
        currentMenu = "ShowText";
}
```

This calls the getString method which is a convenience method used to retrieve String data from the B+Tree using a String key. The important method here is the 'get' method of the B+Tree:

```
public String getString(String index) {
	try {
		return mediaBTree.get(index);
	} catch (BTreeException e) {
	      displayErrorAlert( "Can't access B+Tree data at index "+index,e);
	}
	return "";
}
```

When we are building the 'Show Image' screen we call the showImage() method:

```
public void showImage() {
    imageForm = new Form("Show Image");
    imageForm.addCommand(BACK_COMMAND);
    imageForm.setCommandListener(new BTreeDemoCommandListener());
    String imageTextValue = getString("IMAGE_TEXT");
       currentMenu = "ShowImage";
           imageForm.append(imageTextValue);
               imageItem = new ImageItem(null,getImage(),ImageItem.LAYOUT_CENTER,"Image not
found");
               imageForm.append(imageItem);
               display.setCurrent(imageForm);
       }
```

As this screen contains both text and the image the text is retrieved by re-using the getString(..) method described above. The image is retrieved using the getImage() method:

```
protected Image getImage() {
	byte[] bytes = getBytesFromLibrary("PICTURE_KEY");
	Image image = scaleImage(Image.createImage(bytes, 0, bytes.length),160,240);
	return image;
}
```

This gets the bytes that constitute the image from the library using the getBytesFromLibrary(..) method described below. It then takes them and uses them to re-create the image using the Java Image class. Before displaying the image it scales it to fit using the image scaling method in the class (not described here as it doesn't really relate to the B+Tree).

```
public byte[] getBytesFromLibrary(String index) {
	String iString;
	try {
		iString = getString(index);
		if (iString != null) {
			String[] base = splitRecord(((String) mediaBTree.get(index)),"^");
			int numRecords = Integer.parseInt(base[0]);
			int pixLength = Integer.parseInt(base[1]);
			byte[] pixels = new byte[pixLength];
			int start = 0;
			for (int i=0;i<numRecords;i++) {
				byte[] thisRec =
mediaBTree.get(mediaBTree.createRecord(index+"_"+i)).getValue();
				System.arraycopy(thisRec, 0, pixels, start, thisRec.length);
				start+=thisRec.length;
			}
			return pixels;
		}
		else {
			return null;
		}
	} catch (BTreeException e) {
```

```
                displayErrorAlert( "Error gettng bytes from B+Tree",e);
        }
        return null;
}
```

The getBytesFromTree(..) method uses the “PICTURE_KEY” index to retrieve all the records associated with the index and concatenate them back into a byte array.

When we are building the ‘Play sound’ screen we call the soundForm() method:

```
public void soundForm() {
        soundForm = new Form("Play Sound");
        String soundTextValue = getString("SOUND_TEXT");
        soundForm.append(soundTextValue);
        soundForm.addCommand(PLAY_COMMAND);
        soundForm.addCommand(BACK_COMMAND);
        soundForm.setCommandListener(new BTreeDemoCommandListener());
        display.setCurrent(soundForm);
        currentMenu = "PlaySound";
}
```

This contains some text and the button to play the sound. The text is fetched by re-using the getString(..) method. The ‘Play Sound’ button is registered with the BTreeDemoCommandListener which listens for the button to be pressed then calls the playSound() method:

```
public void playSound() {
    try {
        InputStream is = new ByteArrayInputStream(getBytesFromLibrary("SOUNDWAV_KEY"));
        Player p = Manager.createPlayer(is, "audio/x-wav");
        p.start();
    }
    catch (Exception e) {
      displayErrorAlert( "Unable to play WAV data!",e);
    }
}
```

The playSound(..) method fetches the bytes required to recreate the sound from the B+Tree using the getBytesFromLibrary(..) method. It puts the bytes into a stream then plays them with the Audio player.

When the user exits from the application the B+Tree is closed using the closeBTree() method.

```
protected void exit() {
        try {
                mediaBTree.closeBTree();
        } catch (BTreeException be) {
              displayErrorAlert( "Error Closing B+Tree",be);
        }
        try {
                destroyApp(true);
        }
        catch (MIDletStateChangeException msce) {
                mainMenu();
        }
        notifyDestroyed();
}
```

Alternative storage approaches offered in the Product media demo

In the product demo a number of ways of retrieving data from the B+Tree are demonstrated. In each case the API and the data set format remains the same but the access to the data storage varies according to the type of B+Tree used.

As distributed (in the BTreeMediaDemoMIDLet code supplied with both the web and product versions of the BTree) the BTree is stored in the MIDlet jar and is accessed by a relative path (as described above). This uses the BTreeMEROJar class (which meets the BTreeROInterface interface).

In the version supplied with the Base License of the product the following additional code is available:

In the definition of the class variables the following are defined:

```
public static final int STANDARD_BTREE = 0;
public static final int RMS_BTREE = 1;
public static final int JSR75_BTREE = 2;
public static final int IN_MEM_BTREE = 3;
public static final int JAR_BTREE = 4;
```

In the openBTree(…) method the type is passed as a parameter to determine which type of B+ tree is used to open the dataset. You can find further details of each of these B+ tree types in the development guides but for the moment all you need to know is that each of these types meets the BTreeRO interface and can be used to open the same dataset:

```
public void openTree(boolean cache, int type) throws BTreeException {
	if (type == JSR75_BTREE) {
		//Make sure that the MEDIA.DAT and MEDIA.IND files are in the photos
directory
		String path = System.getProperty("fileconn.dir.photos"); // for example:
file:///C:/Data/Images/
		mediaBTree = BTreeMEJSR75RO.openExistingTree(path+"MEDIA");
	}
	if (type == RMS_BTREE) {
		String path = "";
		mediaBTree = BTreeMERMSRO.openExistingTree(path+"MEDIA");
	}
	if (type == IN_MEM_BTREE) {
		//Make sure that the MEDIA.DAT and MEDIA.IND files are in the photos
directory
		String path = System.getProperty("fileconn.dir.photos");
		mediaBTree =
BTreeMEROInMem.openExistingTree(path+"MEDIA.DAT",path+"MEDIA.IND");
	}
	if (type == JAR_BTREE) {
		//Make sure that the MEDIA.DAT and MEDIA.IND files are in the photos
directory
		mediaBTree = BTreeMEROJar.openExistingTree("/MEDIA.DAT","/MEDIA.IND");
	}
	if (type == STANDARD_BTREE) {
		//Make sure that the MEDIA.DAT and MEDIA.IND files are in the photos
directory
		String path = "";
		mediaBTree = BTreeMERO.openExistingTree(path+"MEDIA");
	}
}
```

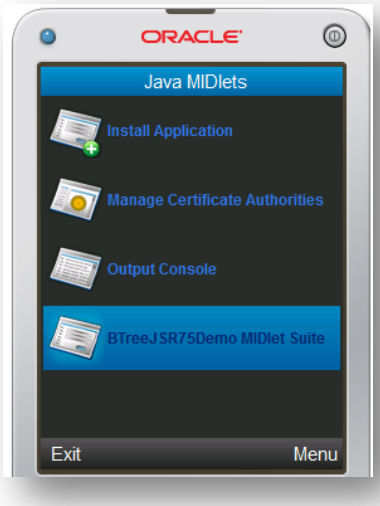
The initial distribution uses the JAR_BTREE type – you can experiment by modifying the code to use to use the different types of tree. Note that not all types of B+Tree will be supported by all platforms.

The jars required to support each type of tree are supplied in the project that comes with the base license.

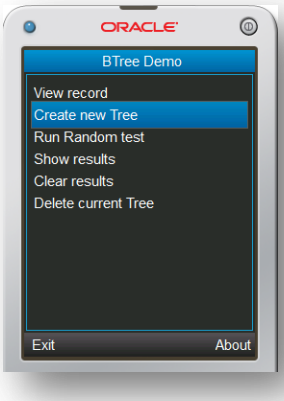
BTreeJSR75Demo (Base License only)

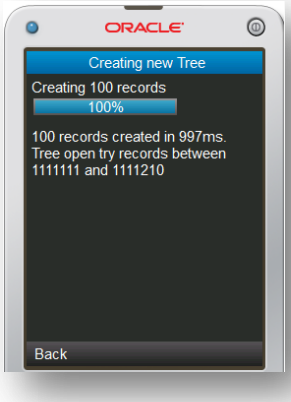
This demo uses the 'standard'JSR75 approach for handling files on J2ME devices. This allows files to be handled in pretty much the same way as files on a desktop or server file system i.e. random read and write access.

To start the demo in the J2ME SDK environment right click on the BTreeMEJSR75Demo class in the BTreeJSR75Demo project and choose the 'Run As' option then select 'Java Emulated ME Midlet' option in the Run menu. The following screen will be displayed:



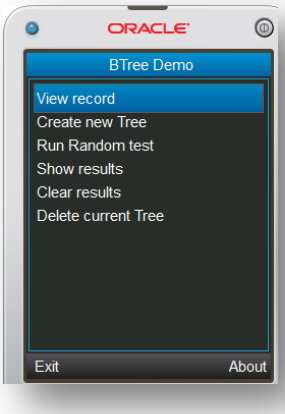
When you click on the 'BTreeJSR75Demo MIDLet suite'option the main menu will be displayed. Before you do anything else you must first create a B+Tree using the 'Create New Tree'option:

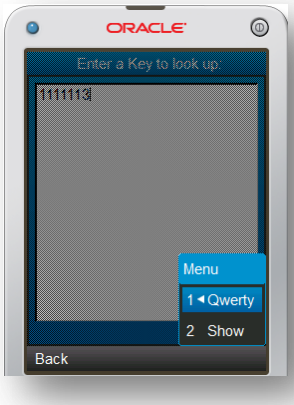


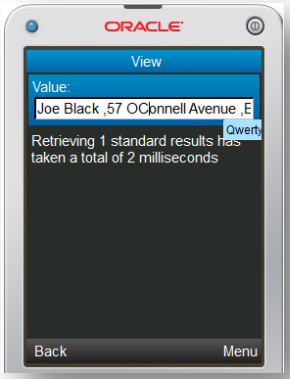


In this case we have created a hundred new entries. You should generally use a small number until you find out how powerful your environment is. The screen will show the key values that you can use in future tests on this B+Tree.

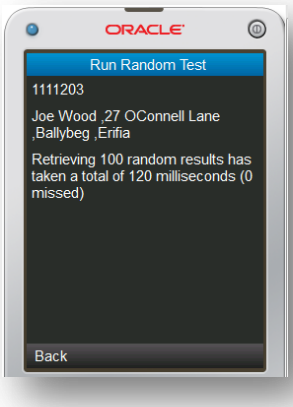
After you have created the B+Tree go back to the main menu by pressing the ‘Back’ button in the bottom left of the screen. Select the ‘View Record’ option from the main menu and enter the key of a record that you want to look up. The maximum value of key that you can use is equal to 1111111 plus the number of entries that you created, minus 1 (this is the number displayed when you created the tree). In this case we have used 1111113:





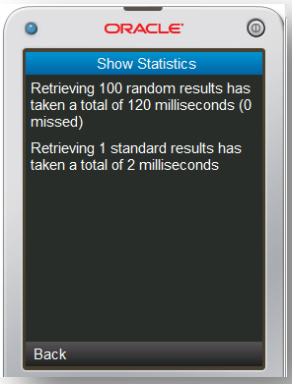


You can see that there is a value returned for this key. The next thing to do is to run the random test. This allows you to randomly access entries by their keys and will give you some idea of the performance of the system. Use the ‘Run Random Test’option on the main menu and enter the number of tests that you want to run:



In this case we have elected to run 100 tests and we can see that accessing the 100 records took 120 milliseconds.

If you go back to the Main Menu and select the ‘Show Results’ option you will see the cumulative results of all the tests run so far:



You can reset the results by choosing the ‘’Clear results’ option from the Main Menu. You can also delete the B+Tree by selecting the ‘Delete Current Tree’ option.

The code behind the BTreeJSR75Demo

The BTree implementation for JSR75 has the same interface as the normal file based BTree implementations.

The path of the tree is defined in the static variable STANDARD_TREE. This may vary from implementation to implementation if you are not using the SDK (or are using an older version of it:

```
        static String STANDARD_TREE = "file:///root1/VMTEST";
```

When the application is first started the code in the startApp() method is run. From a BTree perspective the following code is the most important:

```
        try {
                btree = (BTreeJSR75Demo)
BTreeJSR75Demo.openExistingTree(STANDARD_TREE,12,420);
                btree.loadMem();
                recordsInTree = btree.getNumTestRecords();
        }
        catch(Exception e) {
                System.out.println("Error opening existing Tree");
                e.printStackTrace();
        }
         mainMenu();
    }
```

In this case if there is an existing BTree then it is opened with a number of index and data buffers. This number is arbitrary and you can change it to test the effect of different buffer sizes on your code (for a fuller explanation of how to do this see the ….. section in the development guide). If you look at the code for the BTreeJSR75Demo class you will see that it is an example of an extension to the BTree class. Extensions can be used to add functionality to your BTree – in this case we provide functionality to find the number of records in the BTree. This functionality is implemented in the getNumTestRecords() method which looks at the last record to calculate how many records are in the tree. This can be done because the records are added sequentially. You can find more detail in the section on extending BTrees in the Development guide.

If there is no BTree already defined at the STANDARD_TREE location a new one will have to be created. This done using the createNewTree(..) method.

```
    private void createNewTree(int max, String fName, Gauge aGauge) throws BTreeException {
       try {
              cleanTree(); //Delete the old one first
              GenerateRandomTest aTest = new GenerateRandomTest();
              aGauge.setMaxValue(max);
              aGauge.setValue(0);
              aTest.createRandomTree(STANDARD_TREE, max,0, aGauge);
              aTest = null;
              System.gc();
              try {
                     btree = (BTreeJSR75Demo)
BTreeJSR75Demo.openExistingTree(STANDARD_TREE,250,0);
              }
              catch(Exception e) {
                     System.out.println("Error opening existing Tree");
                     e.printStackTrace();
              }
       }
```

The createRandomTree(…) method of GenerateRandomTest will create the tree and update the thermometer bar which indicates the progress of the tree creation. From a BTree perspective the only important code sections are:

```
       BTreeMEJSR75 btree = null;
       try {
              btree = (BTreeMEJSR75) BTreeMEJSR75.createNewTree(resource);
       } catch (BTreeException e) {e.printStackTrace();}
```

which creates the new tree and:

```
              try {
                     btree.put(baseindex,record);
              }
              catch (BTreeException e) {
                     e.printStackTrace();
              }
```

which adds a new Record.

The code to display and clear results are in the showResults() and clearResults() methods respectively.

When you select the option to ‘Delete current tree’the cleanTree() method is called which closes and deletes the BTree:

```
    private void cleanTree() {
       try {
              if (btree != null) { btree.closeBTree();}
              if (btree != null) { btree.deleteBTree();}
       }
       catch (Exception e) {
              e.printStackTrace();
       }
```

```
        btree = null;

    }
```

BTreeMEROInMemDemo (Base License only)

This illustrates a way to use the B+Tree implementation remotely (either over an http connection or, in this case, a socket connection).

To run the demo we first have to set up a socket connection. To do his click on the VMSocketPoller class in the BTreeSocketUtilities project and select 'Run As..' then 'Java application'. You will see the following displayed on the Console:

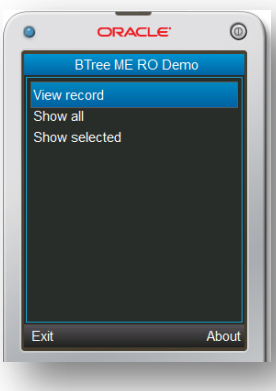
```
Socket polling started on port 5000
```

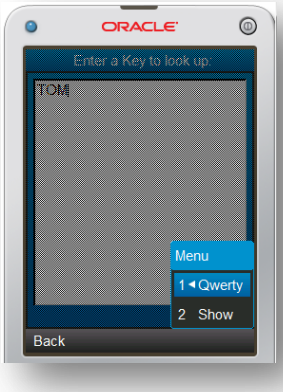
Then right click on the BTreeMEDemo class in the BTreeMEDemo project and choose the 'Run As' option then select 'Java Emulated ME Midlet' option in the Run menu. The following will be displayed on the VMSocketPoller console as the application starts:

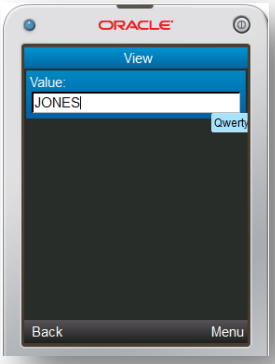
```
Reading file SocketTree.DAT
File read successfully - SocketTree.DAT
Reading file SocketTree.IND
Closing socket
File read successfully - SocketTree.IND
Closing socket
```

The BTree has now been loaded on to the device. The following screen will be displayed. Select the 'Show All' option and all of the entries (i.e. the indices) in the BTree will be shown. By selecting one of the entries and pressing the 'Show' button at the bottom left of the screen you will be shown the data associated with that entry:

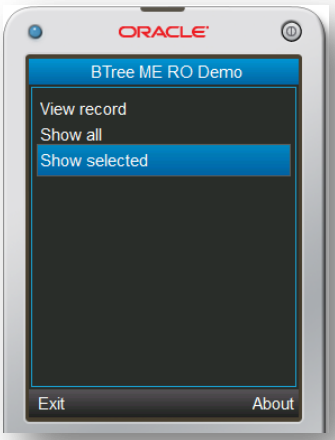
Press the 'Back'button in the bottom left hand corner of the screen to return to the Main menu. You can now select the 'View Record'option and enter one of the indices shown above to see the data associated with it:



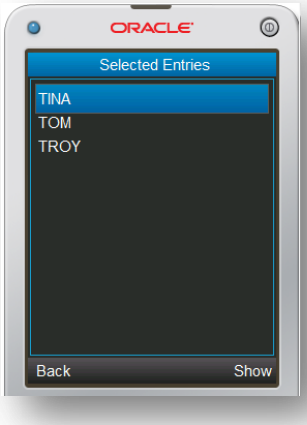


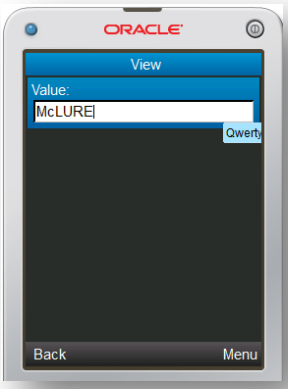


Press the 'Back'button in the bottom left hand corner of the screen to return to the Main menu. You can now select the 'Show Selected'option. This allows you to show all the indices that start with a particular string. In this case we select 'T':



When we select the show option from the Menu (press the button in the botom right of the screen) we can see all the indices that start with 'T'.





Selecting one of the entries and pressing the ‘Show’button in the bottom right of the screen will show the data associated with the selected index.

On the Main Menu screen pressing the Ábout’button in the bottom right of the screen will display the folloing screen:

The code behind the BTreeMEROInMemDemo

The startApp() method opens the BTree over a socket which has been opened by the VMSocketPoller class. The code to open the BTree in startApp() is as follows:

```
openBTree("socket://localhost:5000/SocketTree",false); //Using Socket
```

This calls the openBTree(..) method. The important part of the code here is :

```
private boolean openBTree(String resource, boolean newTree) {
	boolean ok = false;
	try {
		btree = new BTreeMEROInMem(resource+".DAT",resource+".IND");
		ok = true;
	} catch(BTreeFileException bfe) {
		error("Could not open BTree  - Check Connection ("+bfe.toString()+")");
	}
	catch(BTreeException be) {error("BTree Exception  - Check Connection
("+be.toString()+")");}
	catch(Exception e) {error("General Exception opening BTree  - Check Connection
("+e.toString()+")");}
	finally {
		return ok;
	}
}
```

The BTreeMEROInMem class constructor takes the URL and tries to open the B+Tree on the socket.

When we select the ‘Show All’ option on the menu the showListAll() method is called. The important part of this method is the code that iterates across the BTree collecting all the indices as it goes:

```
    private void showListAll() {
```

```
        keylistAll = new List("All Entries", Choice.IMPLICIT);
        keylistAll.addCommand(BACK_COMMAND);
        keylistAll.addCommand(SHOW_COMMAND);
        keylistAll.setCommandListener(new BTreeDemoROCommandListener());

         try {
                btree.initNext();
                BTreeRecordInterface[] temp = btree.getNext();
                while ((temp != null) && (temp[0] != null)) {
                        keylistAll.append(temp[0].asString(),null);
                        temp = btree.getNext();
                }
        } catch (BTreeException be) {
                System.out.println(be.toString());
        }
         display.setCurrent(keylistAll);
         currentMenu = "KeyListAll";
    }
```

As you can see the List is created as the variable keyListAll. The initNext() method is called to reset the pointer back to just before the first index in the B+Tree then the getNext() method is called in a loop to repeatedly get the next entry in the B+Tree until the last value is null indicating that the iteration across the B+Tree is complete.

When we select 'Show Selected' option we call the showListSome() method:

```
private void showListSome(String aString) {

        keylistSome = new List("Selected Entries", Choice.IMPLICIT);
        keylistSome.addCommand(BACK_COMMAND);
        keylistSome.addCommand(SHOW_COMMAND);
        keylistSome.setCommandListener(new BTreeDemoROCommandListener());

        try {
                String[] temp = btree.getIndicesStartingWith(aString,120);
                for (int i=0;i<temp.length;i++) {
                        keylistSome.append(temp[i],null);
                }
        } catch (BTreeException be) {
                System.out.println(be.toString());
        }
         display.setCurrent(keylistSome);
         currentMenu = "KeyListSome";
}
```

In this case the method getIndicesStartingWith(..) is called with the string that was entered and the number 120 as parameters. The 120 is the maximum number of indices to be returned by the method (there are far less than 120 entries in this B+Tree – 120 is just used as an example here).

If the 'Show' option is selected from the list display then the viewForm(String, String) method is called with the value of the selected entry. This fetches the value associated with the index using the BTree get(..) method and populates the resultField with the value returned.

If the ‘View Record’ option is selected from the main menu then the viewForm() method is called. This fetches the value associated with the index entered by the user (in the lookupKeyBox) using the BTree get(..) method and populates the resultField with the value returned.

BTreeMEDemo (Base License only)

This illustrates a way to use the B+Tree implementation remotely (either over an http connection or, in this case, a socket connection). Although it uses the same technology this demo differs from the example shown above in the BTreeROInMemDemo since the BTree can be modified and saved in this case.

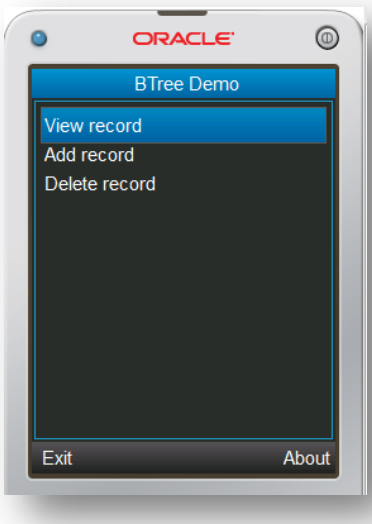
To run the demo we first have to set up a socket connection. To do his click on the VMSocketPoller class in the BTreeSocketUtilities project and select 'Run As..' then 'Java application'. You will see the following displayed on the Console:

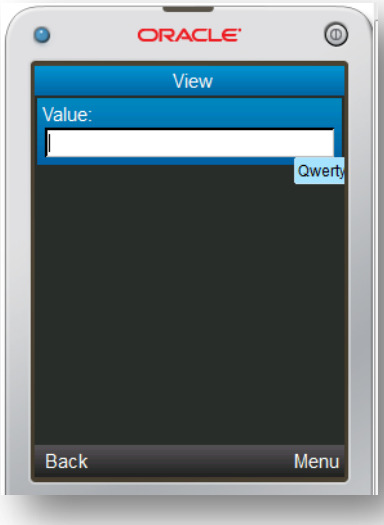
```
Socket polling started on port 5000
```

Then right click on the BTreeMEDemo class in the BTreeMEDemo project and choose the 'Run As' option then select 'Java Emulated ME Midlet' option in the Run menu. The following will be displayed on the VMSocketPoller console as the application starts:

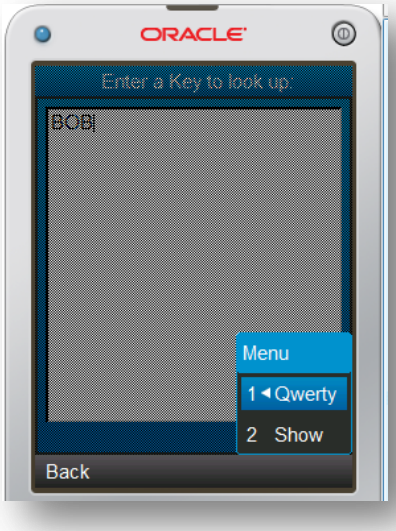
```
Reading file SocketTree.DAT
File read successfully - SocketTree.DAT
Reading file SocketTree.IND
Closing socket
File read successfully - SocketTree.IND
Closing socket
```

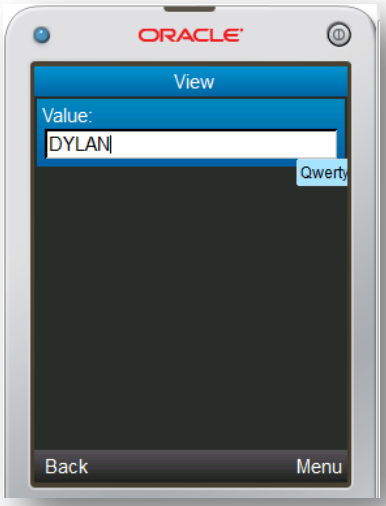
The BTree has now been loaded on to the device. The following screen will be displayed. Select the 'ViewRecord' option and you will be prompted to enter the index of the record that you want to view:





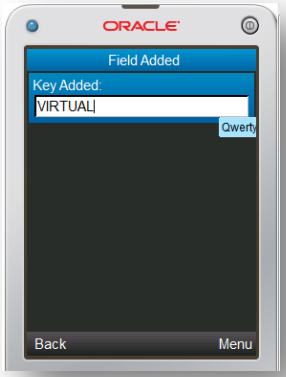
Enter the key 'BOB' and press the 'Show' option on the menu the value 'DYLAN' will then be displayed:



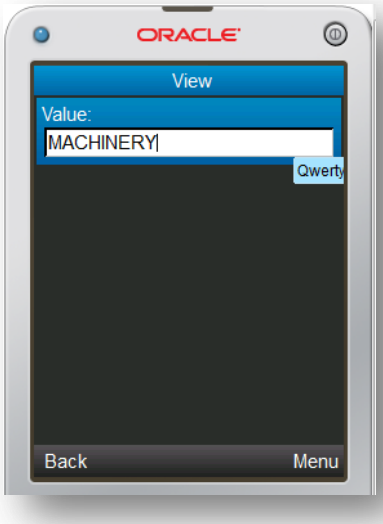


Press the 'Menu' option in the bottom right hand corner to return to the main menu.

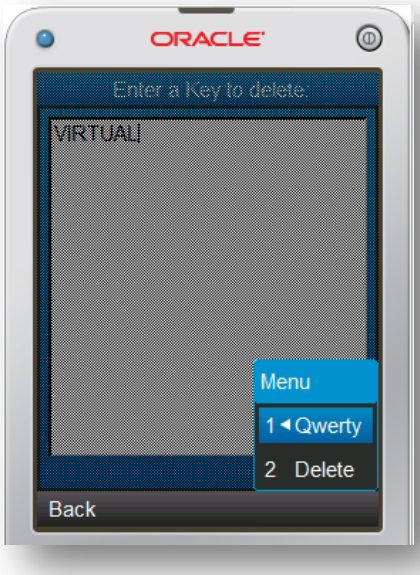
When you select the ''Add Record" option you will be asked to enter the key then the data – you can then choose the "Add" option from the menu and the data will be added to the BTree.

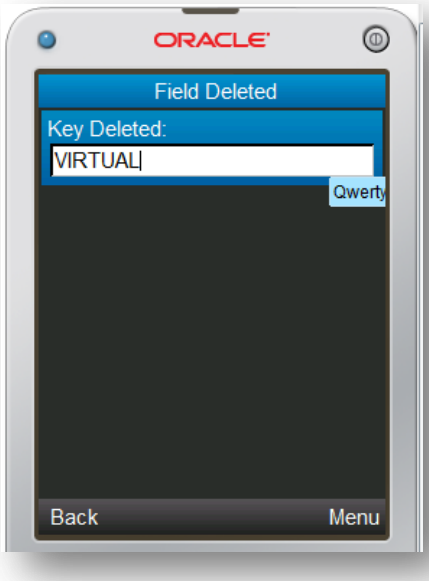


You can then go back to the main menu and check that your new entry has been added by selecting the 'View Record' option again and entering the key 'VIRTUAL':



You can then return to the Main Menu (using the 'Back' button) and select the 'Delete Record' option and delete the key 'VIRTUAL':





Now if you use the 'View Record' option and enter the key 'VIRTUAL' you will find that there is no entry for the key.

When you press the 'Exit' option on the Main Menu the application will close but as this process is occurring you will see the following displayed on the VMSocketPoller console:

```
 Writing file SocketTree.IND
Number of bytes IN FILE 1024
Writing file SocketTree.DAT
Number of bytes IN FILE 1024
File written successfully - SocketTree.IND
Closing socket
```

```
File written successfully - SocketTree.DAT
Closing socket
```

This indicates that the BTree has been written out to the socket ready for use on the next occasion that it is required. Any changes that you have made to the BTree will have been written out.

The code behind the BTreeMEDemo

The startApp() method opens the BTree over a socket which has been opened by the VMSocketPoller class. The code to open the BTree is as follows:

```
openBTree("socket://localhost:5000/SocketTree",false); //Using Socket
```

The openBTree(..) method opens the BTree but if If the socket is not open then a default tree (which has the same contents as the Socket tree) will be created

```
private void openBTree(String resource, boolean newTree) {
   try {
          btree = BTreeME.openExistingTree(resource);
   }
   catch(BTreeFileException bfe) {
          System.out.println(bfe.toString());
          System.out.println("Creating default Tree");
          try {
                 btree = BTreeME.createNewTree("TEST");
                 btree.put(btree.createRecord("BOB"),btree.createRecord("DYLAN"));
                 btree.put(btree.createRecord("STEVIE"),btree.createRecord("WONDER"));
                 btree.put(btree.createRecord("TINA"),btree.createRecord("WEYMOUTH"));
                 btree.put(btree.createRecord("KURT"),btree.createRecord("COBAIN"));
                 btree.put(btree.createRecord("ANGIE"),btree.createRecord("STONE"));
                 usingLocalDefaultTree = true;
          }
          catch(BTreeException be) {error("BTree Exception "+be.toString(),5);}
   }
   catch(BTreeException be) {error("BTree Exception "+be.toString(),5);}
}
```

The code to read in the files of the BTree dataset is in the VMSocketListener class.

The code to display the value of a record for a given index is in the viewForm() method. The get(..) method of the BTree is called with the string entered in the preceding screen:

```
private void viewForm() {
    if (lookupKeyBox.getString().length()!=0) {
          try {
             resultField.setString(btree.get(lookupKeyBox.getString()));
              display.setCurrent(viewForm);
              currentMenu = "ViewForm";
          }
          catch (BTreeException be) {System.out.println(be.toString());
       }
    }
}
```

The code to add an entry to the BTree is in the confirmAddForm() method. The put(..) method of the BTree takes as its parameters the strings entered in the key and value entry boxes of the previous screens:

```
private void confirmAddForm() {
   if ((addKeyBox.getString().length()!=0)  && (valueBox.getString()!=null)){
       try {
            btree.put(addKeyBox.getString(), valueBox.getString());
       }
       catch (BTreeException e) {error("BTree Exception "+e.toString(),5);}
       finally {
          addConfirmField.setString(addKeyBox.getString());
           display.setCurrent(confirmAddForm);
           currentMenu = "ConfirmAddForm";
       }
   }
}
```

The code to delete an entry from the BTree is in the confirmDeleteForm() method. The remove(..) method of the BTree takes as its parameters the string entered in the key entry field in the previous screen:

```
private void confirmDeleteForm() {
   if (deleteKeyBox.getString().length()!=0){
           try {
                  btree.remove(deleteKeyBox.getString());
           }
           catch (BTreeException be) {System.out.println(be.toString());}
           deleteConfirmField.setString(deleteKeyBox.getString());
            display.setCurrent(confirmDeleteForm);
            currentMenu = "ConfirmDeleteForm";
       }
}
```

When the ‘Exit’ option in the bottom left of the main menu screen is selected the BTree is closed and the application is exited. The closeBTree() causes the BTree to be written out over the socket and written to file by the VMSocketListener. Any changes made to the BTree will be saved to the BTree written by the VMSocketListener and you will see the following written to the VMSocketListener console:

```
Writing file SocketTree.IND
Number of bytes IN FILE 1024
Writing file SocketTree.DAT
Number of bytes IN FILE 1024
File written successfully - SocketTree.IND
Closing socket
```

Nokia Asha J2ME Development environment B+Tree Demos

These demos have been packaged for use with the Nokia Asha SDK and development environment and are therefore Eclipse projects. It should be easy to use the code in any environment. The code is in the src directory and the B+Tree library is in the root directory of the project.

Due to the limitations of the Asha environment the libraries have been compiled using JDK 1.4. This makes no difference to the running of the code. The other libraries are compiled using JDK 1.5.

The additional demos in the J2ME demos directory that are distributed with the Base License should be able to run in the Asha environment with minor modifications.

The following classes are provided in the demos supplied with both the Demos downloaded from the Web and those packaged with the base License –

- BTreeAddressDemoMIDlet.java
- VMBTreeMediaDemo.java

Both these demos are identical in purpose to those distributed for the other platforms.

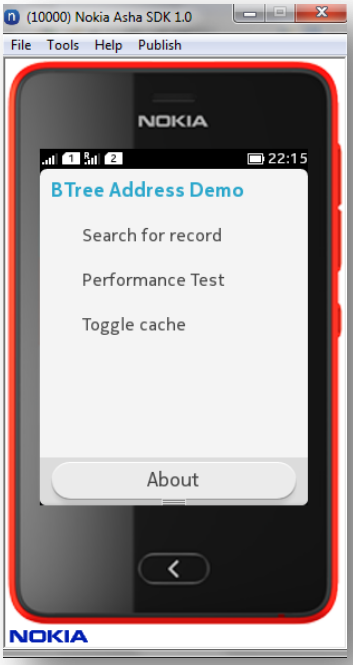
BtreeAddressDemo (Web and Base License)

Setting up the environment

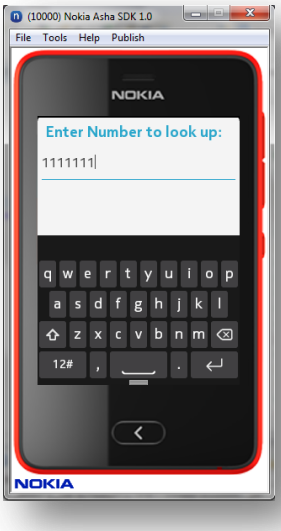
In the Nokia Asha development environment files are given logical names that relate to physical file destinations on the device or emulator. To run these tests you will have to make sure that the VMPHONETEST.DAT and VMPHONETEST.IND files are located in the correct directory. In the case we have chosen to use the 'photos' directory. If you are using the emulator and you're not sure where that directory is located on your disk then you can put the value of the path out to the console by editing the code and using System.out to print out the path name after it has been obtained from the System properties (here is the code - which is in the startApp() method) :

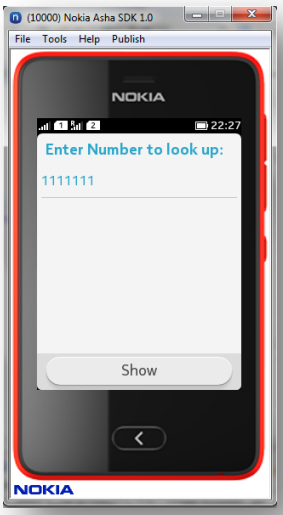
```
//Make sure that the VMPHONETEST.DAT and VMPHONETEST.IND files are in the photos directory
String path = System.getProperty("fileconn.dir.photos");
System.out.println(path); //**** Add this line to see the path *******
filePath = path+"VMPHONETEST";
openBTree(filePath);
```

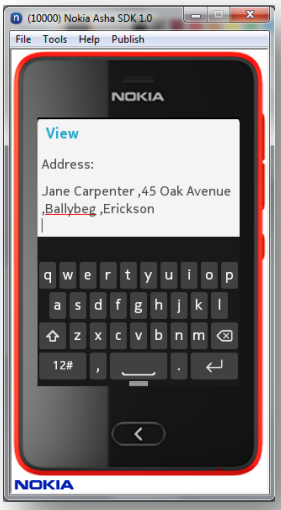
In the Asha development environment right-click on BTreeAddressDemo project and choose the 'Run As' option then select 'Java Emulated ME Midlet' option in the Run menu. The following screen will be displayed:



Select the first option ‘Search for record’. The following screen will be displayed and you should enter a number between 1111111 and 1121110. Choosing the ‘Show’option will display the details associated with that number:



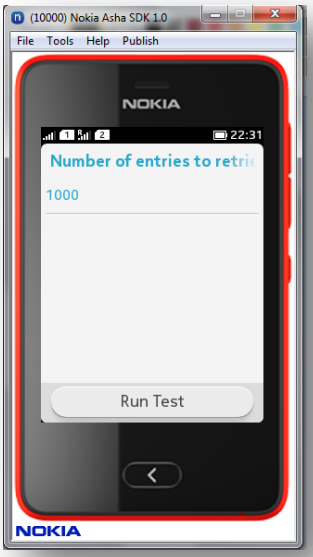


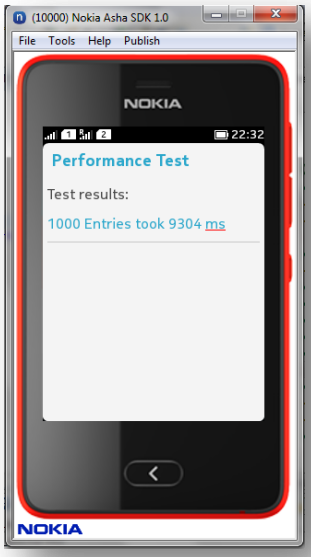


You can return to the main menu by pressing the ‘<’ button at the bottom of the screen.

To run the performance test select the ‘Performance Test’ option from the menu and enter the number of entries that you want to retrieve. You can select any number that you like but you should

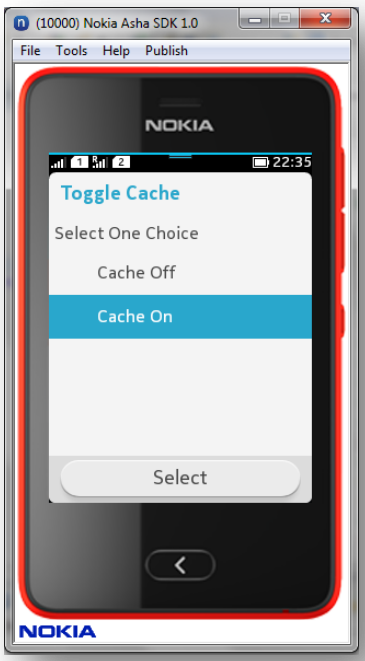
choose a relatviely small number at first as the test may take some time to run. After entering the number select the ‘Run Test’ option. The test will then run and the results will be displayed.



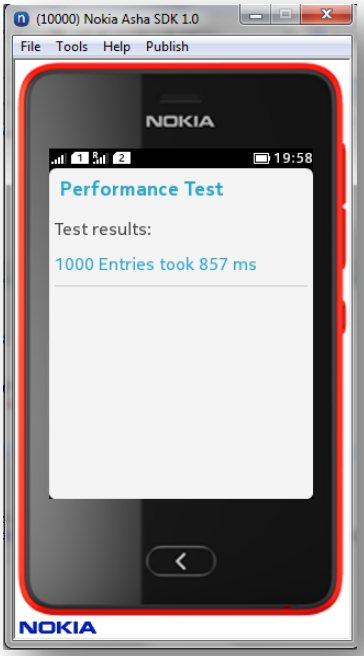


You can return to the main menu by pressing the ‘<’ button at the bottom of the screen.

By default the test is run with the caching feature of Virtual Machinerys B+Tree turned off. To turn on caching choose the ‘Toggle Cache’ option on the main menu. Select the ‘Cache On’ option and press the ‘Select’ button at the bottom of the screen:



You can then rerun the performance test and you will see that the test runs much faster (in this case 857ms compared to 9304ms – your figures will vary depending on the environment that you are running the tests in):



You can return to the main menu by pressing the '<' button at the bottom of the screen. Pressing the 'About' button at the bottom of the menu screen will display details about the demo:

Pressing ‘Done’ will return you to the main menu.

The code behind the BTree Address Demo

The code for the demo is in the BTreeAddressDemoMIDlet class in the com.virtualmachinery.btree.demo package in the BTreeAddressDemo project. This demo uses the Virtual Machinery BTree read only library for J2ME which is implemented using the JSR75 file handling functionality. This functionality is not necessarily available in all handsets so developers should check before using this library. The use of these libraries is covered in more detail in the developers guide that comes with the product.

The Tree is opened using the openBTree(..) method of the BTreeMEJSR75RO class:

```
    private void openBTree(String resource) {
       try {
              if (cacheOn) {
                     addressBTree = BTreeMEJSR75RO.openExistingTree(resource,3,132);
              } else {
                     addressBTree = BTreeMEJSR75RO.openExistingTree(resource);
              }
       }
    catch(BTreeFileException bfe) {
       ……
       }
    }
```

When the initial screen is displayed and the ‘Search for Record’ method is selected then a screen is displayed with an entry field for the key to the data that is to be looked up. After the data has been selected the ‘Show’ button is pressed and the data is looked up and displayed by the ‘viewForm()’ method :

```
private void viewForm() {
   if (lookupKeyBox.getString().length()!=0) {
            String key = lookupKeyBox.getString();
            if (key != null) {
                try {
                       String result = addressBTree.get(key);
                       if (result != null) {
                              addressField.setString(result);
                              display.setCurrent(viewForm);
                              currentMenu = "ViewForm";
                       }
                }
                catch (BTreeException be) {System.out.println(be.toString());
            }
    }
 }
```

The main feature here is the use of the B+Tree ‘get’ method – the key is passed in as a string parameter to the get method and the value associated with that key is returned as a string. The string containing the account details is then displayed in the view.

The performance test is run using the ‘runTest’ method of the class. This uses the data supplied in the runNumberBox (the ‘Number of entries to retrieve’ field on the screen), converts it into an integer then for each iteration creates a random index to look up in the B+Tree. All the time it keeps a count of how many entries it has successfully retrieved:

```
    public void runTest () {
       final Thread testThread = new Thread() {
```

```
            public void run() {
               int count=0;
               int entries = Integer.parseInt(runNumberBox.getString());
               System.out.println("Running test - "+entries+" entries");
               int start=1111111;
                Random random = new Random();
                int max=10000;
                int randNo = -1;
                long startTime = System.currentTimeMillis();
                 try {
                       for (int i=0;i<entries;i++) {
                              randNo = ((int) (random.nextFloat()*max)) + start;
                              if (addressBTree.get(randNo+"") != null) {
                                      count++;
                              }
                       }
                } catch (BTreeException e) {
                       e.printStackTrace();
               } finally {
                       if (count != entries) count = -1*(count-entries);
                       String resultString = count+" Entries took
"+(System.currentTimeMillis() - startTime)+" ms";
                       System.out.println(resultString);
                       resultsField.setString(resultString);
                       display.setCurrent(resultsForm);
                       currentMenu = "ResultsForm";
                  };
                 }
       };
       testThread.start();
}
```

Again the important method is B+Tree 'get' method – the random key is passed in as a string parameter to the get method and the value associated with that key is returned as a string.

The 'Toggle Cache' option is implemented by the toggleCache() method.

```
    public void toggleCache() {
        display = Display.getDisplay(this);
        // declaration field with initialization
        Form toggleForm = new Form("Toggle Cache");
        selectChoice = new ChoiceGroup("Select One Choice", Choice.EXCLUSIVE,
CACHE_OPTIONS, null);
        // add field into form
        int index = toggleForm.append(selectChoice);
        toggleForm.addCommand(SELECT_COMMAND);
        toggleForm.addCommand(BACK_COMMAND);
        toggleForm.setCommandListener(new BTreeDemoCommandListener());
        display.setCurrent(toggleForm);
        currentMenu="ToggleCache";
    }
```

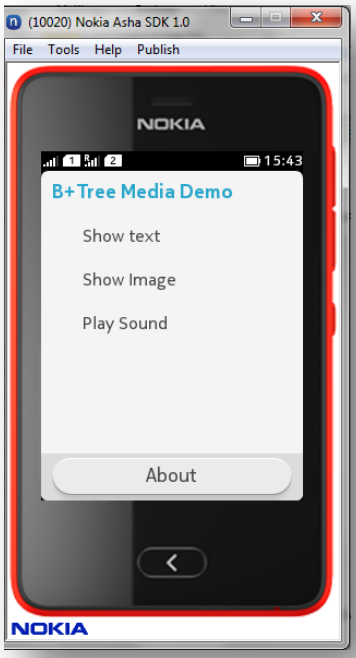
In the commandAction method the selection of the cache option is detected and the BTree is closed and reopened using the openBTree() method (see above) which will open the BTree according to the value selected.

```
if (type.equals("Select")) {
       boolean cacheToGoOn = (selectChoice.getSelectedIndex()==1);
       if (cacheToGoOn != cacheOn) {
              cacheOn = cacheToGoOn;
              try {
                     addressBTree.closeBTree();
```

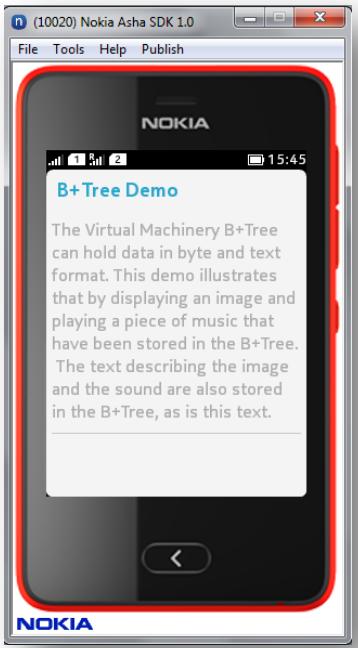
```
                openBTree(filePath);
            } catch (BTreeException e) {
                error("Error toggling cache "+e,10000);
            }
        }
}
```

BTreeMediaDemo (Web and Base License)

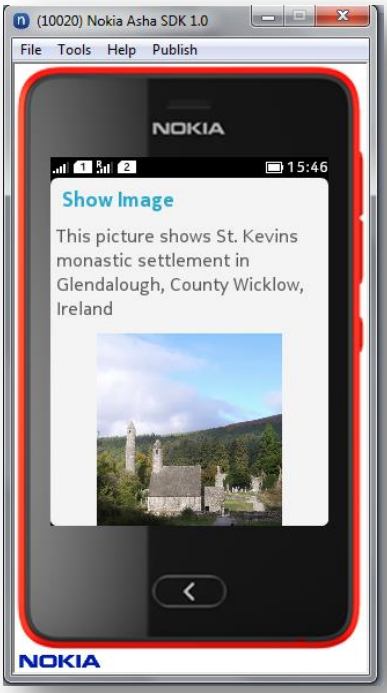
In the Asha development environment right-click on BTreeMediaDemo project and choose the 'Run As' option then select 'Java Emulated ME Midlet' option in the Run menu. The following screen will be displayed:



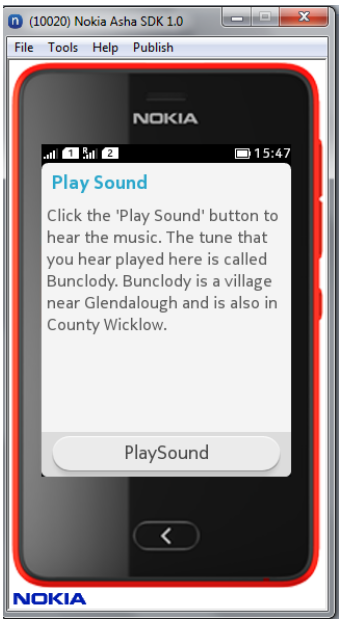
Selecting the 'Show Text' option will bring you to the following screen:



Pressing the back arrow under the image will bring you back to the main menu. You should now select the ‘Show Image ’ option which will cause the folllowing screen to be displayed:

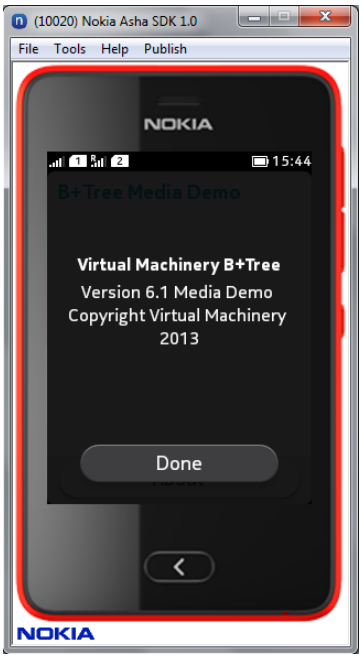


Pressing the back arrow under the image will bring you back to the main menu. You should now select the ‘Play Sound’ option which will display the following screen with a button labelled ‘Play Sound’ at the bottom of the screen.



Pressing the ‘Play sound’ button will cause the tune ‘Bunclody’ to be played (make sure that you have set the volume controls on your handset so that you can hear the tune. Pressing the back arrow under the image will bring you back to the main menu.

Pressing the ‘About’ button on the main menu will display details about the BTreeMediaDemo application:



The code behind the BTree Media Demo

The BTree Media demo uses a different approach to data storage compared to that in the BTree Address demo. In the case of the Web version of the demo the data is stored as part of the MIDlet application and is distributed with it. This makes referencing the data set easily as it is simply a case of using the relative addresses of the files in the jar:

```
protected void startApp() throws MIDletStateChangeException {
        display.setCurrent(fmMain);
        try {
                mediaBTree = BTreeMEROJar.openExistingTree("/MEDIA.DAT","/MEDIA.IND");
        } catch (BTreeException e) {
              displayErrorAlert( "Error opening B+Tree",e);
        }
}
```

In the case of the version distributed with the product a number of alternative storage approaches are offered – you can find out more about these in the section following this ('Alternative storage approaches offered in the Product media demo').

When we are building the 'Show Text' screen we call the 'showText()' method:

```
public void showText() {
        String answer =  getString("TEXT_KEY");
        TextBox textBox = new TextBox("B+Tree Demo",answer, answer.length(),
TextField.UNEDITABLE);
       textBox.addCommand(BACK_COMMAND);
    textBox.setCommandListener(new BTreeDemoCommandListener());
    display.setCurrent(textBox);
    currentMenu = "ShowText";
}
```

This calls the getString method which is a convenience method used to retrieve String data from the B+Tree using a String key. The important method here is the 'get' method of the B+Tree:

```
public String getString(String index) {
        try {
                return mediaBTree.get(index);
        } catch (BTreeException e) {
              displayErrorAlert( "Can't access B+Tree data at index "+index,e);
        }
        return "";
}
```

When we are building the 'Show Image' screen we call the showImage() method:

```
public void showImage() {
        imageForm = new Form("Show Image");
        imageForm.addCommand(BACK_COMMAND);
        imageForm.setCommandListener(new BTreeDemoCommandListener());
        String imageTextValue = getString("IMAGE_TEXT");
        currentMenu = "ShowImage";
        imageForm.append(imageTextValue);
        imageItem = new ImageItem(null,getImage(),ImageItem.LAYOUT_CENTER,"Image not
found");
        imageForm.append(imageItem);
        display.setCurrent(imageForm);
}
```

As this screen contains both text and the image the text is retrieved by re-using the getString(..) method described above. The image is retrieved using the getImage() method:

```
protected Image getImage() {
	byte[] bytes = getBytesFromLibrary("PICTURE_KEY");
	Image image = scaleImage(Image.createImage(bytes, 0, bytes.length),160,240);
	return image;
}
```

This gets the bytes that constitute the image from the library using the getBytesFromLibrary(..) method described below. It then takes them and uses them to re-create the image using the Java Image class. Before displaying the image it scales it to fit using the image scaling method in the class (not described here as it doesn't really relate to the B+Tree).

```
public byte[] getBytesFromLibrary(String index) {
	String iString;
	try {
		iString = (String) mediaBTree.get(index);
		if (iString != null) {
			String[] base = splitRecord(((String) mediaBTree.get(index)),"^");
			int numRecords = Integer.parseInt(base[0]);
			int pixLength = Integer.parseInt(base[1]);
			byte[] pixels = new byte[pixLength];
			int start = 0;
			for (int i=0;i<numRecords;i++) {
				byte[] thisRec =
mediaBTree.get(mediaBTree.createRecord(index+"_"+i)).getValue();
				System.arraycopy(thisRec, 0, pixels, start, thisRec.length);
				start+=thisRec.length;
			}
			return pixels;
		}
		else {
			return null;
		}
	} catch (BTreeException e) {
	      displayErrorAlert( "Error gettng bytes from B+Tree",e);
	}
	return null;
}
```

The getBytesFromTree(..) method uses the "PICTURE_KEY" index to retrieve all the records associated with the index and concatenate them back into a byte array.

When we are building the 'Play sound' screen we call the soundForm() method:

```
public void soundForm() {
	soundForm = new Form("Play Sound");
	String soundTextValue = getString("SOUND_TEXT");
	soundForm.append(soundTextValue);
	soundForm.addCommand(PLAY_COMMAND);
	soundForm.addCommand(BACK_COMMAND);
	soundForm.setCommandListener(new BTreeDemoCommandListener());
	display.setCurrent(soundForm);
	currentMenu = "PlaySound";
}
```

This contains some text and the button to play the sound. The text is fetched by re-using the getString(..) method. The 'Play Sound' button is registered with the BTreeDemoCommandListener which listens for the button to be pressed then calls the playSound() method:

```
public void playSound() {
    try {
       InputStream is = new ByteArrayInputStream(getBytesFromLibrary("SOUNDWAV_KEY"));
       Player p = Manager.createPlayer(is, "audio/x-wav");
       p.start();
    }
    catch (Exception e) {
      displayErrorAlert( "Unable to play WAV data!",e);
    }
}
```

The playSound(..) method fetches the bytes required to recreate the sound from the B+Tree using the getBytesFromLibrary(..) method. It puts the bytes into a stream then plays them with the Audio player.

When the user exits from the application the B+Tree is closed using the closeBTree() method.

```
protected void exit() {
       try {
              mediaBTree.closeBTree();
       } catch (BTreeException be) {
             displayErrorAlert( "Error Closing B+Tree",be);
       }
       try {
              destroyApp(true);
       }
       catch (MIDletStateChangeException msce) {
              mainMenu();
       }
       notifyDestroyed();
}
```

Alternative storage approaches offered in the Product media demo

In the product demo a number of ways of retrieving data from the B+Tree are demonstrated. In each case the API and the data set format remains the same but the access to the data storage varies according to the type of B+Tree used.

As distributed (in the BTreeMediaDemoMIDLet code supplied with both the web and product versions of the BTree) the BTree is stored in the MIDlet jar and is accessed by a relative path (as described above). This uses the BTreeMEROJar class (which meets the BTreeROInterface interface).

In the version supplied with the Base License of the product the following additional code is available:

In the definition of the class variables the following are defined:

```
public static final int STANDARD_BTREE = 0;
public static final int RMS_BTREE = 1;
public static final int JSR75_BTREE = 2;
public static final int IN_MEM_BTREE = 3;
public static final int JAR_BTREE = 4;
```

In the openBTree(…) method the type is passed as a parameter to determine which type of B+ tree is used to open the dataset. You can find further details of each of these B+ tree types in the development guides but for the moment all you need to know is that each of these types meets the BTreeRO interface and can be used to open the same dataset:

```
public void openTree(boolean cache, int type) throws BTreeException {
	if (type == JSR75_BTREE) {
		//Make sure that the MEDIA.DAT and MEDIA.IND files are in the photos
directory
		String path = System.getProperty("fileconn.dir.photos"); // for example:
file:///C:/Data/Images/
		mediaBTree = BTreeMEJSR75RO.openExistingTree(path+"MEDIA");
	}
	if (type == RMS_BTREE) {
		String path = "";
		mediaBTree = BTreeMERMSRO.openExistingTree(path+"MEDIA");
	}
	if (type == IN_MEM_BTREE) {
		//Make sure that the MEDIA.DAT and MEDIA.IND files are in the photos
directory
		String path = System.getProperty("fileconn.dir.photos");
		mediaBTree =
BTreeMEROInMem.openExistingTree(path+"MEDIA.DAT",path+"MEDIA.IND");
	}
	if (type == JAR_BTREE) {
		//Make sure that the MEDIA.DAT and MEDIA.IND files are in the photos
directory
		mediaBTree = BTreeMEROJar.openExistingTree("/MEDIA.DAT","/MEDIA.IND");
	}
	if (type == STANDARD_BTREE) {
		//Make sure that the MEDIA.DAT and MEDIA.IND files are in the photos
directory
		String path = "";
		mediaBTree = BTreeMERO.openExistingTree(path+"MEDIA");
	}
}
```

The initial distribution uses the JAR_BTREE type – you can experiment by modifying the code to use to use the different types of tree. Note that not all types of B+Tree will be supported by all platforms.

The jars required to support each type of tree are supplied in the project that comes with the base license.

Android B+Tree Demos

These demos have been packaged for use with the Android SDK and development environment and are therefore Eclipse projects. The code is in the src directory and the B+Tree library is in the root directory of the project.

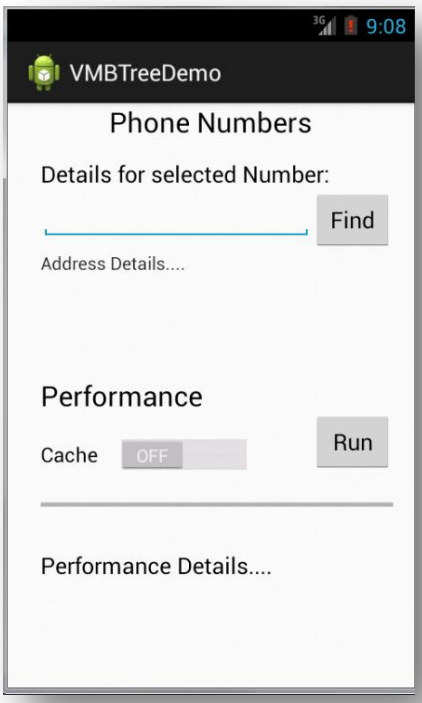
Demos are provided that illustrate how to use the standard B+Tree and B+Tree read only libraries in Android and also how to use the Android-specific B+Tree read only library

The following classes are provided in the demo –

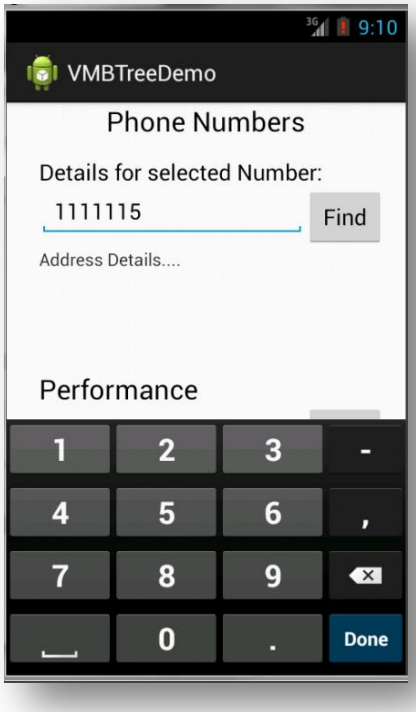
- BTreeAddressDemo.java
- BTreeMediaDemo.java

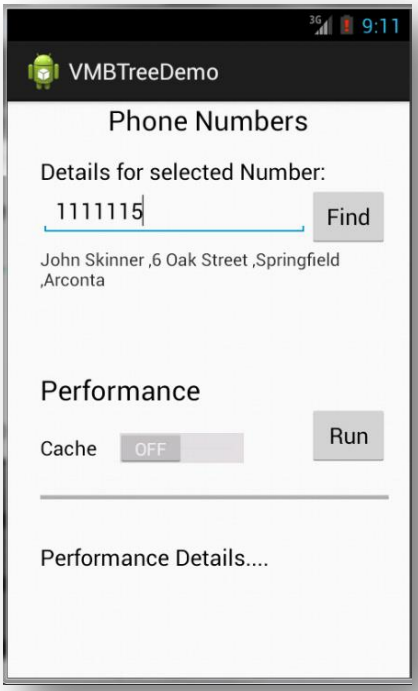
The B+Tree Address Demo (Web and Base License)

If you have a device (or a simulator) you can start the demo by deploying the BTreeAddressDemo.apk file in the root directory of the distribution. If you are using an Android development environment you can start the demo by running the BTreeAddressDemo project. In either case you will see the following screen –

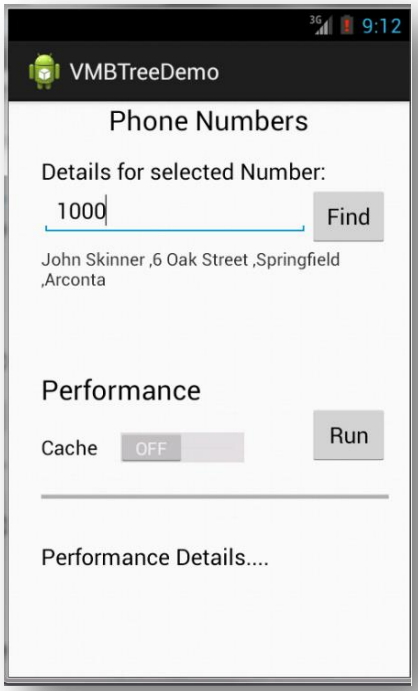


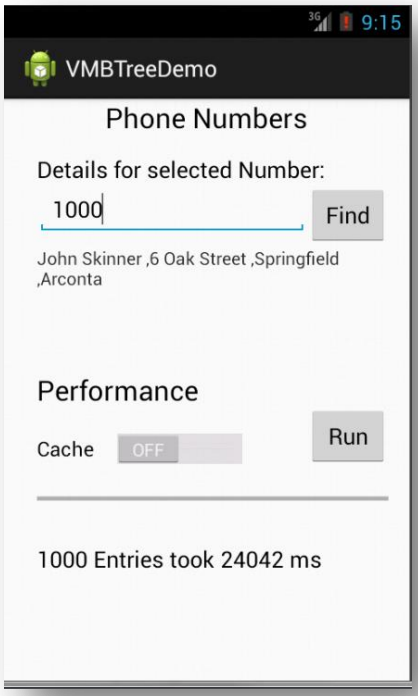
The address search: enter a value in the address search entry field. There are 10000 entries in the B+Tree starting at 1111111 so you can enter a number between 1111111 and 1121110 here. Then press the 'Search' button. The details associated with the search value will be displayed in the details field:



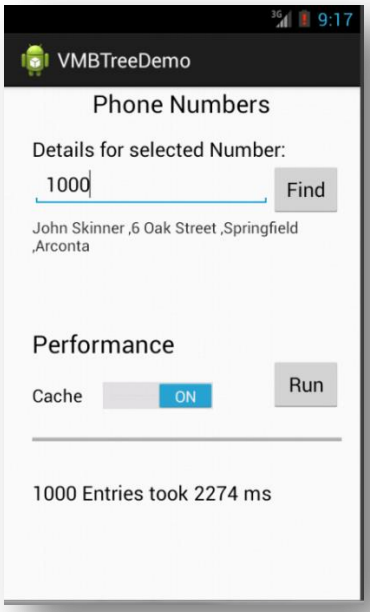


The performance test (cache off) : enter the number of addresses that you want to search for. This will generate that number of random searches. Leave the 'Use cache' check box empty. You should choose a small number of entries to begin with (say 1000) until you have some sense of the power of your machine. The time taken to carry out the searches will be displayed in the 'Performance Test Results' field:





The performance test (cache on) : enter the number of addresses that you want to search for. This will generate that number of random searches. Checke the 'Use cache' check box. You should choose a small nummber of entries to begin with (say 1000) until you have some sense of the power of your machine. The time taken to carry out the searches will be displayed in the 'Performance Test Results' field. As you can see the time taken to run the test was greatly reduced.



The code behind the B+Tree Address Demo

When the application starts the onCreate() method calls the openTreeNoCache() method:

```
protected void onCreate(Bundle savedInstanceState) {
		...
	openTreeNoCache()
		...
}

public void openTreeNoCache() {
	try {
		if (btree != null) {
			btree.closeBTree();
			btree = null;
		}
		btree =
BTreeRO.openExistingTree("VMPHONETESTDAT.mp3","VMPHONETESTIND.mp3",this);
	} catch (BTreeException e) {
		e.printStackTrace();
	}
}
```

As the code shows the files of the BTree dataset have been renamed as mp3 files. This is required because the files are packaged with the Android App. To ensure that the files are not compressed

(and therefore become unusable by the B+Tree) they must be renamed to a type that is not compressed during the application creation process.

When the user enters some text in the 'Details for Selected Number' entry field and clicks the 'Find'button the following code is called:

```
findButton.setOnClickListener(new View.OnClickListener() {
	public void onClick(View v) {
		String key = numberEntry.getText().toString();
		String result = null;
		try {
			result = btree.get(key);
		} catch (BTreeException e) {
			e.printStackTrace();
		}
		if (result !=null) {
			addressText.setText(result);
		} else {
			addressText.setText("No Details found");
		}
	}
});
```

The get(..) method is called on the BTree and the addressText field is populated with the result .

When the user enters some text in the 'Details for Selected Number' entry field and clicks the 'Run' button the following runTest() method is called:

```
public void runTest () {
	performanceText.setText("Running...");
	testComplete = false;
final Thread testThread = new Thread() {
	     public void run() {
		. . .
		int entries = Integer.parseInt(numberEntry.getText().toString());
		. . .
		for (int i=0;i<entries;i++) {
			randNo = ((int) (random.nextFloat()*max)) + start;
			if (btree.get(randNo+"") != null) {
				count++;
			}
			. . .
		}
		. . .
		performanceText.setText(resultString);
	}
}
```

The important coding point here is the use of the get(..) method with the calculated index which uses a random number.

If the value of the 'Cache' switch is changed then the B+Tree will be closed and reopened using the new setting. If the cache is switched off then the openTreeNoCache() method (described above) will be called. If the cache is switched on then the openTreeFullCache() method will be called:

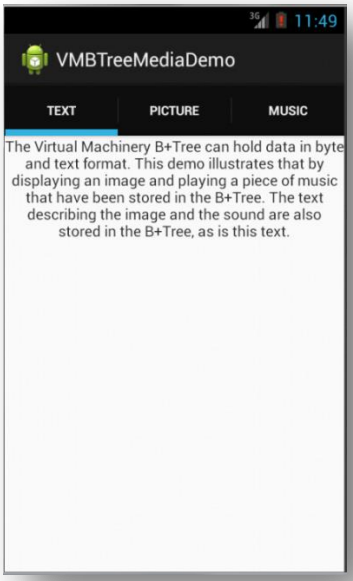
```
public void openTreeFullCache() {
	try {
		if (btree != null) {
			btree.closeBTree();
			btree = null;
```

```
                }
                btree =
BTreeRO.openExistingTree("VMPHONETESTDAT.mp3","VMPHONETESTIND.mp3",3,132, this);
                ((BTreeRO) btree).loadMem();
        } catch (BTreeException e) {
                e.printStackTrace();
        }
}
```

Note the two parameters for the index and data cache sizes, these ensure that the B+Tree will be fully cached. The loadMem() method populates the cache with the pages. If loadMem() is not called then the cache will be populated as each page (index or data) is used.

The B+Tree Media Demo (Web and Base License)

If you have a device (or a simulator) you can start the demo by deploying the BTreeMediaDemo.apk file in the root directory of the distribution. If you are using an Android development environment you can start the demo by running the BTreeMediaDemo project. In either case you will see the following screen –

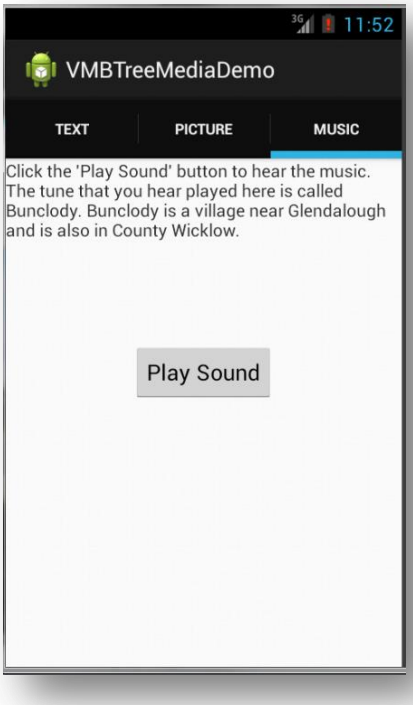


There are three tabs on the demo marked as ‘Text’, ‘Picture’ and ‘Sound’. As you can see above the screen opens on the ‘Text Demo’ tab which displays some information about the demo. All of the text on the tabs is stored in the B+Tree.

Clicking on the ‘Picture’ tab opens the following screen -

This shows a photographic image that has been stored in the B+Tree.

Clicking on the 'Music' tab opens the following screen -



If you click the 'Play sound' button you will hear a piece of music that has been stored in the B+Tree.

The code behind the BTreeMediaDemo

When the application starts the onCreate() method calls the BTree is opened:

```
protected void onCreate(Bundle savedInstanceState) {
	. . .
	aLibrary = BTreeRO.openExistingTree("MEDIA_DAT.mp3","MEDIA_IND.mp3",this);
	. . .
}
```

As the code shows the files of the BTree dataset have been renamed as mp3 files. This is required because the files are packaged with the Android App. To ensure that the files are not compressed (and therefore become unusable by the B+Tree) they must be renamed to a type that is not compressed during the application creation process.

As the application opens on the text tab the following code is run:

```
public static class TextFragment extends Fragment {
	public BTreeROInterface LIBRARY = null;

	public TextFragment(BTreeROInterface aLibrary) {
		LIBRARY = aLibrary;
	}

	public View onCreateView(LayoutInflater inflater, ViewGroup container,
		Bundle savedInstanceState) {
		TextView textView = new TextView(getActivity());
		textView.setGravity(Gravity.CENTER);
		String text = "empty";
		try {
			text = LIBRARY.get("TEXT_KEY");
		} catch (BTreeException e) {
			e.printStackTrace();
		}
		textView.setText(text);
		return textView;
	}
}
```

The important code here is the BTree get(..) call to the BTree that returns the text that is used to populate the screen.

When we move to the 'Picture' tab the following code is called:

```
public static class ImageFragment extends Fragment {
	public BTreeROInterface LIBRARY = null;

	public ImageFragment(BTreeROInterface aLibrary) {
		LIBRARY = aLibrary;
	}

	public View onCreateView(LayoutInflater inflater, ViewGroup container,
				Bundle savedInstanceState) {
		View view =  inflater.inflate(R.layout.image_layout, container, false);
		final TextView imagetext = (TextView) view.findViewById(R.id.imageText);
		try {
			imagetext.setText(LIBRARY.get("IMAGE_TEXT"));
```

```
                } catch (BTreeException e) {
                        e.printStackTrace();
                }
                final ImageView imageView = (ImageView) view.findViewById(R.id.imageView1);
                imageView.setImageBitmap(restoreImage("PICTURE_KEY",LIBRARY));
                return view;
        }
}
```

There are both text and image data to be displayed on this page. The text data is retrieved from the BTree using the get(..) method. The image is recreated using the restoreIMage(..) method:

```
public static Bitmap restoreImage(String index, BTreeROInterface aLibrary) {
        byte[] pixels = getBytes(index,aLibrary);
        if (pixels != null) {
                Bitmap bmp = BitmapFactory.decodeByteArray(pixels, 0, pixels.length);
                return bmp;
        }
        return null;
}
```

This method calls the getBytes(..) method to get the bytes that can be used to recreate the image. It then recreates a bitmap image using these bytes.

```
public static byte[] getBytes(String index, BTreeROInterface aLibrary) {
        String iString;
        try {
                iString = (String) aLibrary.get(index);
                if (iString != null) {
                        String[] base = ((String) aLibrary.get(index)).split("\\^");
                        int numRecords = Integer.parseInt(base[0]);
                        int pixLength = Integer.parseInt(base[1]);
                        byte[] pixels = new byte[pixLength];
                        int start = 0;
                        for (int i=0;i<numRecords;i++) {
                                byte[] thisRec =
aLibrary.get(BTreeROFactory.getInstance().createRecord(index+"_"+i)).getValue();
                                System.arraycopy(thisRec, 0, pixels, start, thisRec.length);
                                start+=thisRec.length;
                        }
                        return pixels;
                }
                else {
                        return null;
                }
        } catch (BTreeException e) {
                e.printStackTrace();
        }
        return null;
}
```

The getBytes(..) method uses the “PICTURE_KEY” index to retrieve all the records associated with the index and concatenate them back into a byte array.

When we move to the ‘Music’ tab the following code is called:

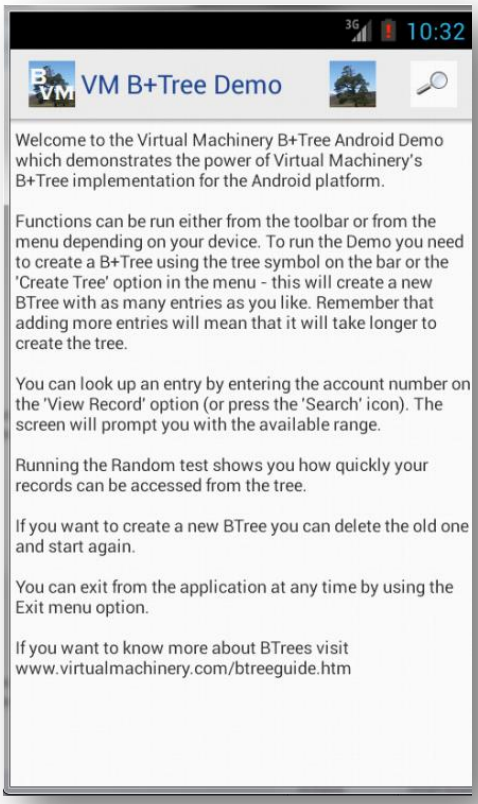
```
public static class SoundFragment extends Fragment {
	public BTreeROInterface LIBRARY = null;

	public SoundFragment(BTreeROInterface aLibrary) {
		LIBRARY = aLibrary;
	}

	public View onCreateView(LayoutInflater inflater, ViewGroup container,
		Bundle savedInstanceState) {
		View view =  inflater.inflate(R.layout.sound_layout, container, false);
		 final TextView soundtext = (TextView) view.findViewById(R.id.soundText);
		 try {
			soundtext.setText(LIBRARY.get("SOUND_TEXT"));
		} catch (BTreeException e) {
			e.printStackTrace();
		}
		final Button button = (Button) view.findViewById(R.id.soundButton);
		 button.setOnClickListener(new View.OnClickListener() {
		     public void onClick(View v) {
				byte[] soundbytes = getBytes("SOUNDWAV_KEY",LIBRARY);
				AudioTrack audioTrack = new
AudioTrack(AudioManager.STREAM_MUSIC,44100,AudioFormat.CHANNEL_CONFIGURATION_MONO,AudioForm
at.ENCODING_PCM_16BIT,soundbytes.length,AudioTrack.MODE_STATIC);
				audioTrack.write(soundbytes, 0, soundbytes.length);
				audioTrack.play();
		     }
		 });

		return view;
	}
}
```

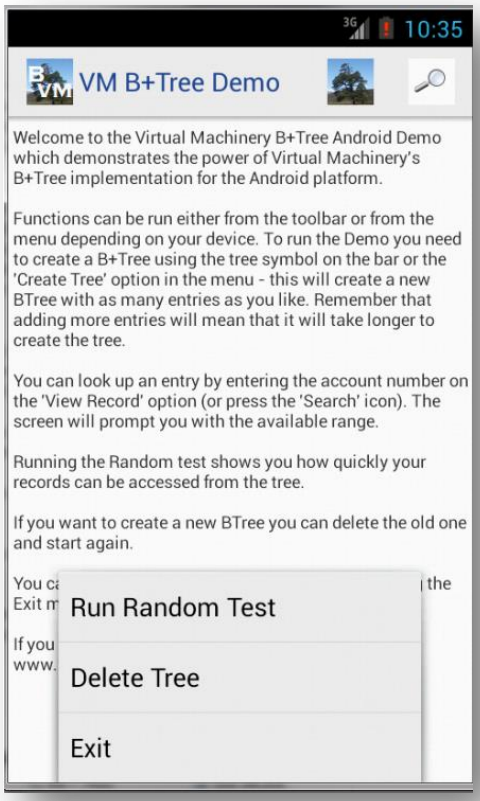
There are both text and sound data to be displayed on this page. The text data is retrieved from the BTree using the get(..) method. When the 'Play Sound' button is clicked the bytes used to recreate the sound are retrieved using the getBytes(..) method described above in relation to the image retrieval. The bytes are then played using an AudioTrack instance.

The B+Tree Read/Write Demo (Base License only)

This demo illustrates the use of the Virtual Machinery B+Tree library to create a B+Tree in an Android application.

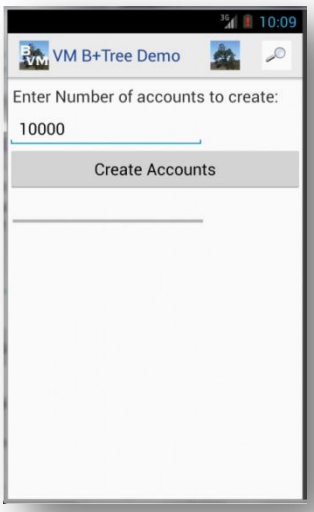
If you have a device (or a simulator) you can start the demo by deploying the BTreeDemoRW.apk file in the root directory of the distribution. If you are using an Android development environment you can start the demo by running the BTreeDemoRW project. In either case you will see the following screen (pressing on the menu button will cause the pop-up menu to appear as shown on the second screen –

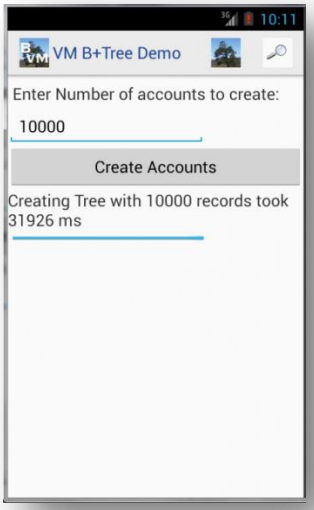




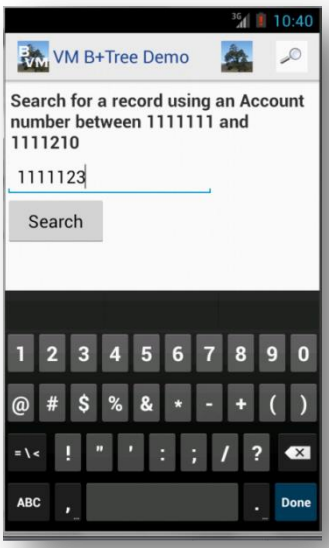
Before you can do anything you will have to create a BTree to do this click on the 'New Tree'icon:

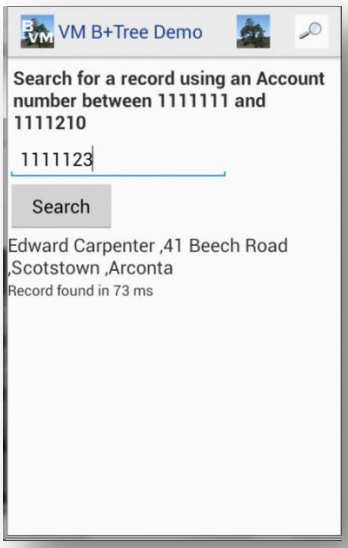
When you have done this the following screen will appear. You can enter the number of entries to be created, the thermometer bar will update as the entries are created and when the process is complete the time taken to carry out the operation will be displayed:



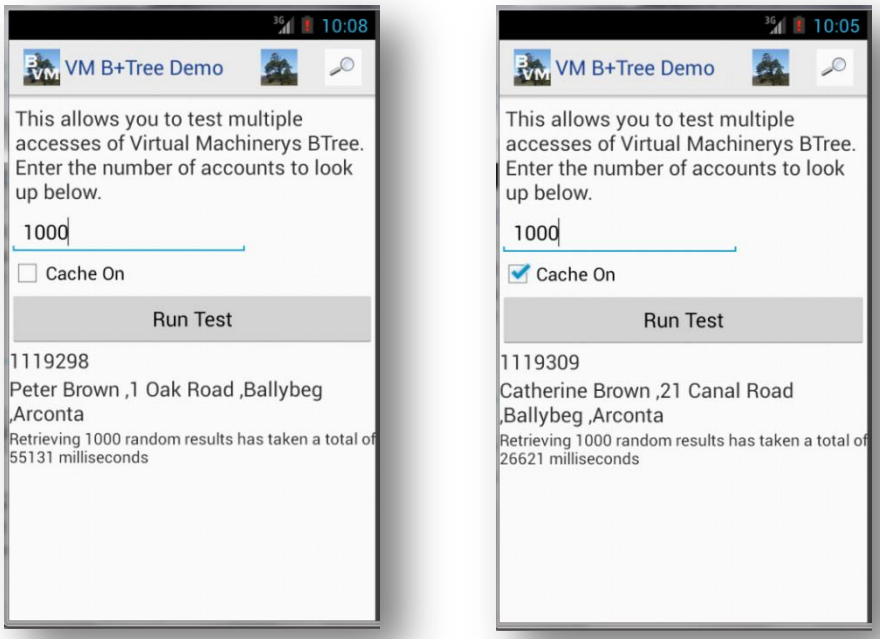


You can now use the B+Tree that has been created. To search for an entry in the B+Tree click on the search icon:  The following screen will be displayed. Enter the number of the account that you want to search for, press the 'Search' button and if the entry is found the details will be displayed :

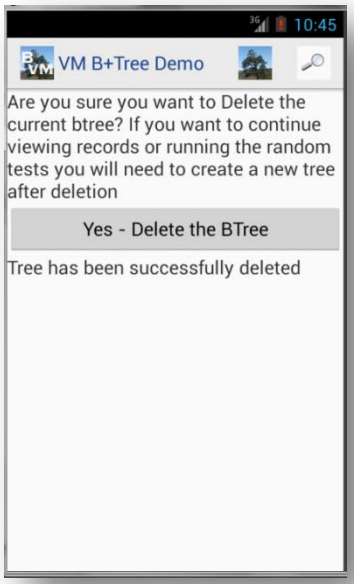




You can test the performance of the B+Tree by selecting the ‘Run random test’ option from the main menu. The following screen will appear. Enter the number of entries that you want to search for then press the ‘Run Test’ button. A corresponding number of random searches will then be made with the data for each search being displayed as it is retrieved. At the end of the test the time taken will be displayed. If you check the ‘Cache On’ check box and rerun the test you will see that it runs faster:



If you want to create a new tree you can delete the existing tree using the ‘Delete Tree’ option on the menu. Press the ‘Yes- Delete the BTree’ button:



You can exit the application by selecting the 'Exit' option on the main menu. The following screen will appear:

The code behind the B+Tree Read/Write Demo

In this example the standard Java BTree library is used to read and write data from and to the B+Tree. The B+Tree is stored in the "data" area of the android device. The code for the application is in the following packages inside the src directory of the BTreeDemoRW Eclipse project:

- com.virtualmachinery.btree.android.demo

  This contains the following classes:

  - BTreeDemoService.java - provides access to the functionality of the B+Tree
  - BuildRandomTree.java – creates the Random B+Tree
  - MainActivity.java – Co-ordinates the screens of the application
- com.example.android.actionbarcompat – this contains code that enables the app to run on multiple devices. It can be ignored for the moment

The MainActivity class initialises the opening text screen and the menu options. It also creates an instance of the BTreeDemoService class which carries out all the BTree related functions of the application. The first operation that must be carried out is to create the B+Tree – this is carried out by the createTree() method in the MainActivity class:

```
public void createTree() {
       path = getDir("data",Context.MODE_PRIVATE)+"/"+fileset;
       TextView numRecordsView = (TextView)findViewById(R.id.num_records_to_create);
```

```
        final TextView createView = (TextView)findViewById(R.id.num_created);
        final ProgressBar progress = (ProgressBar)findViewById(R.id.create_progress);
        demoService.createTree(numRecordsView, createView, progress, path);
    }
```

The path is set by looking up the “data” directory on the device. Then the createTree(..) method of the demo service is called. This creates an instance of the BuildRandomTree class which creates a B+Tree that contains the required number of records using its createPhoneTree(..) method :

```
public void createPhoneTree(long max,String aPath, ProgressBar progress) throws
IOException, BTreeException {
        path = aPath;
        if (max > 4500000) {dpsize=8192;}
        BTreeInterface  aTree = BTree.createNewTree(aPath+".DAT",aPath+".IND",100, 1,
ipsize,dpsize);
        long startphone = 1111110;
        String index;
        String record;
        final long interval = max/100;

        for (long i=0;i<max;i++) {
                if (i%interval == 0) {
                        progress.setProgress((int) (i/interval));
                }
                startphone++;
                index = startphone+"";
                record = generateRecord(); //Generate a random name and address
                aTree.put(index, record);
        }
        progress.setProgress(100);
        aTree.flushBTree();
        aTree.closeBTree();
}
```

The important points here are the creation of the new tree using the static createNewTree(..) method and the addition of records using the put(..) method. At the end of the method the tree is flushed and closed.

The findRecord() method of the MainActivity class calls the findRecord(..) method of the BTreeDemo Service instance:

```
    public void findRecord() {
        TextView keyView = (TextView)findViewById(R.id.search_key);
        TextView responseView = (TextView)findViewById(R.id.search_response);
        TextView timeView = (TextView)findViewById(R.id.response_time);
        demoService. findRecord(keyView, responseView, timeView);
    }

  public void findRecord(TextView keyView, TextView responseView, TextView timeView) {
        long start = System.currentTimeMillis();
         if (aTree != null) {
                String response = null;
                try {
                        response = aTree.get(keyView.getText().toString());
                } catch (BTreeException e) {
                        e.printStackTrace();
                }
                responseView.setText(response);
                long time = System.currentTimeMillis() - start;
                if (response != null ) timeView.setText("Record found in "+time+" ms");
                else  timeView.setText("No record found ("+time+" ms)");
         }
```

```
        else {
                timeView.setText("The Tree is currently empty - either no tree has been
created or the previous tree has been deleted - please create one");
        }
    }
```

Here you can see that the get(..) method of the BTree is called with the contents of the field that holds the index to be searched for.

When we run the random test we call the runRandomTest(..) method of the BTreeDemoService instance:

```
    public void runRandomTest(boolean cache) {
       final long max=
Integer.parseInt(((TextView)findViewById(R.id.num_in_test)).getText().toString());
       TextView progressText = (TextView)findViewById(R.id.test_progress);
       TextView accountText = (TextView)findViewById(R.id.account_number);
       TextView addressText = (TextView)findViewById(R.id.address_details);
       demoService.runRandomTest(max, cache,getDir("data", Context.MODE_PRIVATE) +"/"+
fileset, progressText, accountText,  addressText);
    }
```

This method re-opens the BTree using the openExistingTree(..) method according to whether the cache flag has been set or not and creates and looks up the required number of random indices keeping a count of the time spent getting the associated data from the B+Tree (note that the time taken to display the index and the value is not included in the time):

```
public void runRandomTest(final long max, boolean cache, String path, final TextView
progressText, final TextView accountText, final TextView addressText) {
       if (aTree != null) {
              try {
                     if (cache) {
                            aTree =
BTree.openExistingTree(path+".DAT",path+".IND",aTree.getLastIBlk(),0);
                            aTree.loadMem();
                     } else {
                            aTree = BTree.openExistingTree(path+".DAT",path+".IND");
                     }
              } catch (BTreeException bte) {
                     bte.printStackTrace();
              }

              . . . . . .

              new Thread(new Runnable() {
                     public void run() {
                            try {
                                   for (long i = 0;i<max; i++) {
                                          lastIndex = ""+(randomGenerator.nextInt((int)
numrecords)+1111111);

                                          randomTime -= System.currentTimeMillis();
                                          lastAnswer = aTree.get(lastIndex);
                                          randomTime += System.currentTimeMillis();
                                          randomResults++;
                                          aHandler.post(updateAccount);
                                          aHandler.post(updateAddress);
                                   }
                            }
                            catch (BTreeException be) {
                                   be.printStackTrace();
                            }
                            aHandler.post(updateRandomResults);
```

```
                        }
                }).start();
        }
        else {
                progressText.setText("The Tree is currently empty - either no tree has been
created or the previous tree has been deleted - please create one");
        }
    }
```

When we delete the BTree we call the deleteTree() method of the BTreeDemoService instance:

```
 public void deleteTree(String aPath) {
        File iTemp = new File(aPath+".IND");
        if (iTemp.exists()) {iTemp.delete();}
        File dTemp = new File(aPath+".DAT");
        if (dTemp.exists()) {dTemp.delete();}
        aTree = null;
    }
```

This method whether the index and data files of the dataset exist, and deletes them if they do.

iOS B+Tree Demos

These demos have been packaged for use with the iOS SDK and Xcode development environment and are distributed as XCode projects. The Library that is included in the demos downloaded from the website is a cut down version of the library and has been packaged for the iOS Simulator and will only work on the simulator. The versions of the demo that come with the full license will of use the full libraries that come with the license – there are two universal libraries one of which (the version of libVMBTreeRO.a which can be found in the..\libraries\iOS\VMBTreeRO\SimulatorUniversal directory) will work with the simulator and one of which (the version of libVMBTreeRO.a which can be found in the..\libraries\iOS\VMBTreeRO\Universal directory) will work with individual devices.

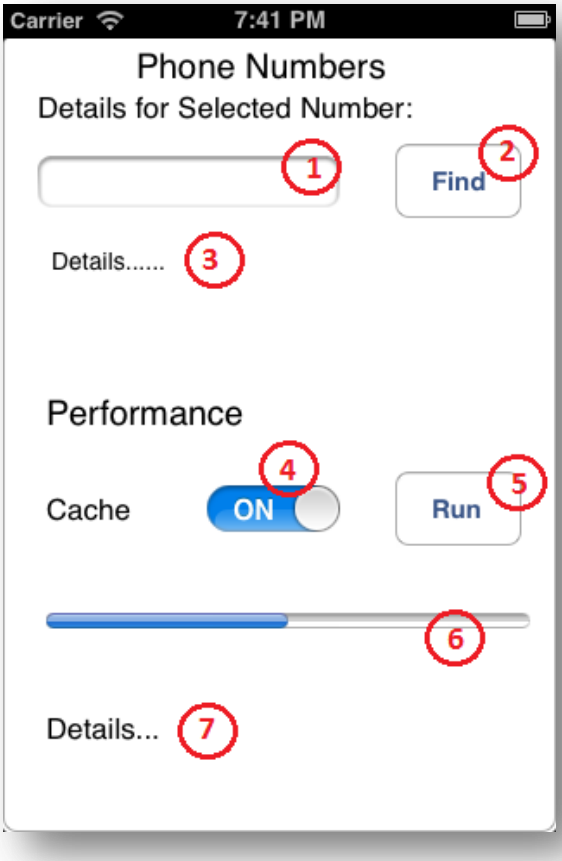
The following projects are provided to create the demo –

- BTreeAddressDemoWeb
- BtreeMediaWebDemo

Both demos are packaged as XCode projects. Two projects are provided and these are located in the ..\demoCode\iOS directory. To use them in XCode open the root directory of the distribution as an XCode project. You should check that the settings conform to your local environment then attempt to build and run.

The BTree Address Demo

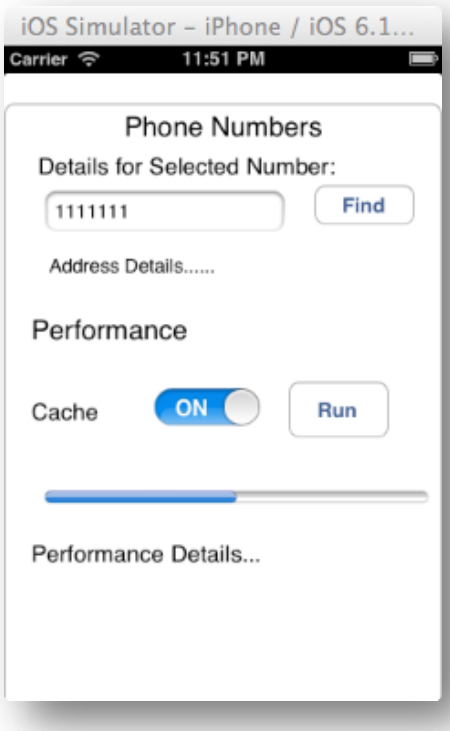
When you start the BTree Address Demo it looks like this -

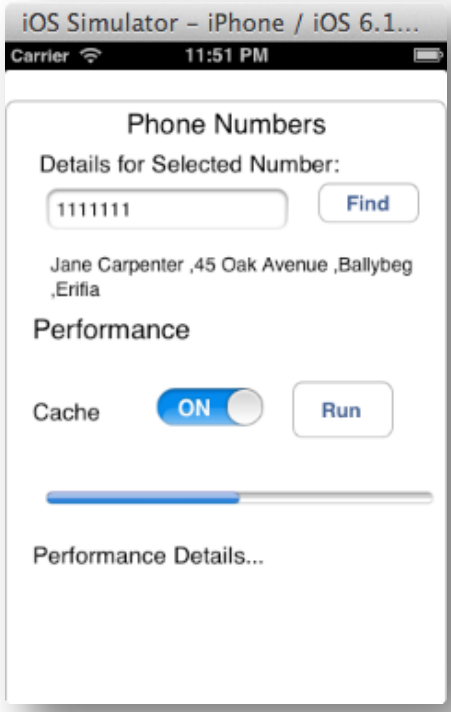


The various numbered parts of the screen are described below –

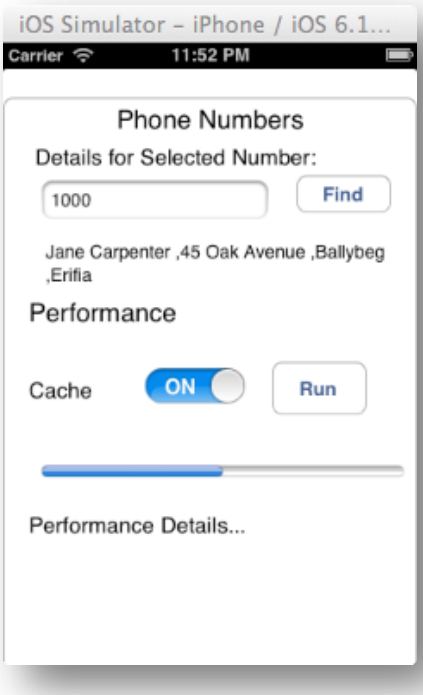
1. The address search entry field.
2. The 'Find' button – used to initiate the search on the B+Tree using the contents of the search entry field
3. The details field which will show the name and address details relating to the number entered in the address search field
4. The 'Cache' switch which switches the cacheing feature of the B+Tree off or on.
5. The 'Run' button which initiates the performance test
6. The progress bar that illustrates the current progress of the test
7. The 'Details' field which will show the results of running the performance test.

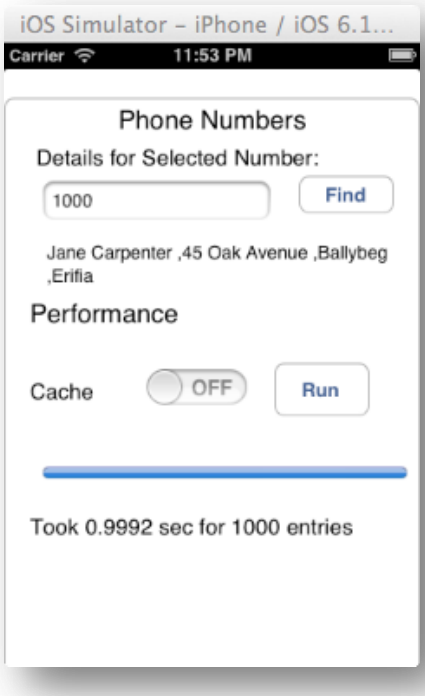
The address search: enter a value in the address search entry field. There are 10000 entries in the B+Tree starting at 1111111 so you can enter a number between 1111111 and 1121110 here. Then press the 'Search' button. The details associated with the search value will be displayed in the details field:



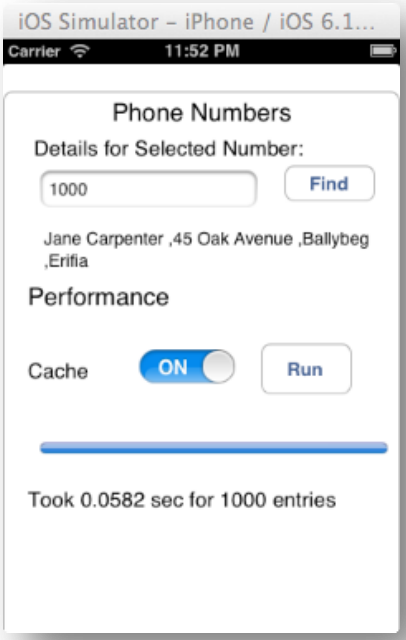


The performance test (cache off): enter the number of addresses that you want to search for. This will generate that number of random searches. Leave the 'Use cache' switch off. You should choose a small number of entries to begin with (say 1000) until you have some sense of the power of your machine. The time taken to carry out the searches will be displayed in the 'Performance Test Results' field:





The performance test (cache on) : Switch the 'Use cache' on and leave the number of entries to search for at 1000 then rerun the test and note the considerable improvement in performance:



The code behind the iOS BTree

Opening the tree: The BTree is opened using the following code (which you can find in VMViewController.m) :

```
- (VMBTreeRO*) openPhoneTreeWithIndexCache: (int) iCache andDataCache: (int) dCache {
    NSBundle *thisBundle = [NSBundle bundleForClass:[self class]];
    NSString *dataFilePath = [thisBundle pathForResource:@"VMPHONETEST.DAT" ofType:@""];
    NSString *indexFilePath = [thisBundle pathForResource:@"VMPHONETEST.IND" ofType:@""];
    self.store = [[VMBTreeRO alloc] initWithDataFile: dataFilePath indexFile: indexFilePath
dataCacheSize: dCache indexCacheSize: iCache];
    return self.store;
}
```

Searching for an entry: To search for a key in the BTree the get method is used. Since the key is passed as a String the getWithString: method that takes a String as a parameter is used. This also returns the result as a String. The String returned is placed in the address details field on the user interface.

```
- (IBAction)FindButton:(id)sender {
    if (self.store == nil) [self openStore];
    NSString *response = [self.store getWithString: self.numberEntryField.text];
    if (response == nil) {
        self.addressDetails.text = @"No Details found for this Number";
    } else {
        self.addressDetails.text = response;

    }
}
```

Switching the cache on and off: When the status of the cache (i.e. whether it is on or off) is changed the B+Tree is reopened with cacheing either enabled or disabled. Since there are 3 index pages and 132 data pages in the tree the tree is opened with all of these pages cached if the cache has been switched on. Here is the code that controls the opening of the B+Tree based on the status of the Cacheing switch:

```
- (void) openStore {
    self.store = nil;
    if ([[self cacheSwitch] isOn])  {
      self.store = [self openPhoneTreeWithIndexCache:3 andDataCache: 132];
    } else {
      self.store = [self openPhoneTreeWithIndexCache:0 andDataCache: 0];
    }
}
```

Running a performance test: The performance test finds the number of iterations to run from the number entry field on the dialog. For each iteration it creates a valid random number to look up in the B+Tree and updates the progress bar to show the proportion of entries that it has looked up. The code is as follows:

```
    int numEntries = [self.numberEntryField.text intValue];
    if (self.store == NULL)  [self openStore];
    int start=11111111;
    int max=10000;
    CFTimeInterval startTime = CFAbsoluteTimeGetCurrent();
    NSString *aString=@"";
    int count=0;
    [self performSelectorOnMainThread:@selector(updateProgressBar:) withObject: [NSNumber
numberWithFloat: 0.0 ] waitUntilDone: NO];
    for (int i=0;i<numEntries;i++) {
      @autoreleasepool {
        aString = [NSString stringWithFormat:@"%i",(start + (arc4random() % max))];
        if ([ self.store getWithString: aString]  != nil) {
           count++;
        }
        if (i !=0) {
          [self performSelectorOnMainThread:@selector(updateProgressBar:) withObject:
[NSNumber numberWithFloat: (float)i/(float)numEntries ] waitUntilDone: NO];
        }
      }
    }
    self.PerformanceDetails.text = [[NSString alloc] initWithFormat: @"Took %.4f sec for %d
entries",CFAbsoluteTimeGetCurrent() - startTime,numEntries];
    self.store = nil;
}
```

The BTreeAddressDemo files

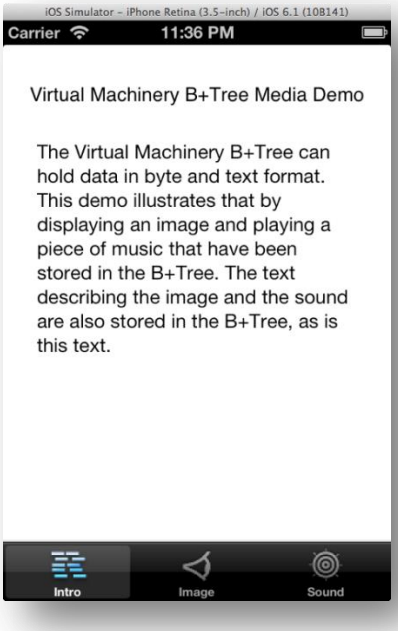
The following significant files are included in the iOS VMBTreeAddressDemoWeb project:

- Main.m – Startup program – creates and starts VMAppDelegate
- VMAppDelegate (.m and .h) – Files for code of main controller class of application
- VMViewController (.m and .h) – Files for code to create and handle main screen
- VMBTreeRO.h, VMBasicBTreeRO.h – Header files for iOS BTreeRO library
- libVMBTreeDemoRO.a – Library file for BTree demos (in the developer license this is replaced by libVMBTreeRO.a)
- VMPHONETEST.IND, VMPHONETEST.DAT – Files that constitute dataset of sample read only BTree.

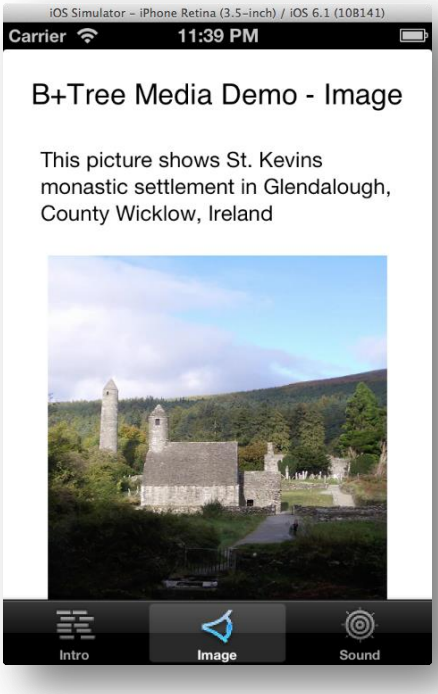
The BTree Media Demo

The BTree Media demo is a very simple app with three tabs that show text, an image a button that allows you to play a sound. The important thing to note here is that all of the data (text, sound and image) has been stored in a B+Tree.

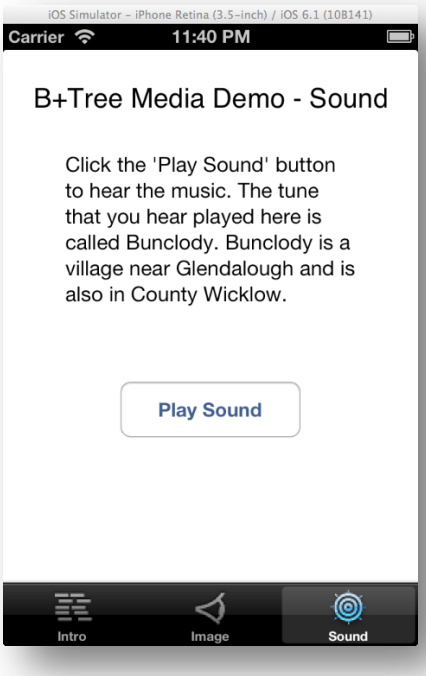
When you start the demo you see the following screen:



Pressing on the 'Image' tab brings up the following screen:



Pressing on the ‘Sound’ tab will bring you to the following screen. If you press the ‘Play sound’ button then you will hear a piece of music:



The code associated with each of the functions of the media demo is described below.

The main methods associated with the BTree are in the VMAppDelegate.m file:

The openMediaTreeWithIndexCache: andDatacache: method creates the paths to the two files (MEDIA.DAT and MEDIA.IND) that make up the read only BTree dataset and then opens the BTree.

```
- (VMBTreeRO*) openMediaTreeWithIndexCache: (int) iCache andDataCache: (int) dCache
{
    NSBundle *thisBundle = [NSBundle bundleForClass:[self class]];
    NSString *dataFilePath = [thisBundle pathForResource:@"MEDIA.DAT" ofType:@""];
    NSString *indexFilePath = [thisBundle pathForResource:@"MEDIA.IND" ofType:@""];
    VMBTreeRO *store = [[VMBTreeRO alloc] initWithDataFile: dataFilePath indexFile:
indexFilePath  dataCacheSize: dCache indexCacheSize: iCache];
    //Keys available are TEXT_KEY, PICTURE_KEY, SOUNDWAV_KEY, SOUNDMP3_KEY
    return store;
}
```

In order to store byte arrays larger than the maximum size of the data set you need to store the data in separate records and then reconstitute them afterwards when you retrieve them from the database. You can create your own way of doing this but the procedure that was used in this case is described in the section ‘How we created the B+Tree used in the media demo’ in the ‘VirtualMachinery B+Tree - Development Guide - Java Standard’ document. The getBytesWithKey: method is used to retrieve data stored this way:

```
-(NSData*) getBytesWithKey: (NSString*) aKey {
       NSString *str = [self.store getWithString: aKey];
```

```
        NSArray *split = [str componentsSeparatedByString:@"^"];
        int numrecords = [[split objectAtIndex: 0] intValue];
        int numbytes = [[split objectAtIndex: 1] intValue];
        unsigned char mediaBytes[numbytes];
        int start =0;
        for (int i=0;i<numrecords;i++) {
               NSMutableString* longKey = [NSMutableString string];
               [longKey appendString: aKey];
               [longKey appendFormat: @"_%i",i];
               VMBTreeRecord *aRecord = [self.store createRecordWithString: longKey];
               VMBTreeRecord *imageRecord = [self.store get: aRecord];
               unsigned char *data = [imageRecord byteValue];
               int lth= (int) [imageRecord length];
               memcpy(mediaBytes+start,data,lth);
               start+=lth;
        }
    NSData *mediaData = [NSData dataWithBytes: (const void *) mediaBytes length:
numbytes];
        return mediaData;
}
```

The code for the text tab is found in the file VMFirstViewController.m and is as follows:

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
    VMAppDelegate *appDelegate = (VMAppDelegate*)[[UIApplication sharedApplication]
delegate];
    NSString *value = [appDelegate getStringWithKey:@"TEXT_KEY"];
    infoText.text= value;
}
```

The important line here is the call to the getStringWithKey: method of VMAppDelegate. This method uses the instance of VMBTreeRO associated with the VMAppDelegate instnace to retrieve the data held at the key “TEXT_KEY” as a string :

```
-(NSString*) getStringWithKey: (NSString*) aKey {
        return [self.store getWithString: aKey];
}
```

The code for the image tab is found in the file VMSecondViewController.m and is as follows:

```
- (void)viewDidLoad {
    [super viewDidLoad];
       // Do any additional setup after loading the view, typically from a nib.
    VMAppDelegate *appDelegate = (VMAppDelegate*)[[UIApplication sharedApplication]
delegate];
    self.startImage = [appDelegate getImageWithKey:@"PICTURE_KEY"];
       self.imageView.backgroundColor = [UIColor blackColor];
       self.imageView.clipsToBounds = YES;
       self.imageView.image = startImage;
    NSString *value = [appDelegate getStringWithKey:@"IMAGE_TEXT"];
       imageText.text= value;
}
```

There are two BTree related calls here – the first is to the getImageWithKey: method of VMAppDelegate. This method uses the getBytesWithKey: method that we discussed earlier and then uses the bytes retrieved to create a UIImage instance:

```
-(UIImage*) getImageWithKey: (NSString*) aKey {
        NSData *imageData = [self getBytesWithKey: aKey];
        UIImage *image = [[UIImage alloc] initWithData: imageData];
```

```
        return image;
}
```

The UIImage instance returned by this method is then stored in the imageView associated with the VMSecondViewController .

The text associated with the image is then retrieved using the getStringWithKey: method that we discussed earlier.

The code for the image tab is found in the file VMSoundViewController.m and is as follows:

```
- (void)viewDidLoad {
    [super viewDidLoad];
    VMAppDelegate *appDelegate = (VMAppDelegate*)[[UIApplication sharedApplication]
delegate];
    NSString *value = [appDelegate getStringWithKey:@"SOUND_TEXT"];
    soundText.text= value;
}
```

This uses the getStringWithKey: method (which we discussed earlier) to retrieve the text that is used to describe the sound that user will play when they hit the 'Play Sound' button.

When the user presses the 'Play Sound' button the following code is executed:

```
-(IBAction) soundClicked: (id) sender {
    VMAppDelegate *appDelegate = (VMAppDelegate*)[[UIApplication sharedApplication]
delegate];
    NSData* data =  [appDelegate getBytesWithKey:@"SOUNDMP3_KEY"];
    _avPlayer = [[AVAudioPlayer alloc] initWithData: data error: nil];
    [self.avPlayer prepareToPlay];
    [self.avPlayer setDelegate: self];
    [self.avPlayer play];
}
```

The getBytesWithKey: method that we discussed earlier is used to retrieve the data that represents the sound – in this case the sound is in MP3 format. The sound is then played by sending the bytes to an instance of AVAudioPlayer.

The BTreeMediaDemo files

The following significant files are included in the iOS VMBTreeMediaDemo project:

- Main.m – Startup program – creates and starts VMAppDelegate
- VMAppDelegate (.m and .h) – Files for code of main controller class of application
- VMFirstViewController (.m and .h) – Files for code of text tab
- VMSecondViewController (.m and .h) – Files for code of image tab
- VMSoundViewController (.m and .h) – Files for code of sound tab
- VMBTreeRO.h, VMBTreeRecord.h – Header files for iOS BTreeRO library
- VMBTreeDemoRO.a – Library file for BTree demos (in the developer license this is replaced by libVMBTreeRO.a)
- MEDIA.IND, MEDIA.DAT – Files that constitute dataset of sample read only BTree.