



Developing Map Viewing Applications with J2ME and ESRI[®] Server-side Technologies

An ESRI Technical Paper • April 2004

Copyright © 2004 ESRI
All rights reserved.
Printed in the United States of America.

The information contained in this document is the exclusive property of ESRI. This work is protected under United States copyright law and other international copyright treaties and conventions. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage or retrieval system, except as expressly permitted in writing by ESRI. All requests should be sent to Attention: Contracts Manager, ESRI, 380 New York Street, Redlands, CA 92373-8100, USA.

The information contained in this document is subject to change without notice.

U.S. GOVERNMENT RESTRICTED/LIMITED RIGHTS

Any software, documentation, and/or data delivered hereunder is subject to the terms of the License Agreement. In no event shall the U.S. Government acquire greater than RESTRICTED/LIMITED RIGHTS. At a minimum, use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in FAR §52.227-14 Alternates I, II, and III (JUN 1987); FAR §52.227-19 (JUN 1987) and/or FAR §12.211/12.212 (Commercial Technical Data/Computer Software); and DFARS §252.227-7015 (NOV 1995) (Technical Data) and/or DFARS §227.7202 (Computer Software), as applicable. Contractor/Manufacturer is ESRI, 380 New York Street, Redlands, CA 92373-8100, USA.

ESRI, MapObjects, ArcIMS, the ESRI globe logo, www.esri.com, and @esri.com are trademarks, registered trademarks, or service marks of ESRI in the United States, the European Community, or certain other jurisdictions. Other companies and products mentioned herein are trademarks or registered trademarks of their respective trademark owners.

Developing Map Viewing Applications with J2ME and ESRI Server-side Technologies

An ESRI Technical Paper

Contents

What is J2ME?	1
Configurations	1
Profiles	1
Optional packages	1
Devices that support J2ME	1
Warnings	2
ESRI server-side technologies	2
MapObjects—Java	2
ArcIMS	2
MobileGIS application	2
Before you start	2
Goal	2
MobileGIS code	3
MapObjects—Java server code	5
ArcIMS server code	6
Running the application	8
What's next?	10
References	11

Developing Map Viewing Applications with J2ME and ESRI Server-side Technologies

This paper will discuss developing a simple map viewing application, written for the Java™ Micro Edition (J2ME) platform and targeted to run on mobile clients such as cell phones, PDAs, and other devices. The maps are generated using one of two ESRI Java server-side technologies: MapObjects®—Java Edition or ArcIMS®. A General Packet Radio Service (GPRS), LAN, or other connection is required for the device to connect to the server. This paper does not cover developing Location-Based Services (LBS) applications.

What is J2ME?

J2ME¹ is defined as 'the edition of the Java platform that is targeted at small, standalone or connectible consumer and embedded devices. The J2ME technology consists of a virtual machine and a set of application programming interfaces (APIs) suitable for tailored runtime environments for these devices. The J2ME technology has two primary kinds of components—configurations and profiles'.² The J2ME architecture defines configurations, profiles, and optional packages as elements for building complete Java runtime environments.

Configurations

Configurations are composed of a virtual machine and a minimal set of class libraries. These provide the base functionality for a particular range of devices that share similar characteristics. Currently, there are two J2ME configurations: Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC).

Profiles

Configurations must be combined with a higher level of APIs or profiles that further define the application life cycle model, the user interface, and access to device-specific properties. There are several profiles including Mobile Information Device Profile (MIDP), Foundation Profile (FP), Personal Profile (PP), and Personal Basic Profile (PBP).

Optional packages

The configurations and profiles can be further extended to offer APIs for using existing and emerging technologies such as Web services, wireless messaging, and database connectivity.³

Devices that support J2ME

All major cell phone manufacturers support the J2ME platform. At the time this paper was written, the most commonly found configuration and profile are CLDC 1.x and MIDP 1.x, respectively. Although MIDP 2.x has been released by Sun, very few devices support this profile. Sun's Web site lists the currently available configurations and profiles supported by the devices.⁸

- Warnings**
- Although Sun provides specifications, device vendors implement the configurations and profiles a little differently. An application may look and work as expected when developing using Sun's J2ME Wireless Toolkit, but it is recommended that the developer also test the application using the vendor's own wireless toolkit and device.
 - MIDP 1.x and 2.x do not support floating-point precision, thus there is a possible loss of precision. Developers need to handle this either on the client-side or server-side application.
 - Since most devices are constrained in processing power and memory, the application design must be simple.
 - Applications running in the emulators do not give an idea of the actual speed and working of an application and should be tested using a device.
 - Not all clients support JPEG, GIF, and PNG24 image formats; therefore, use PNG8 as the image output format on the server.

ESRI server-side technologies

MapObjects—Java MapObjects—Java Edition is a powerful collection of components used to build custom, cross-platform mapping applications. Both high-level and fine-grained components are available that perform a variety of spatial operations, which can be used in the presentation, Web, and server tiers. Since these are pure Java-based objects, they are deployable and can run on any platform that supports the J2SE 1.4 version and above. It can be embedded/integrated into an existing Web application and power the mapping functionality or be used as the center of a map-based Web application.

ArcIMS ArcIMS provides a standard platform to integrate, share, and exchange GIS data over the Internet. Users can integrate local data sources with Internet data sources for display, query, and analysis in an easy-to-use Web browser. It is highly scalable and uses an XML (ArcXML/AXL)-based communication model.

MobileGIS application

Before you start This paper assumes that the reader has some basic experience developing J2ME applications and using one or both ESRI server-side technologies used in this paper: MapObjects—Java Edition and ArcIMS.

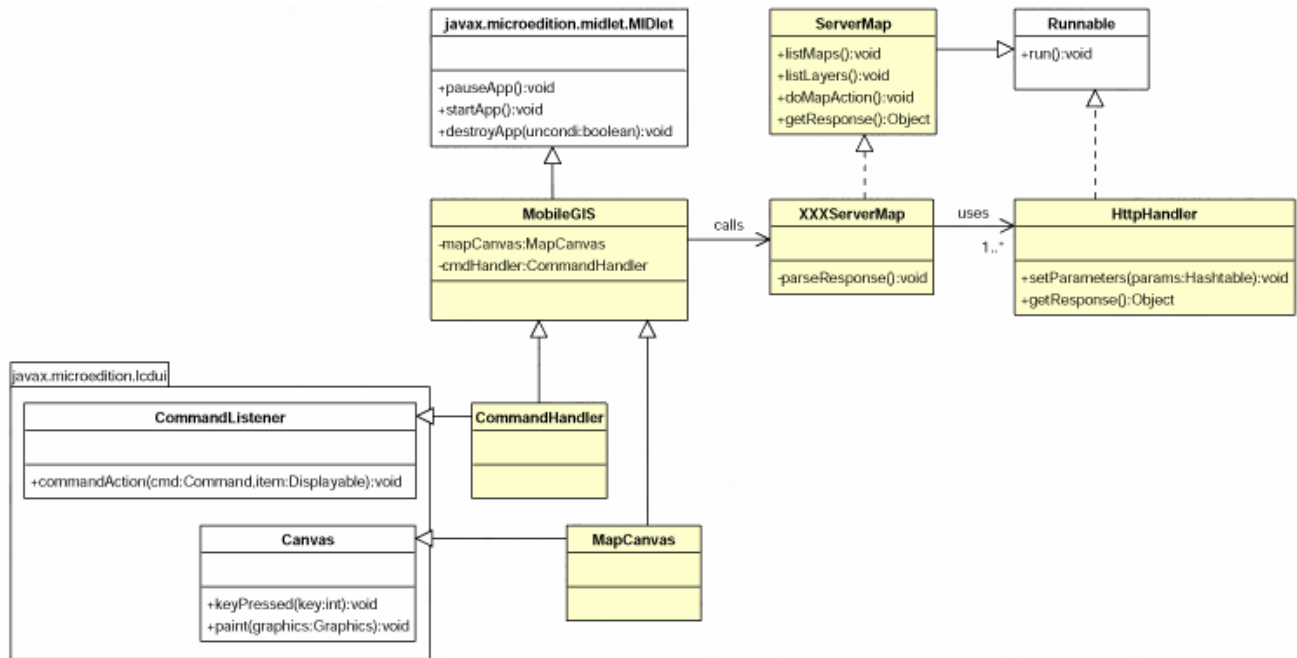
Goal The MobileGIS application lets users connect and view maps that are hosted on a server. Once connected, the client application shall support basic interaction, such as zooming and panning, with the map. This particular application will be targeted for use on mobile phones.

If using ArcIMS to serve maps, an ArcIMS server must be available, with image services that output maps in PNG8 format. You will write a simple JSP page that communicates with this server using the ArcIMS Java Connector.

If using MapObjects—Java, AXL files must be created. You will write a simple JSP page to interact with the map and output images. Creating ArcIMS services and MapObjects—Java AXL files is beyond the scope of this paper.

MobileGIS code The end user experience of the client will be the same whether or not an ArcIMS or MapObjects—Java server is used. The following code snippets lay down the design and common set of classes of the MobileGIS application.

The following UML class diagram shows the relation between the classes that are part of the MobileGIS application. Classes shown in yellow will be the focus of this paper.



MobileGIS.java This MIDlet class creates the user interface for the application and handles the user interaction. This class creates an instance of the appropriate class that implements the ServerMap interface.

```

public class MobileGIS extends MIDlet {
    ServerMap serverMap;

    public void startApp() {
        if (userSelection == MOJ)
            serverMap = new MOJServerMap();
        else
            serverMap = new ArcIMSServerMap();
    }
}

```

```

    }
    // Canvas to allow displaying of map image
    private class MapCanvas extends Canvas {
        public void paint(Graphics) {
            paint(mapImage);
        }
    }
    // Perform action on map.
    public void keyPressed(int) {
        serverMap.doMapAction(action);
    }
}

// Handle user interaction.
private class CommandHandler implements CommandListener {
    public void commandAction(Command, Displayable) {
        // Call appropriate serverMap method and update UI.
    }
}
}
}

```

ServerMap.java

This interface defines the behavior that is expected and used by the MobileGIS class. All implementing classes provide appropriate interaction with the server, based on calls made by the MobileGIS class.

```

public interface ServerMap extends Runnable {
    // List maps on server.
    public void listMaps();

    // List map layers.
    public void listLayers();

    // Do specified map action.
    public void doMapAction(int);

    // Get server response.
    public Object getResponse();
}

```

HttpHandler.java

This class handles communication via HTTP with the server. It is written as a thread so the application does not freeze when communicating with the server in the background.

```

public class HttpHandler implements Runnable {
    // Constructor to handle simple string-based communication.
    public HttpHandler(String, String)

    // Constructor to handle fetching of image from URL.
    public HttpHandler(String, int, int)

    // Set request parameters.
    public void setParameters(Hashtable)

    // Server response must be cast appropriately.
    public Object getResponse()
}

```


MapObjects—Java
server code

```

        // Communicate with server.
public void run() {
    // Connect to URL.
    // Write request.
    // Get response.
    //if (image)
        //fetch image from server
        //else
        //parse string
    }
}

```

Serverside
(*MobileMOJ.jsp*)

The bulk of the processing is handled on the server. For example, the creation of maps, image writers, and layer visibility are handled on the server. When a client requests to zoom and pan, depending on the client's current extent, the server computes the new extent to generate the map image.

```

Hashtable maps; // Collection of maps on server
Hashtable writers; // Image writers

if (request == mapList) {
    // Return list of maps from specific directory on
    // server.
}
else if (request == layerList) {
    if (maps == null) {
        // Initialize maps from axl.
        // Initialize image writers.
    }

    // Return list in specified map.
}
else {
    // Set current map state as specified in request.
    // Perform action on map.
    // Output map image.
    // Return image URL and map envelope.
}
}

```

Client
(*MOJServerMap.java*)

The communication model is a simple set of parameter key-value sets as part of the URL. The server interprets the request parameters and performs the appropriate action on the map and returns the URL to the generated map image. The client is then responsible for fetching the image from this URL.

```

public class MOJServerMap implements ServerMap {
    // Implement ServerMap methods.

    public void run() {
        HttpHandler conn = new HttpHandler(serverUrl, "");
        conn.setParameters(params);

        // Connect to server using HttpHandler.
    }
}

```

```

        response = parseResponse(conn.getResponse());

        if (getImage)
// Fetch image from server.
    }

// Parse String response from server.
private String[] parseResponse(String) {
// Parse response string.
}
}

```

ArcIMS server code

Serverside
(MobileArcIMS.jsp)

The server-side JSP simply redirects requests to the specified server using the ArcIMS Java Connector. The Java Connector allows an application to communicate with the ArcIMS application server. Due to limitations in the client-side XML parser, any '#' characters in the response are stripped before being sent back to the client. The state of the map is retained on the client.

```

// Make TCP connection to ArcIMS server.
return connection.send(request); // Return ArcXML.

```

Client
(ArcIMSServerMap.java)

The communication model in this case is ArcXML, which is a subset of XML used by ArcIMS. XML support is provided by kXML, a lightweight open source XML parser available for MIDP applications. Formatting of requests and parsing of response are handled by the client. The XML response is parsed and appropriate objects are created, which are then used for subsequent requests.

```

public class ArcIMSServerMap implements ServerMap {
// Implement ServerMap methods.

public void run() {
    HttpHandler conn = new HttpHandler(serverUrl,
    arcXMLRequest);
    conn.setParameters(params);
    parseArcXML(conn.getResponse());
}

// Parse XML response from server.
private void parseArcXML(String) {
    if (parsedXML == imageUrl)
        response = fetchImage();
    else
        response = parsedObject;
}

// A layer in an ArcXML map service.
private class Layer {
    String name;
    int id;
    boolean visibility;
}

private calculateFullExtentEnvelope() {

```

```
// Calculate envelope based on smallest (min) or largest //
(max).
    // Point of each layer's envelope
}

// The envelope/extents of the map service
private class Envelope {
    long minX, minY, maxX & maxY;
    int digits; // Digits after decimal point

private void stringToLong(String minx, miny, maxx, maxy) {
    // Convert String to long and remove decimal point.
}

private String longToString(long val) {
    // Return long as String with decimal point.
}

private void zoomIn() {
    // Compute zoomed in envelope.
}







private void zoomOut() {
    // Compute zoomed out envelope.
}

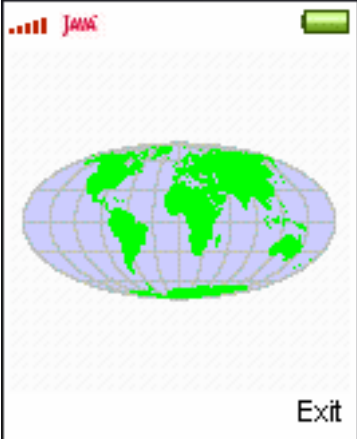

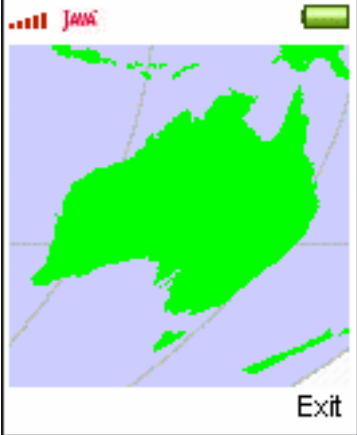

private void pan(int) {
    // Compute envelope after panning in specified
    // direction.
}
}
}
```

Running the application

The following screen captures show the application running on the Sony Ericsson T610 emulator.



	MOJServerMap	ArcIMSServerMap
Once the application is launched, the user can select whether to use a MapObjects—Java or ArcIMS server. Click List Maps to list the maps on this server.	 <p>Connect Servers <input type="radio"/> ArcIMS <input checked="" type="radio"/> MapObjects Java List Maps Exit</p>	 <p>Connect Servers <input checked="" type="radio"/> ArcIMS <input type="radio"/> MapObjects Java List Maps Exit</p>
The list of maps (MapObjects—Java) or services (ArcIMS) are displayed. After selecting a map, click List Layers to see the layer list.	 <p>Connect Servers <input type="radio"/> ArcIMS <input checked="" type="radio"/> MapObjects Java Maps <input type="radio"/> USA <input checked="" type="radio"/> World List Layers Exit</p>	 <p>Connect Servers <input checked="" type="radio"/> ArcIMS <input type="radio"/> MapObjects Java Maps <input checked="" type="radio"/> World <input type="radio"/> USA List Layers Exit</p>
Once the layers are listed, the user can select the visibility of the layers. Click Get Map to show World map with the cities layer turned off.	 <p>Connect MapObjects Java Maps <input type="radio"/> USA <input checked="" type="radio"/> World Layers <input checked="" type="checkbox"/> world30 <input checked="" type="checkbox"/> country <input type="checkbox"/> cities Get Map ↑ Exit</p>	 <p>Connect MapObjects Java Maps <input checked="" type="radio"/> World <input type="radio"/> USA Layers <input checked="" type="checkbox"/> Countries <input checked="" type="checkbox"/> Geogrid <input type="checkbox"/> Cities Get Map ↑ Exit</p>

	MOJServerMap	ArcIMSServerMap
The map is now displayed. The server respects any projection on the map and returns the map with the projection. The user can now use the keypad to zoom or pan the map.		
Several zooms and pans later, we can view a desired section of the map. Click Exit to quit the application.		

What's next?

- To make the application usable on PDAs and phones that do not have a keyboard, additional commands should be added to the MapCanvas. CommandListeners must be implemented to handle the event generated and serverMap.doMapAction called with appropriate action.
- The JSP sessionID cookie can be stored in the header information of the request that is sent by the client. This allows the server to use sessions to store the state of the map, and thus the communication between the server and client is reduced.
- Features can be added to the application if the application is targeted at heavier clients, such as smart phones, that have more processing power and memory.

References

- J2ME: <http://java.sun.com/j2me/>
- Symbian Glossary of Technical Terms:
<http://www.symbian.com/technology/glossary.html>
- Java 2 Platform, Micro Edition: <http://java.sun.com/j2me/j2me-ds.pdf>
- ESRI: <http://www.esri.com>
- MapObjects—Java Edition: <http://www.esri.com/software/mojava/index.html>
- ArcIMS: <http://www.esri.com/software/arcims/index.html>
- kXML: <http://kxml.enhydra.org/>
- J2ME device list: <http://java.sun.com/webapps/device/device>

