

DEV04: Bringing the Big Screen to the Small Screen: Streaming Audio/Video for BlackBerry smartphones

November 11, 2009

Shadid Haque
Application Development Consultant

Streaming Media

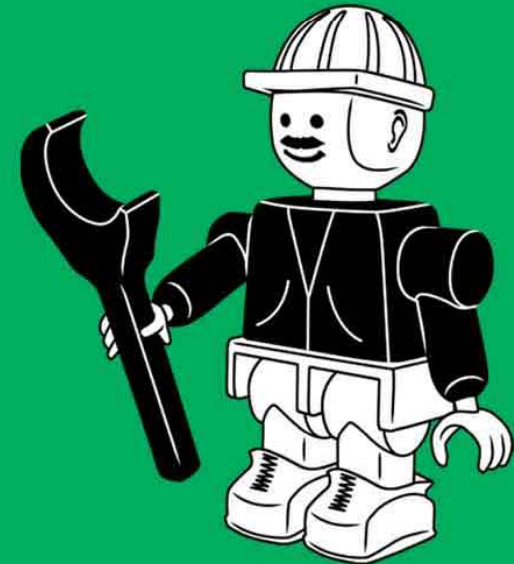
- Has a huge demand in the consumer space
- People love their TV and Radio
 - And they want it to-go
- Your next killer app?
- Exactly what the BlackBerry® Platform can do for you



Streaming Media and BlackBerry smartphones

Streaming Media over

- ▶ **RTSP**
 - HTTP using locator URL



Streaming Media over RTSP

- Real Time Streaming Protocol (RTSP)
 - Supported in v4.3.0 and above
 - EVDO
 - Supported in 4.5.0 and above
 - EVDO, EDGE, 3G and WiFi
 - Supported audio formats for streaming
 - AAC, AAC+, AMR-NB

Streaming Media over RTSP

- Using the BlackBerry Media Player

```
String url = "rtsp://mystreamingserver.com/001.aac";  
Browser.getDefaultSession().displayPage(url);
```

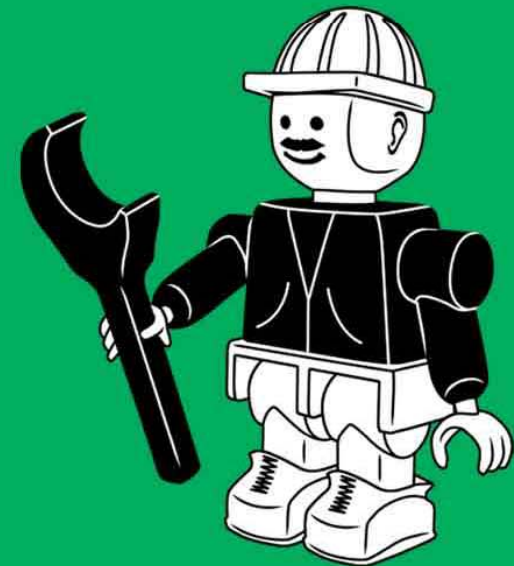
- Using Mobile Media API

```
String url = "rtsp://mystreamingserver.com/001.aac";  
Player player = Manager.createPlayer(url);  
player.start();
```

Streaming Media and BlackBerry smartphones

Streaming Media over

- RTSP
- ▶ **HTTP using locator URL**



Streaming Media over HTTP using locator URL

- Using the BlackBerry Media Player

```
String url = "http://mystreamingserver.com/001.mp3";  
Browser.getDefaultSession().displayPage(url);
```

- Using Mobile Media API

```
String url = "http://mystreamingserver.com/001.mp3";  
Player player = Manager.createPlayer(url);  
player.start();
```

Streaming Media over HTTP using locator URL

- Advantages
 - Extremely simple API calls
 - Hides many complex details
 - Doesn't require additional knowledge
 - It simply works!



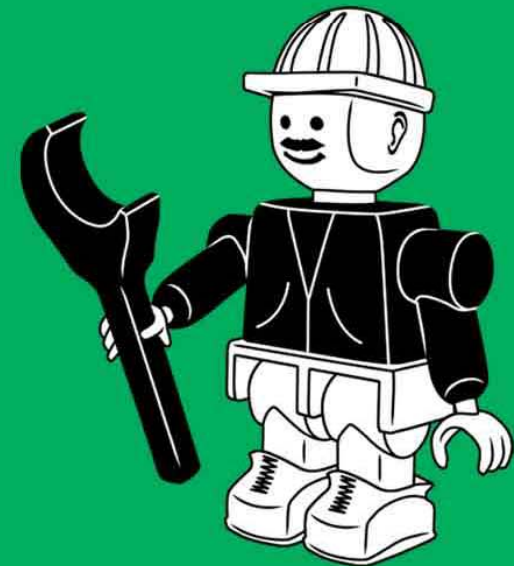
Streaming Media over HTTP using locator URL

- Disadvantages
 - No control over the
 - Buffer
 - Streaming parameters
 - No seek() support
 - No cache support
 - No preprocessing of data



Welcome to Advanced Streaming Media

- ▶ **Use custom DataSource**
 - Gain total control
 - Use the right Transport



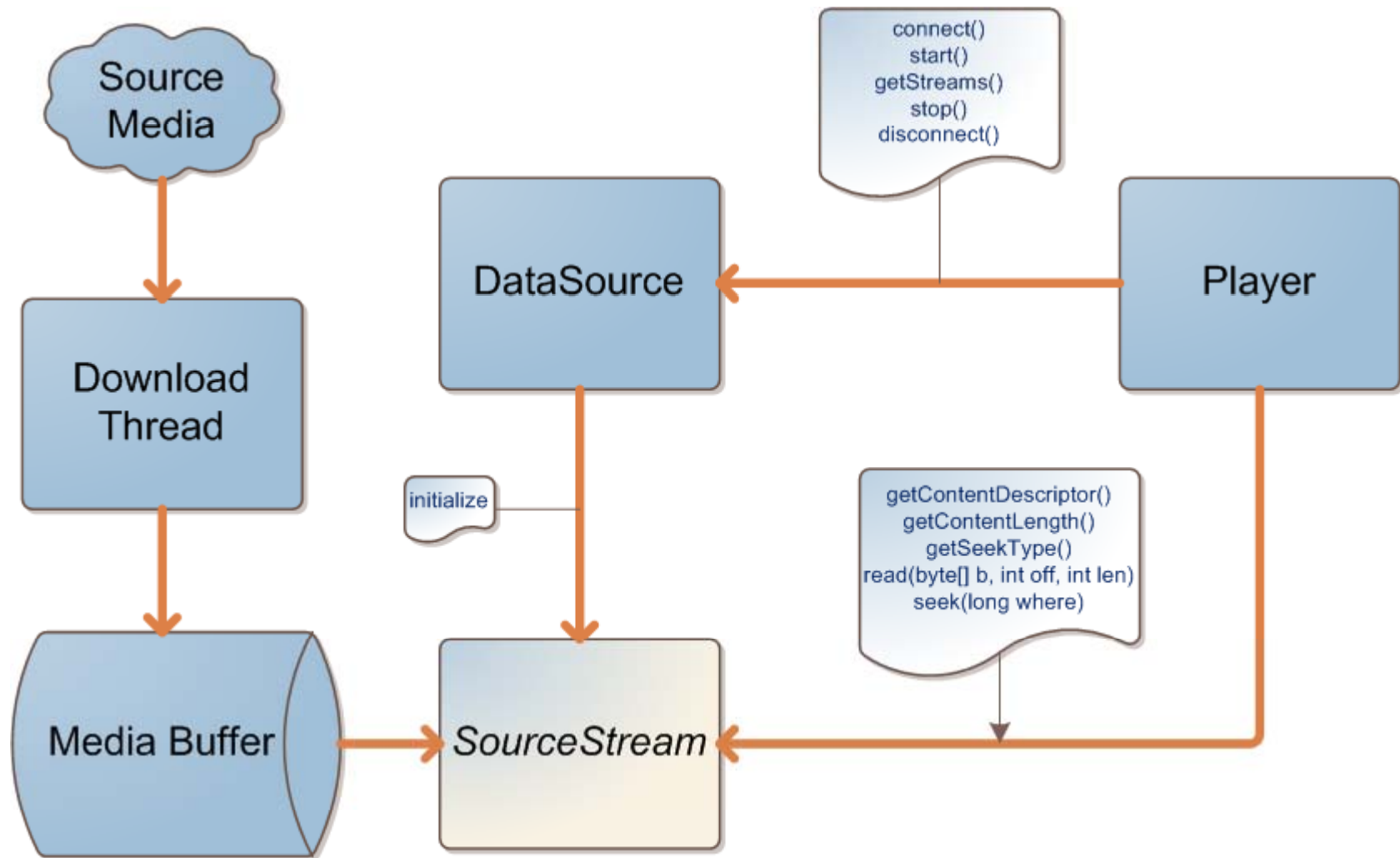
Streaming using custom DataSource

- Instantiating a Player from a DataSource

```
DataSource dataSource = new MyDataSource(String url);  
player = Manager.createPlayer(dataSource);
```



Streaming using custom DataSource



Streaming using custom DataSource

- Implement a DataSource
 - `public void connect()`
 - Open connections and I/O streams here.
 - `public SourceStream[] getStreams()`
 - Returns the SourceStream(s)
 - More on SourceStream later...
 - `public void start()`
 - Start downloading data
 - In a separate Thread!

Streaming using custom DataSource

- Implement a DataSource
 - public void disconnect()
 - Clean up connections, I/O streams
 - public String getContentType()
 - Return content type
 - public void stop()
 - Stop downloading data



Streaming using custom DataSource

- Implement a SourceStream
 - `public ContentDescriptor getContentDescriptor()`
 - Return a ContentDescriptor based on the content-type
 - `public long getContentLength()`
 - Return the content length
 - CDMA device hack. More on this later..
 - `public int getSeekType()`
 - Return `RANDOM_ACCESSIBLE`

Streaming using custom DataSource

- Implement a SourceStream
 - `public int read(byte[] b, int off, int len)`
 - Feeds data to Player
 - Perfect place to control feed
 - Block/resume feed
 - Streaming parameters? More on this later..
 - `public long seek(long where)`
 - Implementation varies based on connection
 - More on this later..

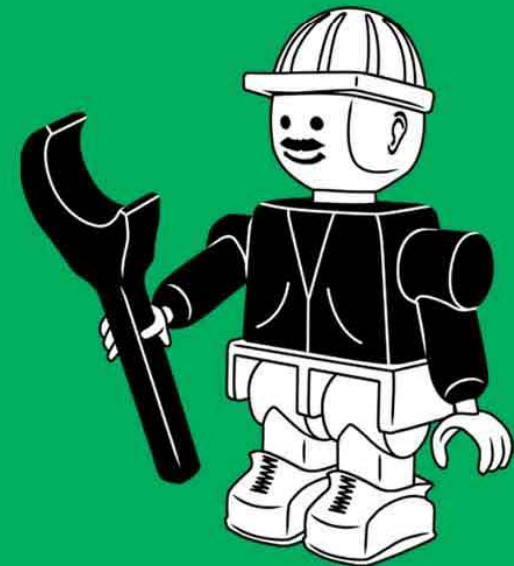
Streaming using custom DataSource

- Implement a SourceStream
 - public long tell()
 - Return the current position on the stream



Welcome to Advanced Streaming Media

- Use custom DataSource
- ▶ **Gain total control**
- Use the right Transport



SourceStream.getContentLength()

- **“Ideally” should return the content-length**
- **Device issues**
 - Seeks to random locations
 - Ahead of what’s downloaded
 - Impossible to do for HTTP streams
 - Return -1 instead
 - Stops the random seeking
 - setTime(), getMediaTime() and getDuration() will not work
 - Not required if streaming from FileConnection.

SourceStream.read(byte[] b, int off, int len)

- Player calls this to get chunks of data
 - Control the feed to Player
 - How?
- Introducing Streaming Parameters
 - initialBuffer
 - restartThreshold
 - bufferCapacity

```
SourceStream.read(byte[] b, int off, int len)
```

- **initialBuffer**

- Determines how much to buffer before playback starts for the first time
- There is no one perfect initialBuffer value
- Determine dynamically based on
 - Connection type
 - Bit rate
 - Bandwidth and latency

SourceStream.read(byte[] b, int off, int len)

- **restartThreshold**

- Determines when to resume a read call based on the data available in the buffer
- You don't want to resume immediately
 - Remember initialBuffer?
- There is no one perfect restartThreshold value
- Determine dynamically based on
 - Connection type
 - Bit rate
 - Bandwidth and latency

```
SourceStream.read(byte[] b, int off, int len)
```

- **bufferCapacity**

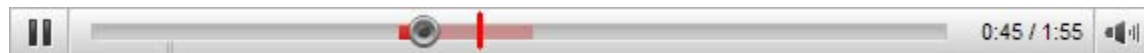
- This is the size of the buffer where data is stored
 - Before it's fed to the Player
- A buffer too small can frequently run out
- A buffer too big can waste valuable resources
- There is no one perfect bufferCapacity value
- Determine dynamically based on
 - Connection type
 - Bit rate
 - Bandwidth and latency

SourceStream.seek(long where)

- Called by Player to seek to a point
 - between 0 and the content-length
- For HTTP streaming
 - Honor this call only if the seek is feasible
 - Seek point is within the available data
 - Otherwise, return the current position.
- For FileConnection streaming
 - Honor this call using mark(), reset() and skip()

Seek calls initiated by User

- Seek calls within available data
 - Simply call skip() on buffer

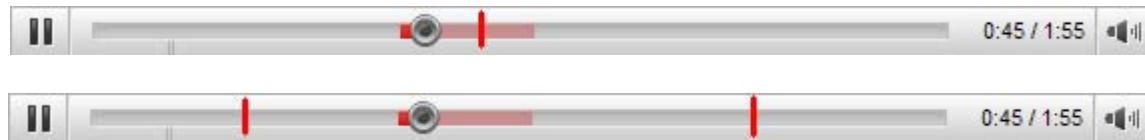


- Seek calls beyond what's available
 - Tear down HTTP connection
 - Clean up Player and other I/O resources
 - Reopen the connection using HTTP range header
 - Re-initialize the Player and other I/O resources



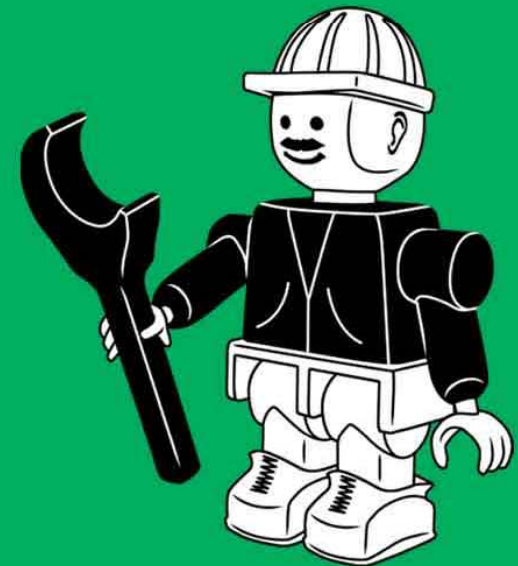
Seek calls initiated by User

- We know how to seek to a specific byte #
- But users don't understand bytes
 - They understand time
 - Convert the time point to a byte point
 - Using Bit rate
 - Then seek to that byte point.



Welcome to Advanced Streaming Media

- Use custom DataSource
- Gain total control
- ▶ **Use the right Transport**



Use the right Transport

- Use Wi-Fi[®], use Wi-Fi and use Wi-Fi!
- If Wi-Fi becomes available during a steaming session
 - Prompt the user for a switch
 - Reconstruct the streaming session over WiFi
- If Wi-Fi is not available
 - Use carrier transports (WAP2, TCP Cellular)
- Using BlackBerry[®] Enterprise Server/BlackBerry[®] Internet Service is not recommended!

Introducing the StreamingPlayer class

- Implements all that we talked about
- Hides the complex details
- MMAPI Player like interface
- Completely flexible
- Open Source!



StreamingPlayer

```
StreamingPlayer sp = new
    StreamingPlayer(url, "audio/mpeg");

// Optionally set streaming parameters,
// register StreamingPlayerListener etc.

sp.start();

// Seek within the stream and pause or
// close StreamingPlayer
```

StreamingPlayer

- **StreamingPlayer(InputStream is, String forcedContentType)**
 - Creates a new StreamingPlayer from an InputStream.
- **StreamingPlayer(String locator, String forcedContentType)**
 - Creates a new StreamingPlayer as per the locator String and forcedContentType.
- **StreamingPlayer(String locator)**
 - Creates a new StreamingPlayer as per the locator String.

StreamingPlayer

- **void realize()**
 - Creates the underlying Player object and calls realize() on it.
- **void prefetch()**
 - Puts this StreamingPlayer in the PREFETCHED state.
- **void start()**
 - Starts the playback by calling Player.start() and puts this StreamingPlayer in STARTED state.

StreamingPlayer

- **void stop()**
 - Pauses the playback by calling `Player.stop()` and puts this `StreamingPlayer` in `PREFETCHED` state.
- **void deallocate()**
 - Deallocates this `StreamingPlayer`.
- **void close()**
 - Closes this `StreamingPlayer` by closing connection to the media source.

StreamingPlayer

- **long skipBytes(long n)**
 - Allows the playback to skip an arbitrary number of bytes from the current playback position.
- **void skipTime(long n)**
 - Skips n microseconds in the media stream.
- **long setMediaPosition(long where)**
 - Skips playback to 'where' byte which must be between 0 and total length of content.
- **void setMediaTime(long where)**
 - Skips playback to where microseconds.

StreamingPlayer

- Getters and Setters
 - `void increaseBufferCapacity(int percent)`
 - `void setContentType(String contentType)`
 - `void setInitialBuffer(int initialBuffer)`
 - `void setPauseThreshold(int pauseThreshold)`
 - `void setRestartThreshold(int restartThreshold)`
 - `int getBufferCapacity()`
 - `long getBufferContentLength()`
 - `int getConnectionType()`
 - `long getContentLength()`
 - `String getContentType()`

StreamingPlayer

- Getters and Setters
 - `Control getControl(String controlType)`
 - `Control[] getControls()`
 - `int getDuration()`
 - `int getInitialBuffer()`
 - `String getLocator()`
 - `int getPauseThreshold()`
 - `int getRestartThreshold()`
 - `int getState()`

StreamingPlayer

- Registering/Removing StreamingPlayerListener
 - `addStreamingPlayerListener(StreamingPlayerListener listener)`
 - `removeStreamingPlayerListener(StreamingPlayerListener listener)`

StreamingPlayerListener

- **void bufferStatusUpdated(long available)**
 - Invoked when the size of the buffer changes
- **void downloadStatusUpdated(long totalDownloaded)**
 - Invoked every time a new chunk is downloaded
- **void feedPaused(long available)**
 - Invoked when the feed to the underlying Player is paused
- **void feedRestarted(long available)**
 - Invoked when the feed to the underlying Player is resumed again.

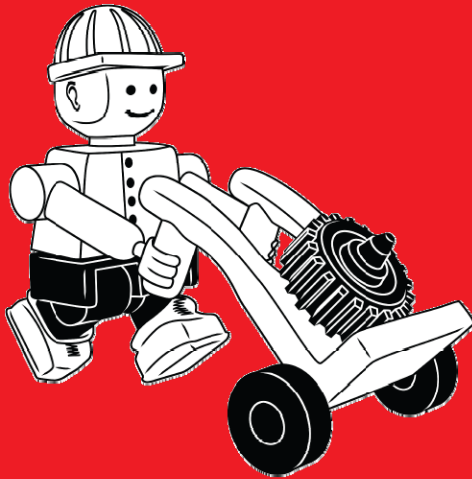
StreamingPlayerListener

- **void initialBufferCompleted(long available)**
 - Invoked only once when the buffer is filled with enough data to meet `StreamingPlayer.getInitialBuffer()` requirement
- **void playerUpdate(String event, Object eventData)**
 - Invoked when the underlying Player's `PlayerListener.playerUpdate(Player player, String event, Object eventData)` is called
- **byte[] preprocessData(byte[] bytes)**
 - Invoked for each chunk of data being downloaded before inserting the chunk in to the buffer

Session Surveys

Remember to complete your Breakout Session Evaluation in 1 of 3 ways:

- On your BlackBerry® smartphone – an email will automatically be sent to you
- On your BlackBerry smartphone - use the Developer Conference Mobile Conference Guide, from Sweet Caesar
- Login to Virtual DevCon at one of the Cyber Zone locations



Thank You

November 11, 2009

Shadid Haque
Application Development Consultant