Vipera Inc
200 Page Mill Road
Palo Alto, CA 94306. USA
www.vipera.com
info@vipera.com

Vipera OTA Provisioning Server

Technical Overview

Version 1.0

The Vipera provisioning portal allows content providers to stock content repositories with J2ME application bundles, and to deliver those bundles to MIDP devices via WAP Push. The Vipera provisioning portal provides a complete and hosted solution for all aspects of J2ME content storage, content catalogization and delivery.

Revisions

Version	*Comments*
1.0	Initial revision.

References

Key	*Reference*
JSR 124	J2EE™ Client Provisioning Specification. `http://www.jcp.org/en/jsr/detail?id=124`
VIPERA	Vipera Mobile Network Operator - Vision Whitepaper `http://www.vipera.com/`
ADMIN-URL	Provisioning portal content providers login (HTML browser) `http://ota.vipera.com/admin`
CAT-URL	Provisioning catalog browsing (WML or HTML browser) `http://ota.vipera.com/`
ADMIN-URL	Provisioning repository administration `http://ota.vipera.com/admin`
PUSH-URL	HTTP-POST URL for triggering WAP-Push provisioning `https://ota.vipera.com/push`

1 What is Provisioning?

Over-the-air (OTA) provisioning is the process of downloading and installing J2ME content (MIDlets) on demand. For example, a mobile user would like to find an interesting game and download it to his or her mobile phone. Or a corporation would like to install a new application version to the mobile device of a certain employee.

The piece of software infrastructure making provisioning possible is called a *provisioning server*. Provisioning servers are often compared with vending machines, allowing end users to pick applications from a list of software choices.

If users will simply download a JAD and JAR file, then a web server such as Tomcat, Apache or IIS will probably suffice. However, if a content provider wants to deploy a J2ME application to different device types, and provide application bundles optimized for various device screen sizes or other device capabilities, then a provisioning server is a much better alternative. JSR 124 [JSR 124] is the architecture for such a server.

1.1 Why bother?

But why bother at all with J2ME application installation and provisioning? Aren't WAP, XHTML, SMS and MMS all we need for making mobile users happy? Indeed not. Applications deployed on the client can offer a richer set of features to both developer and end user, and have the added benefit of being useful even when not connected to the network. However, updates are more difficult because there are so many different mobile device platforms. A centrally managed repository of content and applications solves the problem of deploying to a wide variety of devices.

2 Vipera Provisioning Server – Concepts

The Vipera provisioning server is a *hosted* JSR 124 server, that's why we also call it the *provisioning portal*. No special software needs to be installed by content providers in order to take advantage of Vipera provisioning server. Instead, content providers obtain a user name and password for logging into the Vipera provisioning server. Via an HTML web interface, content providers can create their project repositories, upload JSR 124 bundles, deliver bundles to end users via WAP Push, monitor incoming download requests, etc.

The Vipera server is for delivering J2ME applications, regardless of whether those applications are Vipera enabled or not. Provisioning of other type of content (ring tones, wallpapers, etc.) is currently not supported.

2.1 Provisioning Use Cases

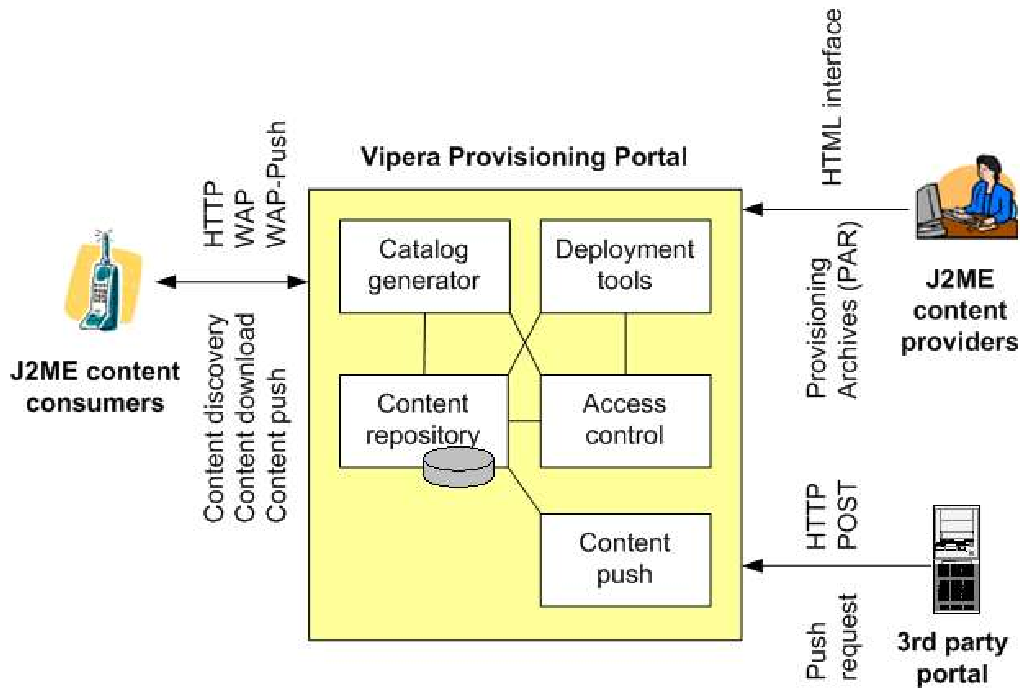


Illustration 1: Architecture and Use Cases

Agents	*Use Cases*
Content consumers (end users)	Browse online catalog of available application bundles (using HTML or WML browser)
	Pick an application bundle
	Download application bundle optimized for device type (via browser on the device)
	Invite other users to try out the application
Content providers (ISVs or J2ME developers)	Request creation of account on Vipera provisioning server
	Log into Vipera provisioning server account
	Create project repository
	Upload JSR 124 provisioning archive (PAR) into repository
	Deliver application bundles to mobile phones, via WAP Push
	Analyze HTTP headers of download requests
	Optimize deployment descriptor of provisioning archive
3rd party portal	Trigger delivery of an application bundle to a mobile user (via HTTP POST request)
	Customize the bundle on-the-fly, by adding JAD key/value pairs, Vipera address, Vipera password etc. to the application being downloaded

Table 1: Use Cases.

2.2 Provisioning Process

The provisioning process can be broken down into three tasks:

- *Stocking*: managing the repository; adding and removing J2ME application bundles
- *Discovery*: finding out what bundles are available for delivery from the provisioning server
- *Delivery*: delivering the application bundle to the client.

2.2.1 Stocking

Stocking is the process of a content provider uploading a J2ME *provisioning archive* (PAR) to the portal. A provisioning archive is a ZIP archive bearing the suffix `.par`, it includes one ore more *client bundles*, plus a deployment descriptor named `provisioning.xml`.

A client bundle consists of one MIDlet, along with icons etc. that can be used by the provisioning server to display the bundle in online catalogs. A simple PAR file might have a directory structure like this:

```
META-INF/provisioning.xml
/HelloWorld.jar
/HelloWorld.jad
/HelloWorld_icon.gif
/COPYRIGHT.txt
```

The `provisioning.xml` file might look something like this:

```
<provisioning-archive ...>
    <tool-descriptions>
        <description>Hello World bundle</description>
        <display-name>Hello World bundle</display-name>
         <icon mime-type="image/gif">HelloWorld_icon.gif </icon>
    </tool-descriptions>

    <client-bundle>
        <content-id>http://www.vipera.com/helloworld</content-id>
        <version>1.1.0</version>
        <bundle-type>APPLICATION</bundle-type>
        <descriptor-file>HelloWorld.jad</descriptor-file>

        <tool-descriptions>
            <description>
                A MIDlet to demonstrate the Vipera provisioning server
            </description>
            <display-name>Hello World MIDlet</display-name>
            <icon mime-type="image/gif">HelloWorld_icon.gif </icon>
        </tool-descriptions>

        <user-descriptions>
            <display-name>Hello World MIDlet</display-name>
            <description>
                A MIDlet to demonstrate the Vipera provisioning server
            </description>
            <icon mime-type="image/gif">HelloWorld_icon.gif </icon>
        </user-descriptions>

        <vendor-info>
            <vendor-name>Vipera Inc.</vendor-name>
        </vendor-info>

        <copyright>/COPYRIGHT.txt</copyright>

        <device-requirement>
            <requirement-name>
                SoftwarePlatform.JavaPlatform
            </requirement-name>
            <requirement-value>MIDP/1.0</requirement-value>
        </device-requirement>
    </client-bundle>
</provisioning-archive>
```

If you wish to include variants of HelloWorld optimized for particular devices, you will create appropriate JAD/JAR files and add accordingly further `<client-bundle>` declarations to `provisioning.xml`.

Now that you have stocked the provisioning server with content, you probably want clients to begin finding out what's available on your server. That process is called *discovery*.

2.2.2 Discovery

The discovery process presents a list of available applications based on a query from a client. For that the client opens the URL `http://ota.vipera.com/` from an HTML or WML browser. The Vipera provisioning server automatically detects the browser type (HTML or WAP) and produces the appropriate markup pages. The discovery process results in the production of a URI the client device may request to initiate delivery of a particular bundle.

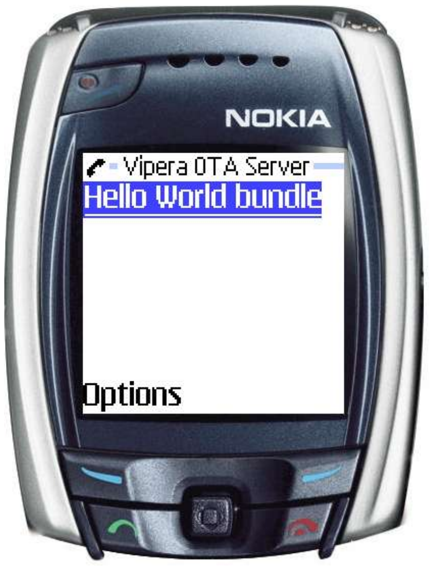


Illustration 2: Catalog browsing through WAP / WML

2.2.3 Delivery

The delivery process deals with picking from a provisioning archive the client bundle matching the device capabilities. The `<device-requirement>` block in the `provisioning.xml` defines restrictions regarding the device types a client bundle can be delivered to.

The `devices.xml` file of the provisioning portal acts as a database of device capabilities. Device capabilities are included in the `<device>` XML blocks. Request-to-device mapping is performed via the `<device-mapping>` XML blocks:

```
<device>
    <!-- Motorola i95cl -->
    <identifier>Motorola/i95cl</identifier>
    <adapter-name>midp</adapter-name>

    <capability>
        <capability-name>HardwarePlatform.ScreenSize</capability-name>
        <capability-value>120x160</capability-value>
    </capability>

    <capability>
        <capability-name>HardwarePlatform.BitsPerPixel</capability-name>
        <capability-value>8</capability-value>
    </capability>

    <capability>
        <capability-name>SoftwarePlatform.JavaPlatform</capability-name>
        <capability-value>MIDP/1.0</capability-value>
    </capability>

    <capability>
        <capability-name>SoftwarePlatform.JavaProtocol</capability-name>
        <capability-value>
            comm, socket, https, ssl, datagram, file
        </capability-value>
     </capability>
</device>

...

<device-mapping>
    <identifier>Motorola/i95cl</identifier>

    <request-mapping>
        <header-name>user-agent</header-name>
        <header-value>Motorola/i95cl</header-value>
    </request-mapping>
</device-mapping>

...
```

No matter whether a client bundle is downloaded from an HTML or WAP/WML browser, or via WAP-Push, the client device will always request the JAD file through HTTP. The provisioning server looks at the `user-agent` HTTP header and iterates over all the `<device-mapping>` blocks. The request-mapping declarations define which `user-agent` values map to which device identifiers. Once the device has been matched, the `<device>` block with the same `<identifier>` value as in the matched `<device-mapping>` block is loaded.

In a second step, the `provisioning.xml` `<device-requirement>` blocks are matched against the device capabilities we just loaded. The resulting client bundle is then delivered to the mobile user.

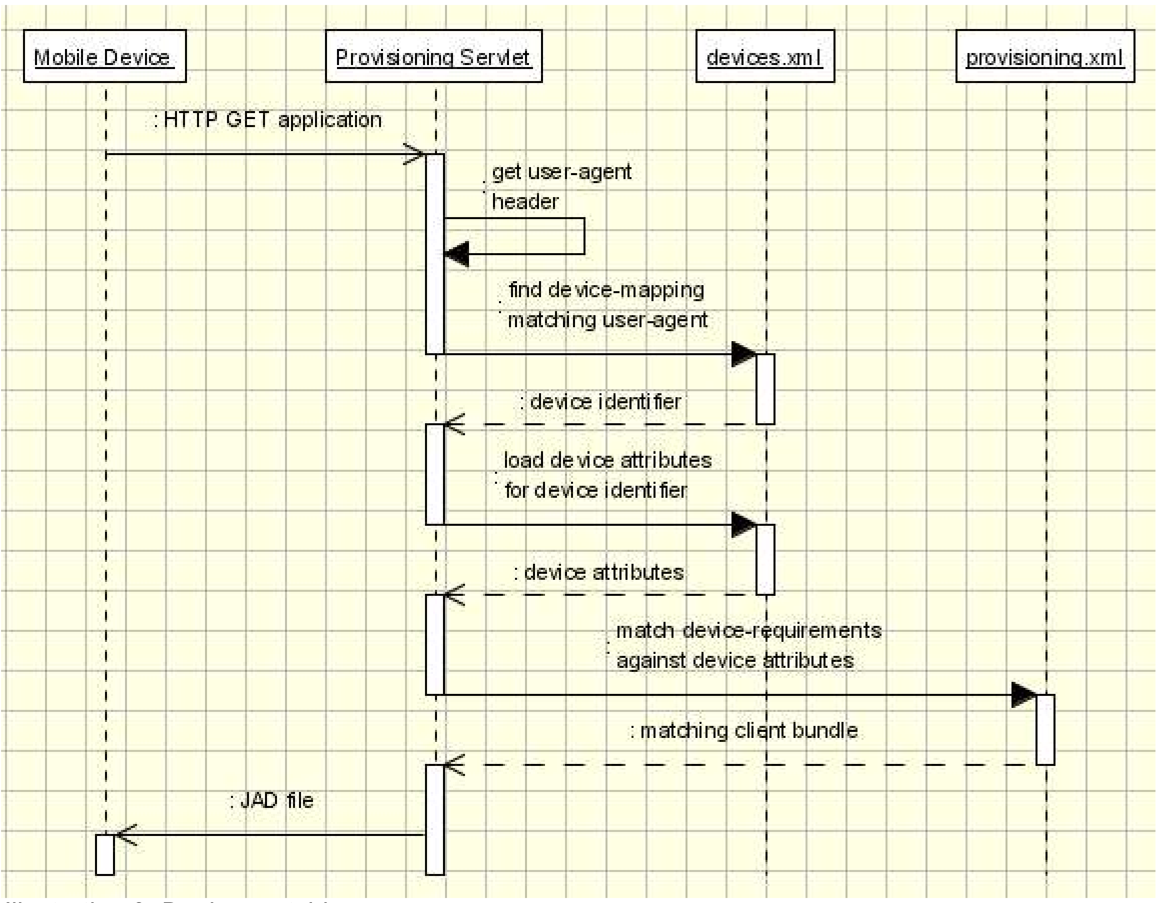


Illustration 3: Device matching process.

The matching configuration file, `matchers.xml`, also plays a key role in determining which bundles are suitable for a client. This file defines the algorithms that the provisioning server uses to compare bundle requirements against device capabilities. It can easily be extended to support new capability attributes. For example, this block of XML from a `matchers.xml` file defines a matcher for checking screen size:

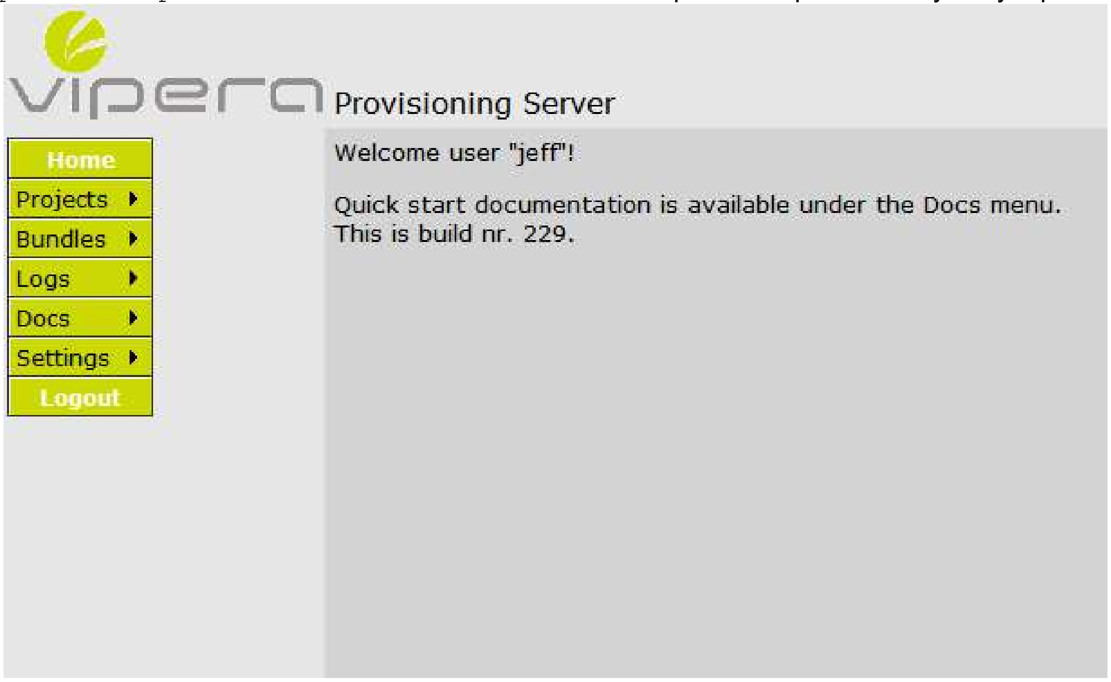
```
<matcher>
    <attribute-name>HardwarePlatform.ScreenSize</attribute-name>
    <matcher-class>
        javax.provisioning.matcher.DimensionMatcher
    </matcher-class>

    <init-param>
        <param-name>allMustMatch</param-name>
        <param-value>false</param-value>
    </init-param>
</matcher>

...
```

3 Vipera Provisioning Server – Usage

In the last chapter we provided the conceptual background of the JSR 124 specification on which the Vipera provisioning server is built. This section offers some more hands-on usage instructions for the various user classes.

3.1 Content Providers and Developers

The use cases for J2ME content providers and developers were defined in section 2.1. Content providers create PAR bundles and upload them to the provisioning server. First of all, you must log into `http://ora.vipera.com/admin` with the user name and password provided to you by Vipera Inc.



Welcome user "jeff"!

Quick start documentation is available under the Docs menu.
This is build nr. 229.

Illustration 4: Portal page after logging in with user name "jeff"

The portal repository is structured into user spaces (e.g., `jeff` has got his personal space in the repository). A user space can then be subdivided into multiple "projects" by the user himself. PAR bundles are always uploaded into projects.

3.1.1 Menu Structure

Menu	***Sub menu***	***Explanation***
Projects	Create	Create a new project in the user's repository
	View	View projects
	Delete	Delete a project and all bundles in it
Bundles	Deploy	Upload a bundle into an existing project
	Delete	Delete a bundle from a project
	WAP Push	Deliver a bundle through WAP Push
Logs	OTA Log	Server log
	Header Log	View HTTP headers of downloads issued so far
Docs	Quick Start	Quick start documentation
	Developer	Developer documentation

Menu	***Sub menu***	***Explanation***
Settings	Password	Change password

Portal administrators will see a further “Admin” menu with functions for creating and editing users, fo changing user roles, for administering the `device.xml` file, etc.

3.1.2 Basic Example

With this example you can practice the deployment and delivery of an application bundle containing a “Hello World” MIDlet.

1. Docs → Quick Start: download the Hello World bundle to your PC
2. Projects → Create: Create a project called “test”
3. Bundles → Deploy: Upload the Hello World .par file to the “test” project
4. Bundles → WAP Push: select the “test” project and the Hello World bundle. Enter the phone number of your Java enabled phone. Push the bundle to your phone.
5. Logs Header → Log: After installation, view the HTTP headers transmitted by your mobile device.

3.1.3 More Advanced Example

This is a variation of the last example to show on-the-fly customization of the JAD. First deinstall the Hello World MIDlet, then go through the steps outlined in 3.1.2 once again but with the following change:

4. Bundles → WAP Push: Set “JAD Property Name” to `message` and enter some text into the “JAD Property Value” field. That text will be displayed by the Hello World MIDlet. This works simply because the MIDlet displays the value of the `message` JAD property in an LCDUI form.

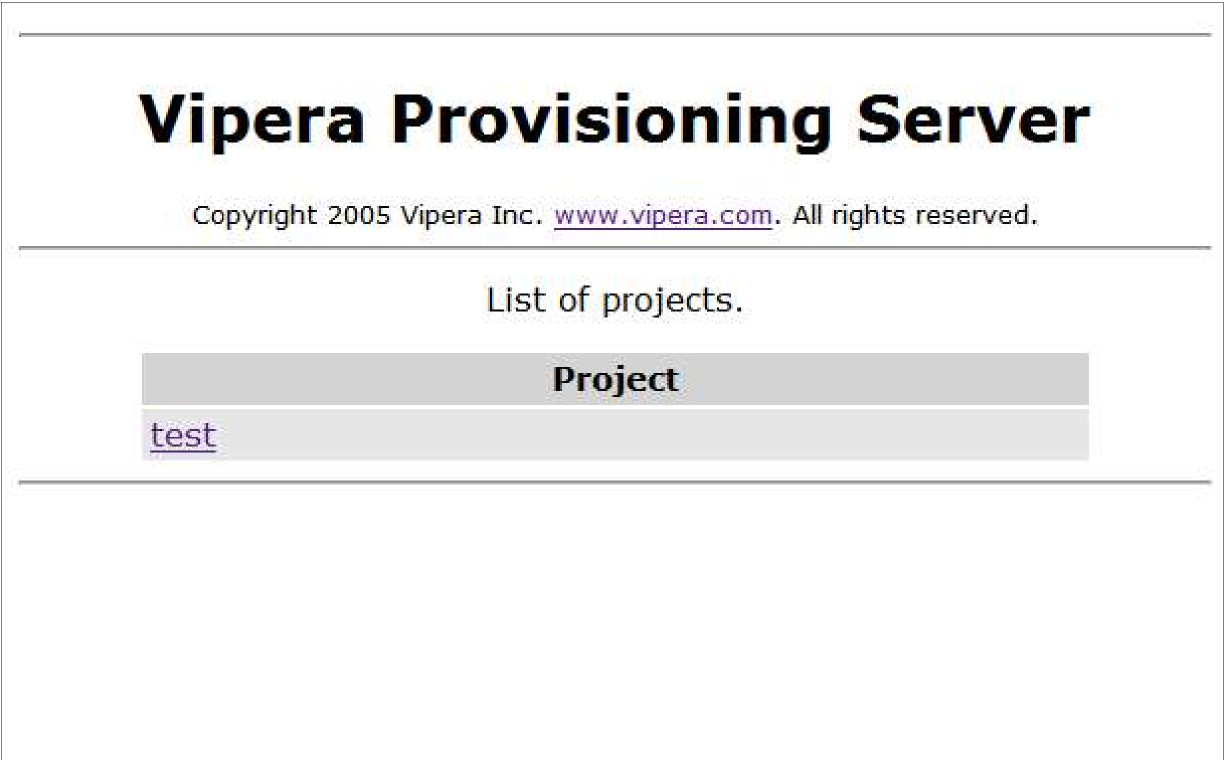
3.2 Mobile End Users

End users access the provisioning server in two different ways depending on who initiates the installation of a bundle

- Explicitly, by browsing the portal with an HTML or WML browser
- or implicitly, when a third party initiates the delivery of a bundle via WAP Push.

3.2.1 Browsing the Bundle Catalog

The URL of the catalog is `http://ota.vipera.com/`. In an HTML browser, the catalog pages are rendered as follows.



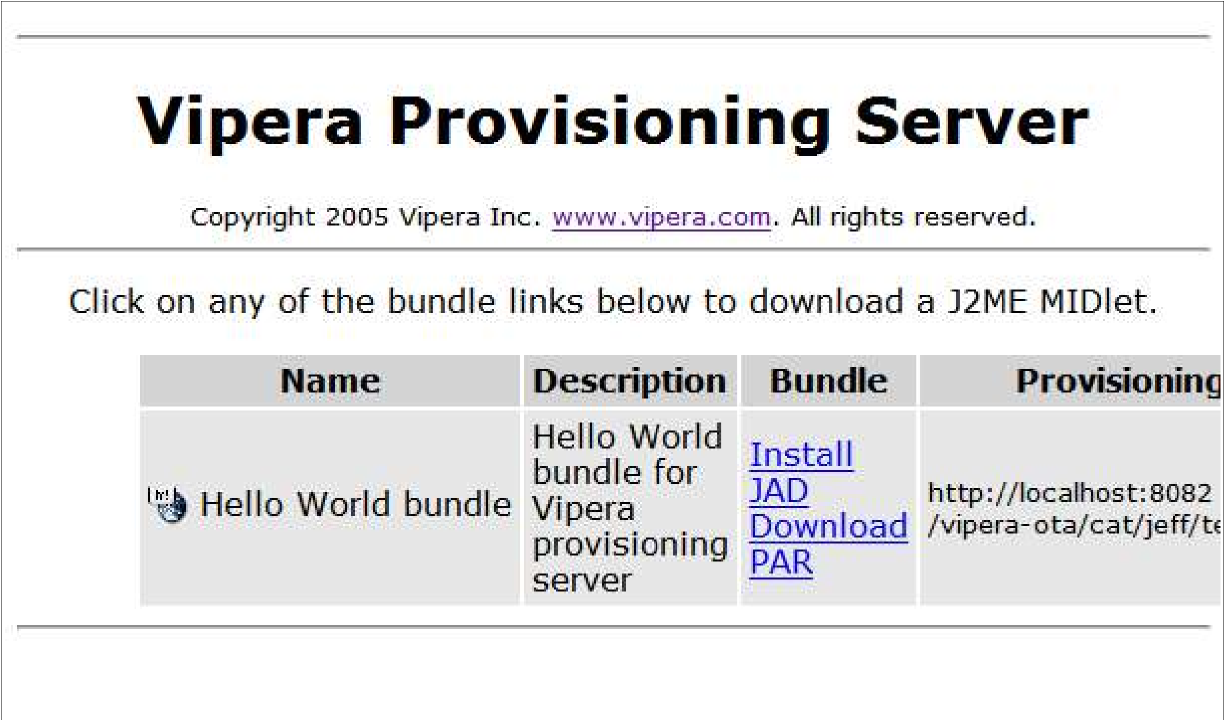
Vipera Provisioning Server

Copyright 2005 Vipera Inc. www.vipera.com. All rights reserved.

List of projects.

Project
test

Illustration 6: A catalog web page after opening the link "jeff".



Vipera Provisioning Server

Copyright 2005 Vipera Inc. www.vipera.com. All rights reserved.

Click on any of the bundle links below to download a J2ME MIDlet.

Name	Description	Bundle	Provisioning
Hello World bundle	Hello World bundle for Vipera provisioning server	Install JAD Download PAR	http://localhost:8082 /vipera-ota/cat/jeff/te

Illustration 7: A catalog web page after opening the link "test".

3.2.2 Delivery through WAP Push

WAP Push offers greater convenience to end users in that bundles are "pushed" to the device via a special type of SMS, without need for the user to browse any catalogs. The provisioning process works as follows.

1. A content provider triggers a push delivery of a bundle to a particular mobile phone number. This is accomplished either manually, through the aforementioned "Bundles → Deploy" menu item, or programmatically by issuing a HTTP POST request to the Vipera provisioning portal. HTTP POST is described in the next section.
2. The end user receives a service notification message. Typically through an audible alert ("beep") or vibration, analogous to the receipt of a conventional SMS.
3. The user is prompted whether he wants to download and install the application.
4. The user accepts the delivery and the MIDlet is downloaded and installed automatically.

3.3 Integration into 3rd Party Portals

The manually triggered WAP Push of section 3.1.2 is appropriate for testing. For automating the delivery of client bundles, the Vipera provisioning portal provides an HTTP interface. A third party software component can issue a properly formatted and authenticated HTTP POST request to the portal, to trigger the delivery of an application through WAP PUSH.

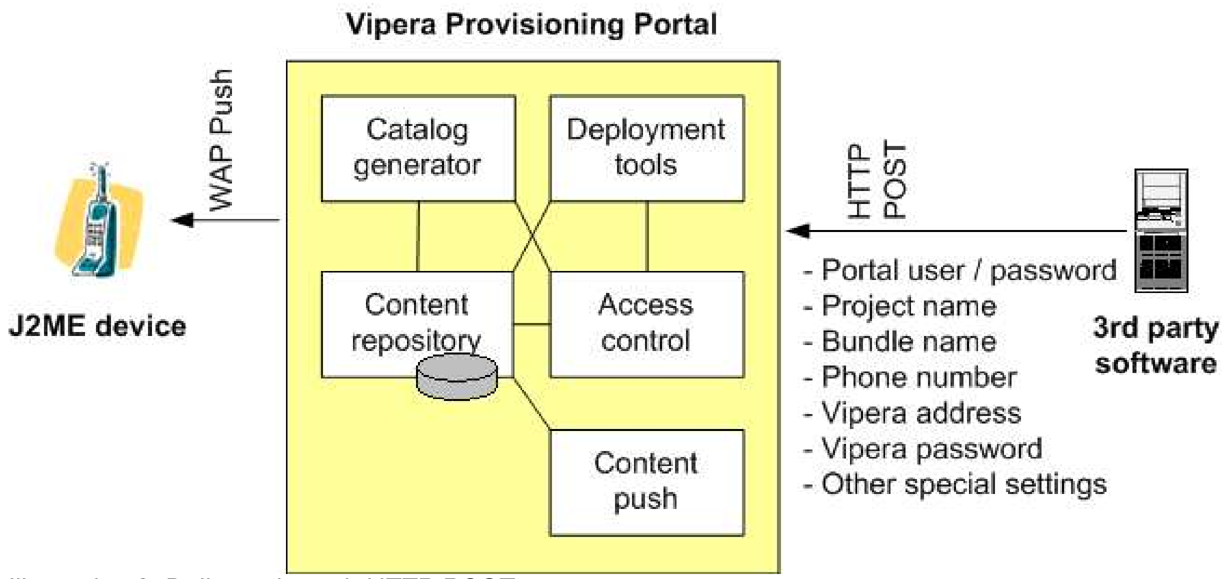


Illustration 8: Delivery through HTTP POST.

3.3.1 HTTP Request

The content type of the HTTP POST request must be `application/x-www-form-urlencoded`
The URL is `https://ota.vipera.com/push`
A proper `Authorization` HTTP header must be inserted for "basic" authentication using your portal user name and password.

The following mandatory and optional HTTP request parameters are supported:

Name	***Explanation***
`project`	Name of the provisioning project as defined on the portal
`bundle`	Name of the bundle as defined on the portal
`phone`	GSM phone number of the target device without any spaces. In international notation, but without any leading + signs or zeros. Example: 41792126677

Table 2: Mandatory HTTP request parameters.

Name	***Explanation***
`msg`	Informative message to show to the user as part of the WAP Push request. Example: “This is the application you requested from the ACME portal”.
`jadname`*`N`*	Used to define key/value pairs which will be inserted into the JAD by the portal, just before the JAD is sent to the client. `N` is an index number starting at 0. Example: `jadname0=MyProperty`, `jadvalue0=MyValue` This will insert the line `MyProperty: MyValue` into the JAD.
`jadvalueN`	
`viperaid`	Vipera address and password. Will automatically be inserted into the `vipera.conf` file.
`viperapasswd`	

Table 3: Optional HTTP request parameters.

3.3.2 HTTP Response

Upon successful execution of the push request, the HTTP response will bear a status code of `200`. Also, an HTML document with content type `text/plain` containing the string `ok` is returned.

On errors a status code different from 200 is returned. The response `text/plain` document *may* contain a string detailing the cause of the error.

4 Terminology

Term	***Explanation***
Provisioning Archive (PAR)	An archive containing one or more client bundles as well as a deployment descriptor XML file
Client Bundle	An application that can be delivered to a mobile device. A PAR may consist of multiple client bundles. We will sometimes use the term bundle as synonym for provisioning archive.
ISV	Independent Software Vendor
J2ME	Java-2 Micro Edition
MIDP	Mobile Information Device Profile
OTA	Over-the-air