NetVenue

Application developer handbook

Issue: 01.01

Status: Standard

Date: September 1999

Standard

NORTEL NETWORKS™

NetVenue

Service developer handbook

Issue: 01.01
Status: Standard
Date: September 1999

Publication history

September 1999

This is the standard release of the Application developer handbook.

Table of Contents

1 Introduction

This document provides instructions on service development for NetVenue terminals.

This manual is designed for people who have a solid background in HTML, Java, or ActiveX. Service developers who are setting up transactional services will find a working knowledge of Perl scripting or other server technologies an advantage.

The service portion of the multimedia terminal is not a stand-alone part of the development process. The Web service developer must deal closely with those who create the menu system and administer the terminals using the Admin Tools Suite.

As well, the Web service developer should design the content in such a way that it takes advantage of the features of the multimedia terminal and runs smoothly with it.

If you are developing content for NetVenue terminals, then you should have access to the following:

- Web server capable of supporting a scripting language
- An SMPT server
- FTP server
- Graphics and Web development applications
- Administration tools
- Access to the network with the NetVenue Manager
- NetOp software
- Access to a NetVenue terminal for test purposes

2 NetVenue terminal hardware

The NetVenue terminal is a powerful device that allows the public to access various services, including E-mail, Web browsing, custom services, and pay telephony.

As a service developer, you do not need to know all of the hardware and software details. This chapter provides required information about the terminal hardware.

Developing services for NetVenue terminals provides a powerful advantage: you know what sort of device users will be running to access your sites. You know, for example, that all of the monitors displaying your service have an 800 by 600 pixel display area. This knowledge can be useful when creating Web-based services and applications.

Extended services

It is important to know whether the service you are developing will be viewed on standard NetVenue terminals, NetVenue terminals with additional peripherals, or both.

Table 2-1 shows what peripheral devices are needed to support non-standard services.

Table 2-1: Supporting services

Service	**Peripheral required**
Print Postscript files	laser printer

Service	**Peripheral required**
Print Web pages	laser printer
Print E-mail messages	laser printer
Print tickets	ticket printer

Touchscreen

The terminal has a touchscreen surface covering the display. This allows users to click on items by touching the screen.

Try to make everything in a service usable with the touchscreen. Some guidelines for doing this are:

- Use large buttons. Larger buttons are easier to activate, and reduce problems caused by the height and position of the user.
- Use buttons rather than hypertext links. Buttons are easier to touch and are more intuitive.
- Avoid pull down menus in HTML forms.
- Avoid any interfaces which require the user to perform a "dragging" action.

Liquid Crystal Display

The Liquid Crystal Display (LCD) is a high resolution 15” screenset with a resolution of 600 x 800 and True color.

Because you know the exact resolution of the screen your service will be appearing on, you can make more effective use of your space. Do not make content that would fit on a 480 by 640 display, as it will look small and out of place on a larger screen. Similarly, content designed for a 1024 by 768 display will generally not fit on the display.

In addition, you may want to optimize your graphics for True Color. Graphics designed for 256 colors will look pixelated.

Telephony component

The NetVenue terminal has a telephony component which can be used for hotlines or pay telephony. A hotline can be a call center, a help line, a touchtone information line, or any other phone number.

With the telephony component, users can access a hotline or use enhanced telephony features, which allow them to make calls and pay with several different card payment options.

You can create a link within your service which automatically dials a hotline.

Keyboard

The keyboard does not have a numerical keypad or function keys. It also does not have Control, Alt, or Command keys. Because of this, users cannot perform some standard operations, such as cutting and pasting. While this should greatly not affect Web services, be aware that those keys are not present, so that you do not give users misleading instructions. For example, do not tell them that they can print by pressing Control + P, or tell them to copy and paste text into the body of an E-mail message.

Printers

There are three types of printers available in terminals.

Receipt printer

The receipt printer in the base of the terminal prints from a role of 4.75-inch wide thermal paper.

Ticket printer

The ticket printer is adjustable to support different widths of paper stock. It is a peripheral component that can be connected to the basic terminal.

Laser printer

The laser printer prints on 8.5 x 11” sheets. Using the kiosk tags, you can embed a command to print a Web-page within the HTML code, or you can use one of the service API commands to print a Postscript file.

See Chapter 6 for more information.

Card reader

The card reader can read stored-value chip or magnetic stripe cards. The terminal currently supports two types of payments through the card reader: pay-per-use and one-time purchases.

A pay-per-use service is a service that is provided based on how long the user uses the service. For example, Internet access can be configured through the Admin Tools Suite as a pay-per-use service.

In one-time purchases, the user pays once for a product or service. Most e-commerce falls into this catagory, where a user pays for a ticket to a movie, for example.

To learn more about card transactions, see Chapter 7.

3 Web services

All NetVenue terminals have the same software package. This allows developers to make assumptions about what systems users are running. However, to keep the system running efficiently and securely, the terminal restricts downloading of files from the Web onto the terminal's hard-drive.

By being familiar with the software, you can create services that run smoothly and efficiently on the terminals.

Embedded Web browser

The multimedia terminals use an embedded Web browser component which uses industry standard technology. Check to see what browser version is on the terminals you are developing for. Anything that runs on the standard version of that browser will run on your terminal. However, embedded browser allows for more flexibility than a traditional browser; the Menu developer can create toolbars using the Admin Tools Suite.

Multimedia files

Movies and multimedia content which require a lot of time to download or that do not change often can be stored locally on the terminal's hard drive. You can make a reference to the file on the hard drive, so that the terminal doesn’t need to download the multimedia file every time the file is needed.

If you plan to host files locally on the terminal, provide the multimedia files to those responsible for downloading customer content to the terminal.

Test your service’s response times under the same conditions as the live terminals to ensure the user interface is responsive.

Codecs supported

Codec stands for compression/decompression. A codec is similar to a file format; a file needs to be opened with the same codec it was created with. If a file format was created with Cinepak, it can only be read by the Cinepak codec. Most multimedia programs, however, provide a broad range of codecs to choose from. It is recommended that you use Media Cleaner Pro 2.0 software to create video files for use on multimedia terminals.

For Web services, the following Codecs are supported:

Table 3-1: Codecs supported

Codec	Size	Data rate
ClearVideo with quickstart	160 x 120 256 colours	Maximum 75Kb/sec; suggested 45Kb/sec
Cinepak with quickstart	160 x 120 256 colours	Maximum 55Kb/sec; suggested 45Kb/sec

Front-end menu system

One major concern when designing content for terminals is that your Web content interfaces with the front-end menus, created by the front-end menu creator. Users do not reach your site by typing your address into your browser, nor do they reach your site through a search engine. Instead, users click on a link to your service from a series of front-end menus.

It is advisable to work with the menu developer to ensure that the description of the service on the front-end menu system accurately describes the service.

Size constraints

When you are programming HTML for the general public, you can never tell what sort of system the user will be running. For example, it is difficult to ensure that a page that works well in 800 by 600 resolution will look good in 640 by 480, or in 1,024 by 768.

With the NetVenue terminal system, you can be certain that the software is being run in an 800 by 600 pixel environment. Because the user cannot resize the windows or choose which toolbars are displayed, you can calculate the size of the window you have to work with.

The bottom of the screen is reserved for the status bar, which is 30 pixels high. However, this status bar can be disabled on a per service basis. Find out from the front-end content developer what the size and location of the Web toolbar is, in pixels. The Web toolbar can be located on any side of the browser window.

By adding the height of the toolbar and status bar, and subtracting that figure from 600, you can find the area you have to work with.

For example:

Suppose that in your background you have the image of a car. You want the car to be positioned immediately above the toolbar, without cutting off the tires, but without any space between the car and the toolbar.

You know that the toolbar is 55 pixels and the status bar is 30 pixels. Therefore, the bottom 85 pixels are reserved.

In order to position the car at the bottom of the browser window, the bottom of the wheels of the car must be in the 515th row of pixels.

Customizing toolbars

Generally, the Web toolbars are created by the front-end menu developer. Because they are customizable, you can ask the front-end menu developer to add or remove buttons for you.

One of the Web toolbar button options is the creation of a custom button. This can link to a different site or provide other services. Talk to your front-end menu developer if you want any special buttons. Be sure that the graphics used are clear enough that the user knows when to push them, or else provide users with instructions on when to use them.

If you include the "goto" button, this will allow the user to go anywhere on the Internet. This can create problems if it is included on a free site, while a pay-per-use Internet service also exists on the terminal. Users can use the goto button to achieve free Internet access, defeating the purpose of the pay-per-use access.

Returning to the menus

It is not always obvious to the users that they can exit a service and return to the front-end menus by pressing the exit button. Some users may be perplexed that they cannot use the back button to return the front-end menus.

Make sure that you provide instructions on returning to the front-end menus. If the purpose of your site is to perform a specific task, such as purchasing tickets, you may want to have a "cancel" button on every page. Another useful button is a "done" button that users can click after performing the transaction. Both of these buttons should take the user back to the front-end menus.

4 Creating the interface

This chapter deals with creating the interface. A good interface can help create a sense of identity, aid in navigation, and decrease the imposition on the user. As with any design process, it is important to consider who the audience is. This chapter is important particularly for designers who have never developed a site for the general public.

Multimedia terminal users

Internet users are notorious for being impatient. NetVenue terminal users may be even more impatient than the average Internet user. Consider the following:

- Most of the terminals will be in public or semi-public locations. Users are going to feel hurried, especially if there is a line-up to use the terminals.
- The user may be paying for the service. If this is the case, users are going to become very frustrated if they spend time and money waiting for pages to download.

For these reasons, try to keep content loading time as short as possible. Use multimedia, Java applets, and large graphics only when necessary.

When possible, downsample JPGs to 8 or 16 colors, to decrease load times. Remember to test everything you develop under realistic conditions. The multimedia terminal is accessible to a larger range of people than personal computer. Users may be unfamiliar with such concepts as back and forward buttons. Follow these guidelines:

- Make sure that the error messages within your service provide the user with a description of what went wrong, what the user should do next, and how the user can contact support or obtain help.
- Do not force the user to use the back, forward, or refresh buttons, if possible. This may require you to create an interface in which the users are always aware of their location.
- Do not create any extra browser windows. This can easily confuse users who are not familiar with a Windows environment.

Consistency

This section explains some strategies to help increase consistency on your site.

- Restrict your palette. Using fewer colors often creates simpler, better images that load quicker.
- Use standard navigational devices throughout the site, such as menus at the top, bottom, or left side of the screen which always tell users where they are.
- If you are using images for buttons, make sure that they are easy to understand and consistent throughout the site.
- Use a site metaphor, such as a virtual museum, only if the metaphor makes sense for your entire site. Be very careful with site metaphors, as they are often far less intuitive then their real world counterparts, particularly when dealing with a general audience.
  For example some sites use the idea of a virtual museum, in which users see a room with images along the walls.
  While most users will understand that they are meant to turn and look at the exhibits, not all will deduce that clicking on the left portion of the screen will cause the image to turn to the left. Be sure that any complicated

navigation like this has sufficient directions.

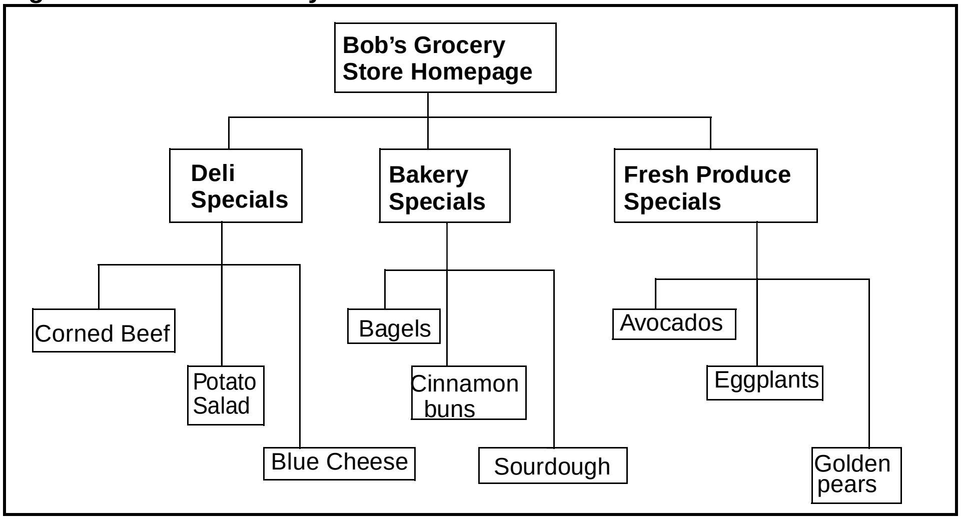
Navigability

It is crucial that users find their way around the Web service. This means allowing users to find information or a service, and then leave again, all quickly and with little effort. As mentioned above, site metaphors are not always the best way to achieve navigability.

Here are some tips

- Create a hierarchy. Assume that the user will enter the top level of the hierarchy, and move downward to find information. Put more general information higher up in the hierarchy, and more specific information lower in the hierarchy. Do not make too many levels in your hierarchy, however.
  Figure 4-1 shows a typical Web-based service hierarchy; from the home page, there are links to the major parts of the site. In this case, they are the deli, bakery, and fresh produce.
  The front page will not contain much information, just a general overview of the site. The specials pages can contain more information, such as an image of each product and its price. The lowest levels can go into detail about the various specials, giving their nutritional value and perhaps even some recipes. With this sort of organization, users will see little information other than what they are interested in

Figure 4-1: Site hierarchy



- Assuming that users know what they are looking for when they enter the service, they should not have to click more than four links to find the information they are looking for. In other words, the site hierarchy shouldn't be deeper than four levels.
- Restrict the number of links on a page. A page with too many links is overwhelming and difficult to navigate. If you must put a lot of links on a page, be sure that they are subdivided into categories.
- Restrict the number of graphics on a page. Use a few strong graphics, rather than a lot of weak ones.
- Make sure all links are valid and tested.

Button sizes

All buttons should be larger in size than those for a standard Web site. The touchscreen is less accurate than other pointing devices, so small buttons are difficult to use. A half inch in width or height is too small for buttons. Consider the following:

- Depending on the height of the user, people will touch a slightly different area of the screen to activate a button. This is the result of parallax because the touch screen is in front of the actual LCD (See Figure 4-2 below)

- The calibration of the touchscreen as a mouse control device is also subjective. The service person who calibrates the touch screen may not see it exactly the same way as another technician or the way the average person views the screen.
- Some users’ fingers are much larger than average. A larger finger may press two small, adjacent buttons if they are placed too closely together.

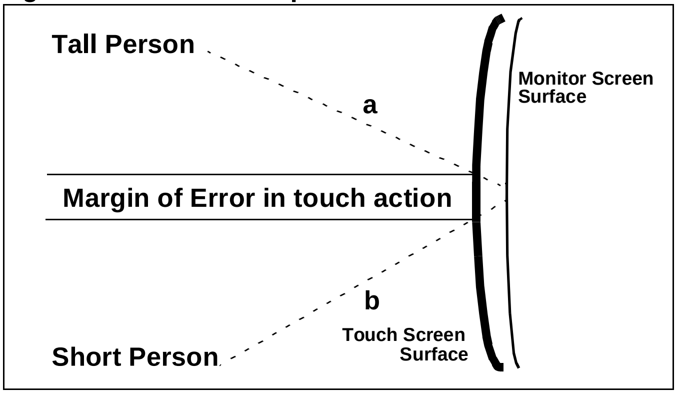
Parallax is more likely to occur vertically on the screen due to variances in the height of the user. Most people will usually take a central position on the horizontal plane. The conclusion is that the height of your buttons is more important than the width from a functional perspective.

You will have to consider a larger margin of error for people using your application on the multimedia terminal.

For example:

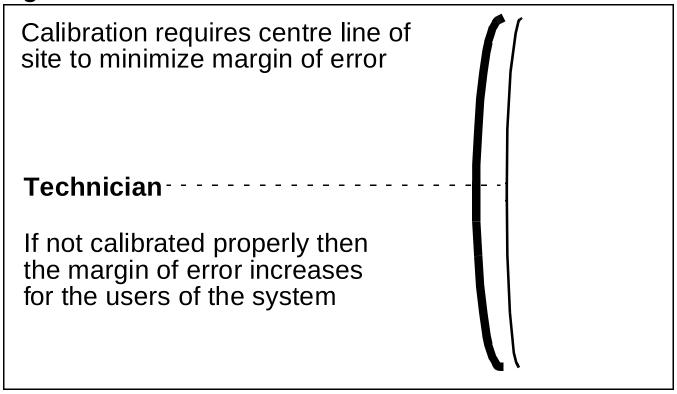
A tall person's line of site would cause them to touch the screen above the ideal contact point, i.e. where line (a) intersects the touch screen surface. The opposite applies for a short person i.e. where line (b) intersects the touch screen surface.

Figure 4-2: Line of site parallax for touch screens



The minimal margin of error assumes that the touch-screen was properly calibrated by a technician who had a line of site perpendicular to the touch screen at the mid-point of the screen, as shown in Figure 4-3.

Figure 4-3: Monitor calibration


Calibration requires centre line of site to minimize margin of error
Technician
If not calibrated properly then the margin of error increases for the users of the system

5 Developing Web services

This section discusses HTML support as it pertains to Web services. It is assumed that as a content developer you already know how to create HTML content, and have some understanding of the tags.

Table 5-1 lists technologies supported by the NetVenue terminal:

Table 5-1: supported technologies

Technology	PC Build	Macintosh Build
HTML	Version 4.0	Version 3.0
DHTML	Yes	No
Javascript	Version 1.2	Version 1.1
Java Applets	Version 1.1	Version 1.0
Cascading Style Sheets	Yes	No

Tags to avoid

Table 5-2 lists some tags that are not supported, the reasons why they are not supported, and suggests work-arounds.

Table 5-2: Tags which should not be used

Tag not supported or strongly discouraged	Reason	Workaround
`news:` tags	Newsgroups are not supported on the multimedia terminals	Provide a Web-based news service.
`mailto:` tags	`Mailto:` tags are not currently supported.	If you need user input, create an HTML form, which does not require a mailto: tag. See the section below for more information.
`FTP:` tags	FTP tags allow users to download content to the terminal. This can jeopardize the security of the terminal.	Make all content viewable through the Web-browser.

Creating forms

If you are creating HTML forms to be used for financial transactions, you may want to use CGI (common gateway interface) scripts or other server technology to send the form information to the database. Consult your database or system administrator to find out what sort of server application development is appropriate for your database.

Working with frames

There are a few things you should be aware of when using frames:

You should disable the ability to resize frames, as this can confuse users, and requires "dragging", which is difficult on the terminals. To disable frame resizing, use the NORESIZE tag inside the FRAME tag.

html sample

```
<FRAME NORESIZE>
```

```
frame information
</FRAME>
```

You should also disable scrolling within frames, or create another scrolling device. Using the scroll bars on the side is difficult with the touchscreen. To disable scrolling, use the SCROLLING tag in the FRAME tag.

html sample

```
<FRAME SCROLLING="NO">
frame information
</FRAME>
```

If you must have scrolling text on your screen, consider using an alternative scrolling mechanism, using JavaScript or another technology.

6 Web service API commands

Web service API (application program interface) commands are HTML tags that are recognized only by NetVenue terminals.

API commands take advantage of services provided by multimedia terminals, such as printing, card reading, and phone dialing. They allow you to request services of the terminal directly from your HTML code, so that you can ask a multimedia terminal to print pages, dial a hotline, or read a card.

Support for API commands

While every API command can be read by every terminal, certain devices must be present, or it ignores the commands. Table 6-1 lists the API commands and the corresponding devices needed.

Table 6-1: Required devices

Command	Device
quit	none
phone	telephony component
print	receipt printer
printpage	laser printer
printfile	laser printer

Command	Device
card	card reader

Implementing API commands in anchor tags

The next section deals with different implementations of API commands. This example uses the kiosk:quit tag, which causes the Web service to exit and return the user to the menus.

If users complete a specific task, for example, purchasing tickets, you may want to give them the option of exiting. This is very simply done with the anchor tag. The syntax for this is as follows:

html sample

```
<A HREF="kiosk:quit”>quit</A>
```

In this example, a text hyperlink is shown for simplicity, but an appropriately sized graphic should be used.

The kiosk tag appears in double quotes following <A HREF=. As with any link, the text or image used for the link is not important, but it must be followed by a </A> tag.

When the user clicks quit, the Web service closes. However, you may want to display a message when the user wants to quit. The next example shows you how to do this.

Implementing kiosk commands as timed effects

Some kiosk commands can be implemented in different ways. The quit tag can be useful as a timed effect. This example uses the kiosk:quit tag to close the browser after displaying a message.

Instead of using the HTML script in the previous example, create a link to another page.

html sample

```
<A HREF="goodbye.html">quit</A>
```

When this line is used, clicking on **`quit`** causes the browser to load a page called goodbye.html. That page could be created as follows:

html sample

```
<HEAD>
<TITLE>goodbye.html</TITLE>
<META HTTP-EQUIV="REFRESH" CONTENT="3; kiosk:quit">
</HEAD>
<BODY>
<P>Have a nice day!</P>
</BODY>
```

In this example, this message appears on the screen:

`Have a nice day!`

The section in the heading creates a delay, so that the kiosk:quit command takes effect after three seconds.

Many kiosk commands become problematic if implemented this way. The refresh command in the above example is actually telling the page to reload the quit page every three seconds. But because the quit command exits this page, the command executes only once. However, if a printfile command was used in the place of the quit command, then a file would be sent to the printer every three seconds. Be very careful when using this type of implementation.

Implementing kiosk commands using JavaScript

Another possible way of implementing kiosk commands is to use JavaScript. One advantage of using JavaScript is that it allows you to pass variables in your kiosk command.

In JavaScript, the kiosk command works like a normal HREF, with one exception. The API command uses a colon, which JavaScript recognizes as an illegal character. A few lines of code can get around this.

1. Pass the entire kiosk command into a variable.
2. Run the **`escape()`** function on the variable, passing the result to another variable.
3. Use **`window.location.href`** to open activate the kiosk command.

In the following example, the JavaScript function **`print()`** takes the user's first and last name, as entered in a form, and writes a personalized greeting.

The third and fourth line define **`firstname`** and **`lastname`** as variables.

In the fifth line, the kiosk command is assembled, by combining the kiosk command, text and variables.

In the sixth line, the **`escape()`** function is used to convert the colon, and exclamation mark, and spaces to hexadecimal values.

In the seventh line, this string is sent to the browser.

html sample

```
<script language="JavaScript"><!--
function print(){
  firstname=document.form1.firstname;
  lastname=document.form1.lastname;
  mystring='"kiosk:print!hello there " + firstname + lastname';
  escaped=escape(mystring);
  window.location.href=escaped
}
//--> </script>
```

API commands

This section explains each API command in detail.

kiosk:quit

The kiosk:quit command exits the Web service. Because a quit button often appears on the Web toolbar, this tag is rarely necessary.

html sample

```
kiosk:quit
```

Example

The following example exits the Web service when the user presses `quit.`

html sample

```
<A HREF="kiosk:quit">quit</A>
```

kiosk:phone

The kiosk:phone tag instructs the terminal to dial a phone number. The user can then pick up the phone to speak to hotline personnel or listen to a recorded message if one is provided by the called number.

The kiosk:phone command provides variables which allow you to specify a prefix or an extension.

html sample

```
kiosk:phone!service_symbol!phone_no!ext
```

The following table explains the different fields in the kiosk:phone tag.

Table 6-2: kiosk:phone fields

field	**definition**
`service_symbol`	The service symbol is assigned by the system administrator to identify the service. It is 3 to 8 characters, all in upper case.
`!`	This separates items within the kiosk tag. Note that this is different from some scripting languages, in which it means "not."
`phone_no`	This is the phone number to dial. You can type NULL to use the default phone number defined in the database for the current service. You do not need to put in spaces or hyphens.

field	definition
`ext`	This is the extension to dial. If a value is entered, the terminal will dial the phone number, wait for approximately six seconds, then dial this number. You can type NULL if there is no extension to dial. If there is no extension, you can omit this field entirely.

Examples

In these examples, the service symbol is FLOWERS. All examples use the anchor tag.

This first example dials the default phone number configured for the FLOWERS service in the configured database.

html sample

```
<A HREF="kiosk:phone!FLOWERS!NULL">Click here to speak to a
customer service representative.</A>
```

In the following example, the terminal dials the default phone number, then dials extension 345.

html sample

```
<A HREF="kiosk:phone!FLOWERS!NULL!345">Click here to speak to a
customer service representative.</A>
```

In the following example, the terminal dials phone number 905-555-1212.

html sample

```
<A HREF="kiosk:phone!FLOWERS!9055551212!NULL">Click here to
```

```
speak to a customer service representative.</A>
```

In the following example, the terminal dials phone number 905-555-1212, and then dials extension 345.

html sample

```
<A HREF="kiosk:phone!FLOWERS!9055551212!345">Click here to
speak to a customer service representative.</A>
```

kiosk:print

The kiosk:print command allows you to print short strings to the terminal's receipt printer. These strings are sent in raw form to the receipt printer. See the notes section below for an explanation of raw form.

html sample

```
kiosk:print!text
```

Table 6-3: kiosk:print fields

field	definition
`text`	This is the text that is sent to the receipt printer. Standard characters are printed as normal. To print special characters or accented characters, use the standard Latin 1 character code. You can find this in appendix B. You can format the text using escape codes. These codes can be found in the Swecoin printer manual.

Example

The following example displays the message "20% off your next purchase" when the word "Coupon" is pressed.

html sample

```
<A HREF="kiosk:print!20% off your next purchase%0A">Coupon</A>
```

Special characters

To insert a special character, use "%XX" where XX is the hexadecimal number. For example, to print é, use %E9.

One useful application of this is to print characters which otherwise cause syntax problems. In the following example, the kiosk command will not work properly, since the browser will interpret the double quotation mark before Walnut as the end of the HREF.

incorrect html sample

```
<A HREF="kiosk:print!One free ticket to "Walnut World". Present
this ticket to the doorman.%0A">Ticket</A>
```

To avoid this problem, use %22 to replace the double quotaion marks. The following example will print out correctly.

html sample

```
<A  HREF="kiosk:print!One  free  ticket  to  %22Walnut  World%22.
Present this ticket to the doorman.%0A">Ticket</A>
```

The final line in a print string must end with a return. The special character for this is %0A.

Printing variables

Often, you may want to print variables in your text string. For example, after asking the user to input their personal information, you may want to print out a personalized voucher which greets them by name. You may want to include the date on a coupon so that users cannot print out a large number of coupons to use over an extended period of time.

One way of achieving this is to use the JavaScript syntax on page 6-4, which demonstrates passing variables into the kiosk command.

kiosk:printfile

The kiosk:printfile command prints one or more PostScript files stored on the local hard drive to the laser printer. This command is ignored if a laser printer is not configured.

html sample

```
kiosk:printfile!filetype=<filetype>&filename=<filename>...&
filetype=<filetype>&filename=<filename>
```

The following table explains fields in the kiosk:printfile tag.

Table 6-4: kiosk:printfile

field	definition
`filetype`	The kind of the file to be printed. Currently, PostScript is the only format supported. Use "PS" for PostScript.
`filename`	The name of a file. To specify a file within the "printfiles" directory, use the format <filename>

Examples

The following example creates an anchor which prints two files—jack.ps and jill.ps—from within the default file folder:

html sample

```
<A HREF="kiosk:printfile!filetype=PS&filename=jack.ps&
filetype=PS&filename=jill.ps>print jack and jill</A>
```

Note: There are two options for using PostScript fonts:

- Use only those fonts available on the specific printer. See"PostScript Fonts" in Appendix B for a list of printer fonts.
- Create the PostScript documents with the non-standard fonts you want to use embedded within the actual document. The downside to this approach is that each document will be approximately 100-200kb larger for each font included.

kiosk:printpage

The kiosk:printpage command prints the current frame in a Web page. If there are multiple frames on the page, only the current frame (the one containing the kiosk:printpage command) is printed. Terminals which do not have a laser printer configured ignore this tag.

html syntax

```
kiosk:printpage
```

Example

In this example, the page is printed when the phrase **`Print this page`** is clicked:

html sample

```
<A HREF="kiosk:printpage">Print this page</A>
```

kiosk:card

The kiosk:card command is the most complex of the API commands. When this command is triggered, the user is asked to insert a card.

Once the card is read, the terminal reads information from the card and from this command and sends them to the server callback URL, a pre-defined server to which transactions are sent. See Chapter 7 for more information on the card clearing process.

html syntax

```
kiosk:card?service_symbol=SERVSYM&do_clear=[1/0]&amount=##&
trans_type=0&prompt=promptstr&operator_id=###&
[product information]
```

Table 6-5 explains the fields in the kiosk:card command.

Table 6-5: kiosk:card

field	**definition**
`service_symbol`	This is assigned by the system administrator to identify this service. It is 3 to 8 characters, all in upper case.
`do_clear=[1/0]`	This tells the terminal who should clear the card. A value of 0 means the content provider's server clears the card. A value of 1 means the terminal does it.
`amount`	This is the dollar amount of the transaction. The decimal is implied to be before the second last digit; 10000 has a value of 100.00.

field	**definition**
`trans_type`	Transaction type. Set to 0 for Sale.
`prompt`	Optional card prompt message. If no message is entered here, the default is used.
`auth_no`	Optional authorization number. Used for COMPLETE or FORCE credit transactions where a previous PRE_AUTH or AUTH_ONLY transaction provided this authorization number.
`operator_id`	Reserved for future use.
`product information`	Specific to the service provider and service. If the vendor requires the customer to enter information such as delivery address, buyer name, model number, etc. in a form, the product information passed in the message could be the form data from the HTML page. The service provider may also wish to include its own reference number in this field for cross-checking. It is also possible to put the commands to create a token receipt in the product information section. The product information tag is implicit. After all the other commands have been entered, all remaining data is assumed to be part of product information. You can include as many fields as necessary in the product info section.

Examples

In the following example, an anchor is used to trigger a transaction. For this transaction:

- The service symbol is NUTS.
- Nortel clears the card.
- The transaction amount is $15.50.
- This is a sale transaction.
- The product information contains two fields: walnuts and peanuts.

- There is no prompt, no authorization number, and no operator ID for this transaction.

html sample

```
<A HREF="kiosk:card?service_symbol=NUTS&do_clear=1&
amount=1550&trans_type=0&prompt=&operator_ID=&clear_status=&wa
lnuts="2.5kg"&peanuts="1.0kg">Buy some walnuts and peanuts.</A>
```

In the following example, an anchor is used to trigger another transaction. For this transaction:

- The service symbol is NUTS.
- The Vendor clears the card.
- The transaction amount is $10.00.
- This is a sale transaction.
- The prompt displayed is "Insert your card to buy some cashews."
- The product information contains the cashews field.

html sample

```
<A HREF="kiosk:card?service_symbol=NUTS&do_clear=0&
amount=10.00&trans_type=0&prompt="Insert your card to buy some
cashews"&operator_ID=&clear_status=0&cashews="1.0 kg">Buy some
cashews.</A>
```

7 Transactions

This chapter explains how to develop transactions. Transactions require server applications (eg. Perl scripts), so a programmer may need to write these applications. However, there is also a lot of work that must be done in HTML.

How transactions work

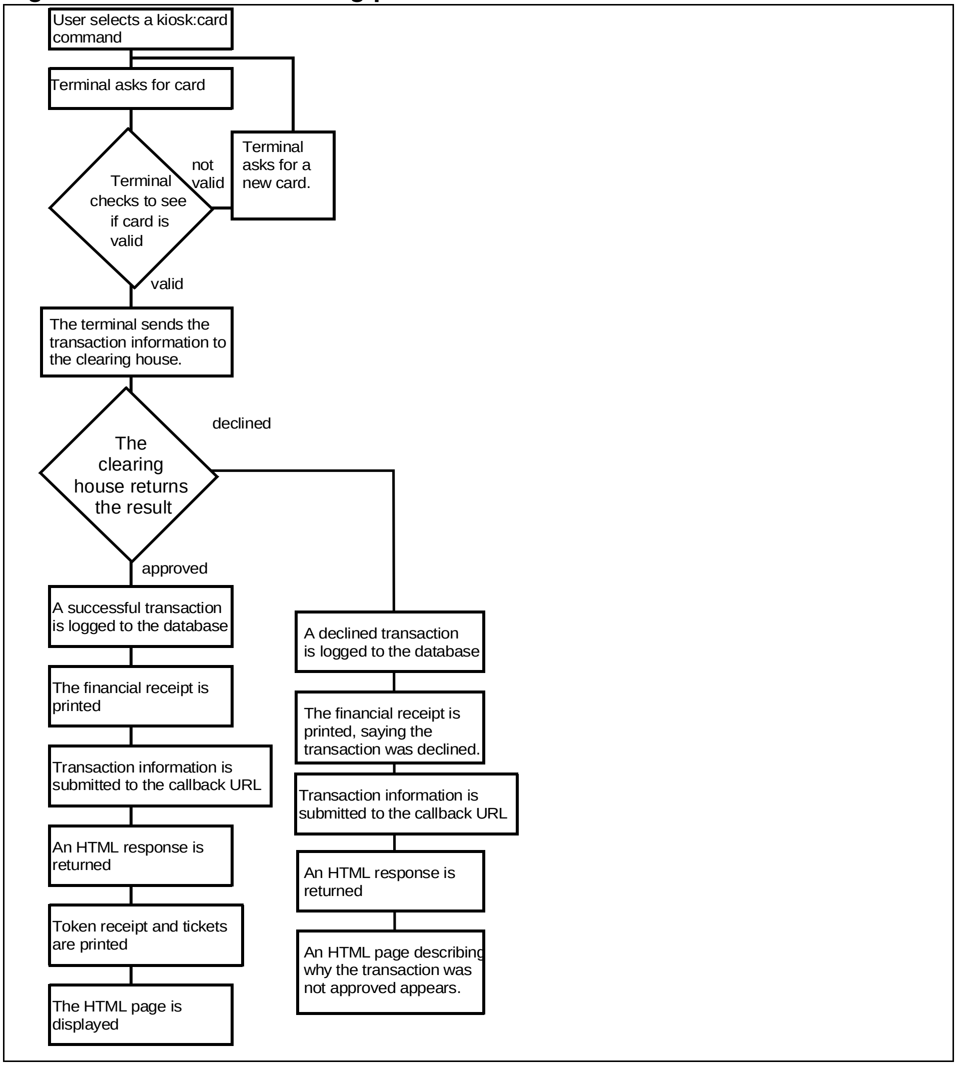
The following sections provide a high-level overview of how transactions are processed. The process depends on whether the terminal or the service provider clears the card.

The following two sections outline these processes.

Terminal clears the card

Figure 7-1 provides an overview of how transactions are cleared when the terminal clears the card.

Figure 7-1: Terminal clearing process


User selects a kiosk:card command
Terminal asks for card
Terminal checks to see if card is valid
not valid
Terminal asks for a new card.
valid
The terminal sends the transaction information to the clearing house.
The clearing house returns the result
declined
approved
A successful transaction is logged to the database
The financial receipt is printed
Transaction information is submitted to the callback URL
An HTML response is returned
Token receipt and tickets are printed
The HTML page is displayed
A declined transaction is logged to the database
The financial receipt is printed, saying the transaction was declined.
Transaction information is submitted to the callback URL
An HTML response is returned
An HTML page describing why the transaction was not approved appears.

Steps:

1. The user selects a kiosk:card link on a Web page. The

Web browser sends the command to the terminal software. For this step, you must set up a kiosk:card command, as explained on page 6-12.

2. The terminal prompts the user to insert a card. The user inserts and removes the card, and the terminal software reads it.
   For smart cards, the card is processed after it is inserted. For all other card types, it is processed after it is removed.

3. The terminal software clears the transaction. In the case of external validation services, this may require communication with a clearing house.

4. The terminal logs the transaction to the NetVenue management system.

5. The terminal calls the service's callback URL with the result of the transaction, in the form of a query string. The query string is generated by the kiosk:card command. If there is any product information in the kiosk:card command, this is sent in the query string.

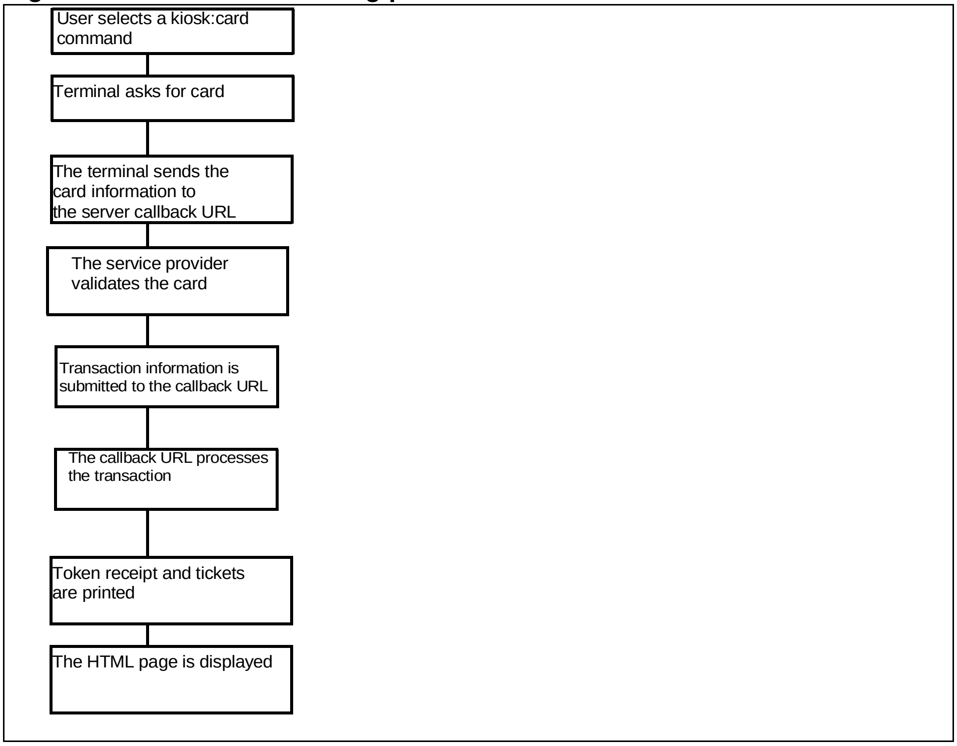
6. A script (in the form of a perl script, active server page, or servlet) on the server retrieves the result of the transaction and sends back a response in HTML format to the terminal. Part of the HTML is a comment which contains receipt and ticket information. The remainder of the HTML is the page to be displayed. If the transaction was declined, then the callback URL displays a message explaining this to the user.
   For this step, a script must be set up, as explained on page 7-8.

7. The terminal software prints the tickets and receipt based on the information sent from the server, and displays the HTML portion on-screen.

Service provider clears card

Figure 7-2 provides an overview of how transactions are cleared using the service provider's clearing facilities.

Figure 7-2: Customer clearing process


User selects a kiosk:card command
Terminal asks for card
The terminal sends the card information to the server callback URL
The service provider validates the card
Transaction information is submitted to the callback URL
The callback URL processes the transaction
Token receipt and tickets are printed
The HTML page is displayed

Steps:

1. The user selects a kiosk:card link on a Web page. The Web browser sends the command to the terminal software. For this step, you must set up a kiosk:card command, as explained on page 6-12.

2. The terminal prompts the user to insert a card. The user inserts and removes the card, and the terminal software reads it.

3. The terminal calls the service's callback URL with the result of the transaction, in the form of a query string. The query string is generated by the kiosk:card command. If there is any product information in the kiosk:card command, this is sent in the query string.

4. A script(in the form of a perl script, active server page, or servlet) on the server uses the card information to process the transaction and determine whether the transaction is approved or declined, and generate the authentification number.
   It then sends back a response in HTML format to the terminal. Part of the HTML is a comment which contains receipt and ticket information. The remainder of the HTML is the page to be displayed. If the transaction was declined, then the callback URL displays a message explaining this to the user.
   For this step, a script must be set up, as explained on page 7-8.

5. The terminal software prints the tickets and receipt based on the information sent from the server, and displays the HTML portion.

Card prompt dialogue

Once the terminal software receives a kiosk:card command, it displays the card prompt dialogue box and then reads the user's card. The types of cards supported by the service provider determine the card graphics displayed in the dialogue. This is configurable through the Admin Tools Suite.

If additional cards are used, such as student cards or loyalty cards, you need to supply artwork for them too.

A default text prompt appears, which is configurable using the administration tools. For example, the administrator could change the text to read "Please insert and remove your student card."

Consult the *User interface customization guide* for information on this.

Call to server callback URL

Once the card has been read, the software appends information to the product information fields and passes the entire string to the service provider's server in the form of a query string. This query string is generated by the kiosk card command, and is sent to the IP address of the callback URL which is set in the Admin Tools software when a service is defined.

The callback URL should contain a script which returns a response to the terminal. See 7-8 for information on setting up the callback URL.

Some fields in the call to server callback URL depend on whether the terminal or the service provider cleared the transaction.

html sample

```
http://callbackURL?[product information]&k_svc_symbol=######
&k_terminal=####&k_cardnum=
########&k_expdate=MMYY&k_authnum=####&k_refnum=########
[&k_termid=#######]&k_result=####&k_do_clear=1
```

Table 7-1 gives a more detailed description of each field.

Table 7-1: call to server callback URL

field	**description**	**When it is used**
`http://call-backURL`	The service provider's server call-back URL, as defined in the terminal database. In most cases this would be a CGI.	Always used
?	Indicates that the data which follows is passed to the result URL	Always used

field	**description**	**When it is used**
`product information`	The same product information data that was passed inside the corresponding “kiosk:card” request. Set to ’NULL’ if no product info was specified.	Always used
`&`	Field separator	Always used
`k_svc_symbol`	Upper case service symbol assigned to this service by the administrator	Always used
`k_terminal`	The terminal (kiosk) number	Always used
`k_cardnum`	The card number that was read	Always used
`k_expdate`	The expiration date (does not include the ’/’ separator) in the form `mmyy` where `mm` is the month and `yy` is the year.	Always used
`k_authnum`	If the transaction was successful then this field contains the authorization number for the transaction.	Used only if terminal is clearing
`k_refnum`	This field contains the transaction reference number generated by the terminal for credit or debit transactions that the terminal cleared.	Used only if terminal is clearing
`k_termid`	This field contains the clearing house or financial institution’s Terminal ID used in the credit/debit transaction.	Used only if terminal is clearing
`k_result`	This field contains the result code of the transaction. If this field is set to 0, it was a successful transaction and an authorization number is provided in the k_authnum field. If k_result was not 0, then the transaction failed and the k_authnum field is empty. If the content provider clears the card: This field will be empty, unless there is a card reader error, in which case the value will be 10000 to 10003.	Used only if terminal is clearing

field	**description**	**When it is used**
`k_track2data`	This is the ASCII Track II data from the card swipe.	Used only if the service provider is clearing
`k_do_clear`	Indication of where the clearing was processed: A 0 means the service provider cleared the card. A 1 means that the terminal cleared the card.	Always used

CAUTION ⚠	When you are setting up a server to interact with your terminals, there are many potential security risks. • A CGI script that is not done properly can create a serious security threat. This is particularly dangerous with forms, where hackers can input code into which allow them to gain access to the server, or affect the server in other ways. • It is highly recommended that you use kiosk:commands in secure (SSL) pages to protect privacy information.

Callback URL

After a card is processed and the transaction is accepted or declined, the terminal browser opens the callback URL for the transaction. Each service may have slightly different needs for a callback URL, depending on the nature of the transaction, the setup of the server, and the methods of the service provider.

The callback URL is a script which takes the information provided by the terminal after a card is read and provides a response to the terminal. If the transaction is cleared by the terminal, the result of the transaction is also processed. Several different technologies can be used to create the callback URL, including:

- PERL

- Active server pages (ASP)
- Java

The callback URL has the following purposes:

- It creates a page which is displayed on the terminal after the transaction.
- It creates a command to print a receipt or ticket.
- It specifies the text strings for dialogue boxes.
- It informs the terminal that the query string was received.
- It typically generates different responses for successful or failed transactions.
- It is often responsible for logging transaction information to a database.

For any kiosk:card command, you must create a callback URL. Appendix B shows a sample ASP callback URL, along with an explanation of the different components of it.

Server response

Once the server callback URL is contacted, it responds by returning an HTML page, which contains the callback result. After calling the server, the terminal waits for a response. This response tells the terminal the result of the transaction, and gives it information to display or print. The following is the format of the HTML response.

html sample

```
<html>
<head>
<title>response</title>

<!-- KIOSK_CALLBACK_RESULT
parameters
-->
</head>
```

```
<body>
<!-- web page content -->
</body>
</html>
```

In the above syntax, the body of the HTML is the Web page that is displayed. The "parameters" section contains all the information about the transaction which the terminal requires. These parameters must be contained within the KIOSK_CALLBACK_RESULT comment, which must be located in the head of the HTML.

The next section contains a more detailed explanation of these parameters.

Parameters

The parameters consist of information about the transaction, and provide the text to be printed out on receipt or tickets, if applicable.

Syntax

html sample

```
approval_status=[1|0]&service_symbol=service symbol&
approval_msg=approval message&receipt_format=
[RAW|TOKEN]&receipt_info=receipt info&ticket_format=
[RAW|LANDSCAPE_A]&ticket_refno=#&ticket_rows=#&ticket_cols=
&ticket_info=data
```

Table 7-2 explains the fields in the parameters section

Table 7-2: Parameters

Field	**Description**
`approval_status`	if `approval status=0` Transaction failed. This can result from incomplete form information or credit card denial. This is for the content provider to determine.
`service_symbol`	An uppercase service symbol to uniquely identify the product/service purchased.
`approval_msg`	Text message to be displayed to customer in a dialogue box.

`ticket_format`	Optional field if tickets are to be printed. This determines the format of the ticket. The two possible values are:
	`RAW`
	`LANDSCAPE_A`
	These formats are explained in the ticket_info section below.

`ticket_refno`	Ticket reference number supplied by content provider for their purposes. This number is stored, but not checked for uniqueness.
`ticket_rows`	Number of rows on the ticket. Used with `LANDSCAPE_A` mode only.
`ticket_cols`	Number of columns on the ticket. Used with `LANDSCAPE_A` mode only.

`ticket_info`	Vendor-specific information to create tickets on the Terminal's ticket printer. There is one `&ticket_info+data` parameter for each ticket to be printed.
	`RAW`
	These are the untouched printer command strings that the terminal sends directly to the printer.
	`LANDSCAPE_A`
	These strings are the ISO 8859-1 Latin characters. See the appendix B.

`receipt_format`	Valid values are `RAW` or `TOKEN`. This field defines the format of the data in the `receipt_info` field.

Field	**Description**
`receipt_info`	Service provider-specific information to create a receipt for the transaction. After the tickets have been processed, the receipt printer string is then sent to the receipt printer. The data in this field can vary depending on the value of the `receipt_format` field: If `receipt_format` is `RAW` receipt_info contains a string or raw printer data sent directly to the receipt printer. This requires knowledge of the printer's command language. If `receipt_format` is `TOKEN` `receipt_info` contains a tokenized list of parameters from which a standard receipt is then generated. This method requires no knowledge of the type of printer being used and guarantees a standard receipt.

Note: Following are some things to be aware of when printing tickets:

- The order of the tags is fixed.
- All fields starting with ticket_ are mandatory if printing tickets. Otherwise they can be omitted.
- Multiple tickets can be printed in a single server response.
- The ticket print job goes to the default ticket printer, if ticket printers exist.
- There is no validation of the information in the ticket_info field. It is up to the content provider to ensure that the ticket string fits into the rows and column specified in the command.

Following are some things to be aware of when printing receipts:

- A single receipt can be printed in a single server response.
- Using RAW format should be avoided, since any change to the model or type of printer could require changes to the server script/CGI.
- Do not use the new line character (%0A) within the

receipt information for Token format.

Examples

Following is an example of an HTML response that prints a tokenized receipt and displays an approval message saying “Approved”:

html sample

```
<html>
<head>
<title>response</title>
<!--KIOSK_CALLBACK_RESULT
approval_status=1&service_symbol=nortel&approval_msg=
Approved&receipt_format=TOKEN&receipt_info=&lang=1&
prvdname=Nortel&srvname=Shoe and Key Sales
&qty1=1&item1=Keys&price1=10.00&qty2=2&
item2=Shoes&price2=2599&subtotal=
35.99&tax1=PST&tax1desc=8%&tax1amt=1.20&tax2=GST&tax2des
c=7%&tax1amt=1.05&extradesc=Shipping   and   Handling&ex-
traamt=500&
totalamt=4300&termid=99NORTEL&refno=1234567&
cardno=512123456789&expdate=1097&authno=98765&gst-
no=R12345678&extra=20% off your next purchase&
ticket_format=LANDSCAPE_A&ticket_refno=1000&
ticket_rows=2 &ticket_cols=80&ticket_info=
This is a test ticket\n\n
-->
</head>
<body>
<b>Thank you for your purchase!</b>
</body>
</html>
```

The Web browser passes the contents of the `KIOSK_CALLBACK_RESULT` tag to the terminal software.

The following is displayed in a message box:

Approved

The output of this HTML on the display is:

Thank you for your purchase!

The following is used to print a ticket from the ticket printer:

html sample

```
ticket_format=LANDSCAPE_A&ticket_refno=1000&ticket_rows
=2 &ticket_cols=80&ticket_info=This is a test ticket
```

The following is used to creat a receipt:

html sample

```
receipt_format=TOKEN&receipt_info=&lang=1&prvd-
name=Nortel&srvname=Shoe and KeySales&qty1=1&item1=
Keys&price1=10.00&qty2=2&item2=Shoes&price2=2599&subto-
tal=
35.99&tax1=PST&tax1desc=8%&tax1amt=1.20&tax2=GST&tax2des
c=7%&
tax1amt=1.05&extradesc=Shipping and Handling&extraamt=
500&totalamt=4300&termid=99NORTEL&refno=1234567&
cardno=512123456789&expdate=1097&authno=98765&gstno=
R12345678&extra=20% off your next purchase
```

The following section explains how the receipt printer formats a message.

Tokenized receipts

This section explains how to create tokenized receipts, and how receipts are formatted.

A tokenized receipt is created by assigning values to the tokens in the receipt info section of the server response. Tokenized receipts print out similarly on any printer type.

Syntax

html syntax

```
receipt_info=token1=value1[&token2=value2] ...
[&tokenn=valuen]
```

Tokens are short names that represent a part of the receipt. For example, `srvname` is the token for service name.

The receipt formats automatically, so all you need to do is assign values to the tokens. The format for the receipt is shown in Figure 7-3.

Figure 7-3: Receipt

```
              <Provider Name>
              <Service Name>

DATE: YY/MM/DD            TIME: HH:MM:SS

<QTY1> <Item1 Desc>             <Price1>
 :
<QTYn> <Itemn Desc>             <Pricen>

       SUBTOTAL                 <SubTotal>
       <Tax1>                   <Tax1Amt>
       <Tax2>                   <Tax2Amt>
       <Extra Cost>             <ExtraAmt>
                                --------
       TOTAL                    <TotalAmt>

TERMID: ########         REFNO: ########
CARD#:  #############      EXP: ####

<APPROVED - THANK YOU | NOT APPROVED (####)>
##########

GST ##########

<EXTRA>
```

Each field in the receipt above has a token and a maximum width. These are shown in Table 7-3.

Table 7-3: Receipt fields

Receipt Field	Token	Width	Description
	`lang`		Language of receipt field labels (1=English, 2=Canadian French). Not shown on actual receipt but sent as a token.
`<Provider Name>`	`prvdname`	30	Name of the content provider. If not provided, taken from database.
`<Service Name>`	`srvname`	30	Name of the service provided. If not provided, taken from database.
`YY/MM/DD`	`date`	8	Date of transaction. Filled in automatically based on the terminal's date.
`HH:MM:SS`	`time`	8	Time of transaction (shown in 24 hour clock). Filled in automatically based on terminal's time.
`<QTY1> … <QTYn>`	`qty1 … qtyn`	4	Quantity
`<Item1 Desc> … <Itemn Desc>`	`item1 … itemn`	160	Description of service or product (e.g. PassCard Time)
`<Price1> … <Pricen>`	`price1 … pricen`	7	Dollar amount ("$" is not shown). Include decimal point.
`<SubTotal>`	`subtotal`	7	Subtotal amount. "$" not shown. Include decimal point.
`<Tax1>`	`tax1`	4	Tax1 type (e.g. "PST")
`<Tax1 Amt>`	`tax1amt`	7	Tax1 amount – '$' not shown (e.g. "1.10"). Include decimal point.
`<Tax2>`	`tax2`	4	Tax2 type (e.g. "GST")

Receipt Field	**Token**	**Width**	**Description**
`<Tax2 Amt>`	`tax2amt`	7	Tax2 amount – "$" not shown (e.g. “1.10”). Include decimal point.
`<Extra Cost>`	`extra-desc`	160	Description of an extra cost (e.g. “Shipping and Handling”) which is not taxable. Can also be included in itemized list
`<ExtraAmt>`	`extraamt`	7	Extra cost amount. "$" not shown. Include decimal point.
`<TotalAmt>`	`totalamt`	7	Total amount. "$" not shown. Include decimal point.
`TERM_ID`	`termid`	8	Terminal ID
`REFNO`	`refno`	8	Reference number for this transaction (generated by Nortel Networks.
`CARD#`	`cardno`	20	Credit card number
`EXP`	`expdate`	4	Expiry Date (e.g. 0897)
`NOT APPROVED (####)`	`reason`	4	The result code replaces the "####". This field is used only if the transaction does not go through, and approval status is set as 0.
`AUTH#`	`authno`	10	Authorization number (from clearing house)
`GST`	`gstno`	9	Optional GST registration number (e.g. “GST R123456789”)
`EXTRA`	`extra`	160	Free-form extra information (e.g. “20% discount off next purchase”)

In the previous section, the following was sent to the receipt printer:

html sample

```
receipt_format=TOKEN&receipt_info=&lang=1&prvd-
name=Nortel&srvname=Shoe and KeySales&qty1=1&item1=
Keys&price1=10.00&qty2=2&item2=Shoes&price2=2599&subto-
tal=
35.99&tax1=PST&tax1desc=8%&tax1amt=1.20&tax2=GST&tax2des
c=7%
&tax1amt=1.05&extradesc=Shipping    and    Handling&ex-
traamt=500&
totalamt=4300&termid=99NORTEL&refno=1234567&
cardno=512123456789&expdate=1097&authno=98765&gstno=
R12345678&extra=20% off your next purchase
```

Figure 7-4 shows output of this script:

Figure 7-4: credit receipt

```
            Nortel
            Shoe and Key Sales

DATE: YY/MM/DD            TIME: HH:MM:SS

1      Keys                     10.00
  :
2      Shoes                    25.99

       SUBTOTAL                 61.98
       PST 8%                    4.34
       GST 7%                    4.09
       Shipping and handling     5.00
                                --------
       TOTAL                    75.41

TERMID: 99NORTEL         REFNO: 1234567
CARD#:  512123456789      EXP: 0402

APPROVED - THANK YOU
AUTH #98765

GST R12345678

20% off your next purchase
```

8 Sample application

This chapter follows the development of a simple service which allows users to purchase tickets for a shuttle. It allows users to change the number of tickets, print off directions to the shuttle and departure times, and dial the shuttle information line.

The NetVenue terminal interface allows for different approaches to creating sites. This example uses frames, JavaScript, and a Perl script. Many other approaches to site development work just as well. The frames, for example, can be avoided by using DHTML.

This site will consist of two pages. The first page will allow the user to access various services, and the second is lets the user know that a transaction was successful.

Designing the service

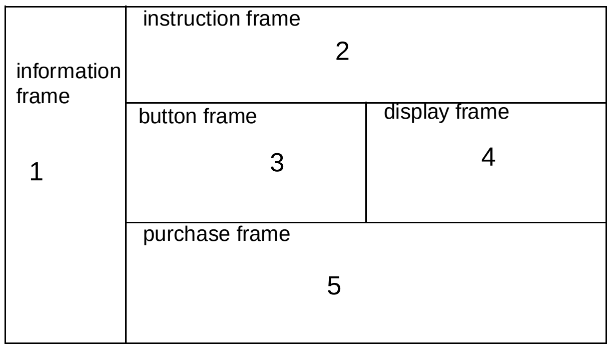
There are different ways of designing sites. This initial design of this site was done in Photoshop. This ensured that the site would fit properly in the 515 by 800 browser window. After the page was designed, it was broken down into frames that would make sense in HTML.

Figure 8-1 shows the layout of the service, with each frame numbered. Below is a description of each frame.

1. One frame on the left side of the screen will display general information about the shuttle service, the logo, and links to printing shuttle information, and dialing the shuttle information line. This frame will be called the *information frame*.

The remainder of the page will be divided into three rows, with the middle row containing two columns. Figure 8-1 shows the division of frames on the page.

Figure 8-1: service layout



2. The top frame contains information on using this page to select the number of tickets. This will be called the *instruction frame.*

3. The center-right frame contains up and down buttons the user can touch to change the number of tickets selected. This frame will be called the *button frame.*

4. The center-left frame displays the number of tickets selected and their cost. This frame will be called the *display frame.*

5. The bottom frame contains a button the user can press to purchase the tickets. This will be called the *purchase frame.*

Creating the frameset

This site uses three JavaScript variables. "Tickets" is the number of tickets purchased. "Cost" is the cost sent for the transaction. Cost2 is the cost that is sent for the token receipt. Because these variables are used by multiple frames, it is best to specify them in the frameset file, making them global variables. The head segment of the frameset file is as follows:

html sample

```
<html>
<head>
  <title>shuttleframe</title>
  <SCRIPT LANGUAGE="JavaScript"><!--
      tickets="";
      cost="";
      cost2="";
      //-->
  </SCRIPT>
</head>
```

The actual frameset follows the head and precedes the body. Notice that in each frameset, the borders are turned off, and framespacing is set to 0. In each frame, scrolling and resize are turned off. This code creates three framesets. The first divides the page into two columns, the second divides the right column the three rows, and the final divides the middle row into two columns.

html sample

```
<frameset rows="" cols="275,*" frameborder="0" border=0
framespacing="0">
   <frame src="info.htm" name="information" scrolling="No"
noresize marginwidth="0" marginheight="0">
   <frameset rows="*,43%,50%" frameborder="0" border="0"
framespacing="0">
     <frame src="instruct.htm" name="instruction" scroll-
ing="No" marginwidth="0" marginheight="0">
```

```
     <frameset   cols="60%,*"   frameborder="0"   border="0"
framespacing="0"  scrolling="no">
       <frame src="display1.htm" name="display" scrolling="no"
noresize marginwidth="10" marginheight="10">
       <frame src="buttons.htm" name="buttons" scrolling="no"
noresize marginwidth="0" marginheight="0">
     </frameset>
     <frame src="purchase.htm" name="purchase" scrolling="No"
noresize marginwidth="0" marginheight="0">
   </frameset>
</frameset>
```

Following this is the body section, which is blank. It is a common practice to put text here for browsers which do not support frames, but for Web Terminal content, this is not necessary. The body section is as follows:

html sample

```
<body>
</body>
</html>
```

Creating the information frame

The first frame is the left column, which contains general information about the shuttle service, the logo, links to printing shuttle information, and a link for dialing the shuttle information line.

Creating the kiosk:printfile command

The printfile command contains parameters telling the teminal that the type of file is postscript, and the file name is schedule.ps. These parameters are placed in an anchor tag.

html sample

```
<html>
<head>
<title>information</title>
</head>
<div align="center">
<body bgcolor="White"><font face="Comic Sans MS" size="+1" col-
or="navy">
Click the button below to print a schedule of shuttle departure
times.
<a href="kiosk:printfile!filetype=PS&filename=schedule.ps">
<img src="button.jpg" width=104 height=85 border="0"></a><p>
```

Creating the kiosk:phone command

The phone command will appear immediately after the printfile command.

html sample

```
Click the button below to be connected with a shuttle service
operator.
<a href="kiosk:phone!SHUTTLE!5554626!001">
<img src="button2.jpg" width=104 height=85 border="0"></a>
</body>
</html>
```

Creating the instruction frame

The instruction frame is the simplest to create:

html sample

```
<html>
<head>
<title>instruction</title>
```

```
</head>
<div align="center">
<body bgcolor="White"><font face="Comic Sans MS" size="+1" col-
or="navy">
How many shuttle tickets do you want?
</font>
</div>
</body>
</html>
```

This file is saved as instruct.htm, so that it will be opened by the frameset.

Creating the button frame

The button frame contains JavaScript operators for which changes the values of the global variables. The head should contain two JavaScript functions.

- **`Upwards()`** increases the value of tickets by one, and the value of cost by 10, and then reloads display.htm, with the new variables.
- **`Downwards()`** decreases the value of tickets by one and the value of cost by 10. **`Downwards()`** also contains an if-else statement so that the user cannot select a negative amount of tickets. The value of tickets must be greater than one for the user to decrease the number of tickets.

html sample

```
<html>
<head>
   <title>button</title>
   <script language=Javascript><!--
     function upwards(){
       parent.tickets++;
       parent.cost=parent.tickets*10;
       parent.frames[2].location.href="display.htm";
     }
     function downwards() {
       if(parent.tickets > 1)
       {parent.tickets--;
       parent.cost=parent.tickets*10;{
       parent.frames[2].location.href="number.htm";}
       else
       { }
     }
   //--></script>
</head>
```

The body of the frame contains two buttons which call the functions.

html sample

```
<body bgcolor="White">
<A href="javascript:upwards();">
<img src="up.jpg" width=104 height=85 border="0"></a>
<br>
<A href="javascript:downwards();">
<img src="down.jpg" width=104 height=83 border="0"></a>
</body>
```

Creating the display frame

The display frame displays the number of tickets selected and their cost.

html sample

```
<html>
<head>
<title>number</title>
</head>
<body bgcolor="White">
<div align="center"><br>&nbsp&nbsp&nbsp<font face="Comic Sans
MS" size="+5" color="navy">
<SCRIPT language=JavaScript>
document.write(parent.tickets)
</script><br>
$
<SCRIPT language=JavaScript>
document.write(parent.cost)
</script>
</font>
</div>
</body>
</html>
```

The only problem with this page is that the numbers are undefined when the page initially loads. So the display reads

Undefined

$ Undefined.

To get around this problem, a second display frame must be created. This one will display the number of tickets and price as zero.

html sample

```
<html>
<head>
<title>number</title>
</head>
<body bgcolor="White">
<div  align="center"><br>&nbsp&nbsp&nbsp<font  face="Comic  Sans
MS" size="+5" color="navy">
0<br>
$0
</font>
</div>
</body>
</html>
```

Creating the purchase frame

The purchase frame takes the information the user has selected and creates a dynamic kiosk:card command. This frame uses a function which links the kiosk:card command with the variables selected.

The JavaScript for this page is as follows:

html sample

```
<html>
<head>
<title>purchase</title>
<script language="javascript"><!--
function card(){
parent.cost=parent.tickets*1000
parent.cost2=parent.tickets*10
mystring="kiosk:phone?service_symbol+AIRSHUTT&do_clear=1&
amount=" + parent.cost1 + "&trans_type=0& prompt =&operator_id=
&clear_status=0&receipt_format=TOKEN &qty=" +  parent.tickets +
"&item1= tickets&price1="  + parent.cost2 +  "&extradesc=";
escaped=escape(mystring);
window.location.href=escaped
}
//--></script>
```

- The first two lines of the function define the variables **`cost`** and **`cost2`**. Cost will be sent to calculate the receipt, and is the amount deducted from the credit card. **`Cost2`** is sent to a Perl script, which will print out a Token receipt.
- The third line of the function combines the kiosk command and variables and stores them as **`mystring`**.
- The fourth line performs the escape function, which protects illegal characters such as colons or question-marks from being misinterpreted.
- The final line sends this new variable to the browser.

The body of the page must call this function as a link.

html sample

```
<div align="center">
<body bgcolor="White"><font face="Comic Sans MS" size="+1" col-
or="navy">
To purchase your ticket(s), press continue.<br>
You will be asked to insert a card for payment.<br>
The terminal will print your ticket(s)<br>
and a receipt.<p>
<font color="white">
<A  href="javascript:card();"><img  src="contin.jpg"  width=132
height=77 alt="" border="0"></A>
</div>
</font>
</body>
</html>
```

On this page, the **`card()`** function is called when a user presses the continue button.

9 Security

Security is important on the NetVenue terminal system. Even though the NetVenue terminal is designed with security as a priority, there are several ways that server administrators and service developers can ensure that the system remains secure.

Digital certificate contents

The X.509 compliant digital certificate that is installed in the terminal contains the following information:

- O=Advanced Public Access Terminals
- OU=Nortel.Access
- IO=Nortel Partners
- CN=Nortel Kiosk Northern Telecom
- Key Algorithm=rsaEncryption
- Key Size=512
- Signature Algorithm=RSA-MD5

Please refer to the documentation provided with your Web server to determine how you can obtain the information from the terminal's digital certificate.

As an example, on a Stronghold Windows NT Web Server (version 2.0b1 – beta version), the following environment variables are set to indicate the contents of the kiosk client digital certificate:

- SSL_CLIENT_O=Advanced Public Access Terminals
- SSL_CLIENT_OU=Nortel.Access
- SSL_CLIENT_IO=Nortel Partners
- SSL_CLIENT_CN=Nortel Kiosk Northern Telecom

The important fields to check are that the IO contains the "Nortel Partners" information and that the CN contains the "Nortel Kiosk Northern Telecom" information. This ensures that the issuing CA was Nortel and that the particular certificate is that of a Nortel kiosk.

CAUTION ⚠	The Web server owner must ensure that its Web server does not contain any untrusted CA certificates. This would allow an imposter to generate a client certificate with the above information and have it signed by an untrusted CA. By doing this an imposter could pass its fake certificate to the server and the server would accept it as valid. The server content would then check for the important fields and would think it was communicating with a kiosk.

Appendix A: ISO8859-1 (Latin 1) Character Set

Use the **Char** column (ISO Latin 1) for token formatted receipts. Use the “437” column for RAW receipts.

Table A-1: Printer characters

Char	437	Hex	Description
		09	Horizontal tab
		0A	Line feed
		20	Space
!	!	21	Exclamation mark
"	"	22	Quotation mark
#	#	23	Number sign
$	$	24	Dollar sign
%	%	25	Percent sign
&	&	26	Ampersand
`	`	27	Apostrophe
(	(	28	Left parenthesis
)	)	29	Right parenthesis
*	*	2A	Asterisk

Char	437	Hex	Description
+	+	2B	Plus sign
,	,	2C	Comma
-	-	2D	Hyphen
.	.	2E	Period (full stop)
/	/	2F	Solidus (slash)
0	0	30	Digit 0
1	1	31	Digit 1
2	2	32	Digit 2
3	3	33	Digit 3
4	4	34	Digit 4
5	5	35	Digit 5
6	6	36	Digit 6
7	7	37	Digit 7
8	8	38	Digit 8
9	9	39	Digit 9
:	:	3A	Colon
;	;	3B	Semicolon
<	<	3C	Less than
=	=	3D	Equal sign
>	>	3E	Greater than
?	?	3F	Question mark
@	@	40	Commercial at
A	A	41	Letter A
B	B	42	Letter B

Char	437	Hex	Description
C	C	43	Letter C
D	D	44	Letter D
E	E	45	Letter E
F	F	46	Letter F
G	G	47	Letter G
H	H	48	Letter H
I	I	49	Letter I
J	J	4A	Letter J
K	K	4B	Letter K
L	L	4C	Letter L
M	M	4D	Letter M
N	N	4E	Letter N
O	O	4F	Letter O
P	P	50	Letter P
Q	Q	51	Letter Q
R	R	52	Letter R
S	S	53	Letter S
T	T	54	Letter T
U	U	55	Letter U
V	V	56	Letter V
W	W	57	Letter W
X	X	58	Letter X
Y	Y	59	Letter Y
Z	Z	5A	Letter Z

Char	437	Hex	Description
[	[	5B	Left square bracket
\	\	5C	Reverse solidus (backslash)
]	]	5D	Right square bracket
^	^	5E	Caret
-	-	5F	Horizontal bar
`	`	60	Grave accent
a	a	61	Letter a
b	b	62	Letter b
c	c	63	Letter c
d	d	64	Letter d
e	e	65	Letter e
f	f	66	Letter f
g	g	67	Letter g
h	h	68	Letter h
i	i	69	Letter i
j	j	6A	Letter j
k	k	6B	Letter k
l	l	6C	Letter l
m	m	6D	Letter m
n	n	6E	Letter n
o	o	6F	Letter o
p	p	70	Letter p
q	q	71	Letter q
r	r	72	Letter r

Char	437	Hex	Description
s	s	73	Letter s
t	t	74	Letter t
u	u	75	Letter u
v	v	76	Letter v
w	w	77	Letter w
x	x	78	Letter x
y	y	79	Letter y
z	z	7A	Letter z
{	{	7B	Left curly brace
|	|	7C	Vertical bar
}	}	7D	Right curly brace
~	~	7E	Tilde
¡	¡	A1	Inverted exclamation point
¢	¢	A2	Cent sign
£	£	A3	Pound sterling
¤		A4	General currency sign
¥	¥	A5	Yen sign
|	|	A6	Broken vertical bar
§		A7	Section sign
¨		A8	Umlaut (dieresis)
©		A9	Copyright
ª	ª	AA	Feminine ordinal
«	«	AB	Left angle quotation, guillemot left
		AC	Not sign

Char	**437**	**Hex**	**Description**
–	–	AD	Soft hyphen
®		AE	Registered trademark
¯		AF	Macron accent
°	°	B0	Degree sign
±	±	B1	Plus or minus
2		B2	Superscript two
3		B3	Superscript three
´		B4	Acute accent
µ		B5	Micro sign
¶		B6	Paragraph sign
·	·	B7	Middle dot
¸		B8	Cedilla
1		B9	Superscript one
º	º	BA	Masculine ordinal
»	»	BB	Right angle quotation, guillemot right
_	_	BC	Fraction one-fourth
_	_	BD	Fraction one-half
_		BE	Fraction three-fourths
¿	¿	BF	Inverted question mark
À	A	C0	Capital A, grave accent
Á	A	C1	Capital A, accute accent
Â	A	C2	Capital A, circumflex accent
Ã	A	C3	Capital A, tilde

Char	437	Hex	Description
Ä	Ä	C4	Capital A, dieresis or umlaut mark
Å	Å	C5	Capital A, ring
Æ	Æ	C6	Capital AE dipthong (ligature)
Ç	Ç	C7	Capital C, cedilla
È	E	C8	Capital E, grave accent
É	É	C9	Capital E, acute accent
Ê	E	CA	Capital E, circumflex accent
Ë	E	CB	Capital E, dieresis or umlaut mark
Ì	I	CC	Capital I, grave accent
Í	I	CD	Capital I, acute accent
Î	I	CE	Capital I, circumflex accent
Ï	I	CF	Capital I, dieresis or umlaut mark
_	D	D0	Capital Eth, Icelandic
Ñ	Ñ	D1	Capital N, tilde
Ò	O	D2	Capital O, grave accent
Ó	O	D3	Capital O, acute accent
Ô	O	D4	Capital O, circumflex
Õ	O	D5	Capital O, tilde
Ö	Ö	D6	Capital O, dieresis, or umlaut mark
x	x	D7	Multiply sign
Ø	Ø	D8	Capital O, slash
Ù	U	D9	Capital U, grave accent
Ú	U	DA	Capital U, acute accent

Char	437	Hex	Description
Û	U	DB	Capital U, circumflex accent
Ü	Ü	DC	Capital U, dieresis or umlaut mark
Y	Y	DD	Capital Y, acute accent
_		DE	Capital Thorn, Icelandic
ß	ß	DF	Small sharp s, German (sz liga-ture)
à	à	E0	Small a, grave accent
á	á	E1	Small a, acute accent
â	â	E2	Small a, circumflex accent
ã	a	E3	Small a, tilde
ä	ä	E4	Small a, dieresis or umlaut mark
å	å	E5	Small a, ring
æ	æ	E6	Small ae dipthong (ligature)
ç	ç	E7	Small c, cedilla
è	è	E8	Small e, grave accent
é	é	E9	Small e, acute accent
ê	ê	EA	Small e, circumflex accent
ë	ë	EB	Small e, dieresis or umlaut mark
ì	ì	EC	Small i, grave accent
í	í	ED	Small i, acute accent
î	î	EE	Small i, curcumflex accent
ï	ï	EF	Small i, dieresis or umlaut mark
¹	¹	F0	Small eth, Icelandic
ñ	ñ	F1	Small n, tilde

Char	437	Hex	Description
ò	ò	F2	Small o, grave accent
ó	ó	F3	Small o, acute accent
ô	ô	F4	Small o, circumflex accent
õ	õ	F5	Small o, tilde
ö	ö	F6	Small o, dieresis or umlaut mark
÷	÷	F7	Division sign
ø	ø	F8	Small o, slash
ù	ù	F9	Small u, grave accent
ú	ú	FA	Small u, acute accent
û	û	FB	Small u, circumflex accent
ü	ü	FC	Small u, dieresis or umlaut mark
y		FD	Small y, acute accent
_		FE	Small thorn, Icelandic
ÿ	ÿ	FF	Small y, dieresis or umlaut mark

Appendix B: Active server page callback URL

This appendix provides an example of an active server page that is used as a callback URL. There are many different ways of setting up a callback URL. This is only one possibility.

For more information on callback URLs, see page 7-8.

The first part of the callback URL is shown below. In this section, information is taken from the query string and turned into variables.

html sample

```
<html>
<head>
<title>Server Callback</title>
<%
ServiceSymbol = Request.QueryString("k_svc_symbol")
TerminalNumber = Request.QueryString("k_terminal")
CardNumber = Request.QueryString("k_cardnum")
ExpiryDate = Request.QueryString("k_expdate")
AuthNo = Request.QueryString("k_authnum")
ReferenceNo = Request.QueryString("k_refnum")
TerminalID = Request.QueryString("k_termid")
TransResult = Request.QueryString("k_result")
DoClear = Request.QueryString("k_do_clear")
Prompt = Request.QueryString("k_prompt")
Name = Request.QueryString("Name")
ESN = Request.QueryString("ESN")
```

```
ID = Request.QueryString("ID")
```

The second section of the callback URL changes any empty variables to ’0’. Fields that are not applicable will be left blank in the query string, but they must have a value of 0 at this point.

html sample

```
if TransResult = "0" then
Approval_Status = "1"
else
Approval_Status = "0"
end if

if AuthNo = "" then
AuthNo = 0
end if

if CardNumber = "" then
CardNumber = 0
end if

if ExpiryDate = "" then
ExpiryDate = "1/1/1999"
end if
```

The next part of the callback URL logs the transaction to the a database. This code is different for various types of databases. Use of this also requires you to set up a database.

html sample

```
' ADD THIS ENTRANT TO THE DATABASE
DateTime = now()
Set cn = Server.CreateObject("ADODB.Connection")
Set rec = Server.CreateObject("ADODB.Recordset")
cn.Open "dsn=contestdb"
```

```
rec.ActiveConnection = cn
rec.CursorType = 3
rec.LockType = 2
rec.Open "Entries"
If rec.EOF <> True Then
rec.MoveLast
End If
rec.AddNew
rec("Name") = Name
rec("ESN") = ESN
rec("Email") = Email
rec("GlobalID") = ID
rec("CardNumber") = CardNumber
rec("Expiry") = ExpiryDate
rec("AuthNo") = AuthNo
if Approval_Status = "1" then
rec("Authorized") = TRUE
else
rec("Authorized") = FALSE
end if
rec("Date") = DateTime
rec.Update
rec.Close
' END OF DATABASE UPDATE
```

The next part of the callback URL creates the response which is sent to the terminal. This response is dependent upon the type of terminal.

This script generates two different responses, based upon whether the transaction was approved. This is represented by the variable, ’ Approval_Status.’

Pay attention to the two if/else statements, which determine which response is generated.

html sample

```
Response.Write "<!-- KIOSK_CALLBACK_RESULT "
Response.Write "approval_status=" & Approval_Status
Response.Write "&service_symbol=" & ServiceSymbol
Response.Write "&approval_msg="
```

```
if Approval_Status = "1" then
Response.Write "Your card was approved and you have been entered
into the contest"
else
Response.Write "Your card was not approved and you have not been
entered into the contest"
end if
if Approval_Status = "1" then
Response.Write "&receipt_format=RAW"
Response.Write "&receipt_info=%1B%54%01%1B%68%04%0E - - - - - -
SWIPE YOUR CARD TO WIN!! - - - - - - "
Response.Write "%0A%0A%0A%0A%1B%54%00%1B%68%01%0F%09%09  Thank
you for helping to test the NetVenue, " & Name & "."
Response.Write "%0A%0A%09%09You have been entered into the draw
and have a chance to win"
Response.Write  "%0A%09%09%09    one  of  four  Maestro  900  DSS
phones."
Response.Write "%0A%0A%1B%68%02%0E   Winners will be notified
September 30, 1999."
Response.Write  "%0A%0A%1B%68%01%0F%09%09%09Please  keep  this
coupon as proof of entry."
Response.Write "%0A%0A%0A%0A%1B%54%01%1B%68%03%0E - - - - - -
THANK YOU AND GOOD LUCK! - - - - - - "
Response.Write  "%0A%0A%1B%54%00%1B%0F%0F%09%09"  &  DateTime  &
"%09%09%09%09Auth no: " & AuthNo & ""
Response.Write "-->"
Response.Write "</head>"
Response.Write  "<body  bgcolor  =""#ffffff""  font  face  =""ver-
dana"">"
Response.Write  "<center><h1><b><font  color=""teal"">Thank  you
for entering the contest " & Name & "!"
Response.Write "<br>Good Luck!</font></b></h1><br>"
Response.Write "Your name and the information below have been
submitted into our database.<br><br>"
Response.Write "Phone #: " & ESN & "<br>"
Response.Write "Global ID: " & ID & "<br><br>"
Response.Write "<b><h1>The winners will be notified September
30, 1999.</h1></b><br><br>"
Response.Write "<b>Please keep the coupon that printed as proof
of entry into the draw.</b><br><br>"
Response.Write "<b>Note: If you answered ""Yes"" to the question
""Do you want a printed receipt?"",<br>"
Response.Write "the NetVenue would have printed a transaction
receipt and a coupon.<br>"
Response.Write "<u>The transaction receipt is for testing only</
u>, and there have been <u>no charges made to your credit card.</
u></b></center>"
```

```
else
Response.Write "-->"
Response.Write "</head>"
Response.Write  "<body  bgcolor=""#ffffff""  font  face=""ver-
dana"">"
Response.Write "<center><font color=""teal""><h2><br><br><br>
<br>We're sorry, but you have either entered an invalid card, or
your card was not approved.</h2></font><br></center>"
end if
%>
</BODY>
</HTML>
```

Glossary

The following terms and abbreviations appear in this document.

API

See application programming interface.

application programming interface (API)

A set of commands that an application program uses to request lower-level service performed by the terminal software. For the Web Terminal, API can be written into the HTML.

attachment

A file that is sent along with an E-mail.

attract loop video

A video that is repeatedly displayed by the Web Terminal to attract potential users, when the terminal is not being used.

card clearing

Making sure a card is valid, or that enough funds exist on the card. The Web Terminal can clear cards using an external source, or content providers can clear cards on their own.

card reader

A device that reads information off of magnetic stripe or chip cards, such as smart cards or credit cards.

Common Gateway Interface

(CGI) Astandard for running programs from a server, in which the program receives arguments as part of the HTTP request.

content server

A server which stores NetVenue content, usually containing HTML services.

credit card

Users can pay for services or products by charging them to a card, and then paying the credit card company at a later date.

database

(DB) A set of data which can be accessed by other programs. In the case of the NetVenue terminal, databases can exist either on the Terminal or on a server.

DB

See database.

Dynamic Hypertext Markup Language

(DHTML) A modification to the HTML standard which allows for more movement and interactivity on Web pages.

E-commerce

Electronic commerce. Financial transactions for a commercial business are performed through an electronic medium, such as the Internet.

E-mail

See electronic mail.

E-mail server

A server which stores the email accounts and E-mail messages for users.

electronic mail

(E-mail) A form of communication, developed as an alternative to the exchange of physical documents by traditional postal or courier services, that allows users to send messages by electronic means and to consult received messages at their convenience.

embedded Web browser

A software application for Web navigation that runs inside the terminal software.

encryption

A security function that allows information to be put into a code before it is sent. It is unlocked when it reaches its destination. The Web Terminal supports 128 bit encryption, which is the highest level of encryption currently available for Internet use.

Ethernet

A communications network standard that is for a10-Mbps (megabit per second) or 100-Mbps baseband local area network (LAN) bus using carrier sense multiple access with collision detection (CSMA/CD) as the access method. It establishes the physical characteristics of the network and the use of various cabling technologies, such as thick or thin coaxial cable, unshielded twisted pair cables, and fiber optic cables.

file transfer protocol

(FTP) A standard high-level protocol for transferring files from one computer to another, usually implemented as an application level program, and uses the Telnet and TCP protocols.

frame relay

A statistically multiplexed interface protocol for packet-switched data communications. Characteristics of frame relay include fast data transmission speeds, and neither flow control nor error correction.

front-end navigation menus

A series of user friendly screens that help the user find and access information and services on the Web Terminal.

glidepad

A pointing device that is activated when a user touches a small surface on the keyboard bezel. It activates an on-screen cursor.

hardware

Any part of a computer that physically exists. The monitor, hard-drive, and printer are all examples of hardware.

HTML

See hypertext markup language.

hypertext markup language

(HTML) An application of SGML (Standard Generalized Markup Language [ISO 8879]) implemented in conjunction with the World Wide Web to facilitate the electronic exchange and display of simple documents over the Internet.

integrated services digital network

(ISDN) A set of standards proposed by the CCITT to establish compatibility between the telephone network and various data terminals and devices. ISDN is a fully digital network, in general evolving from a telephone integrated digital network. It provides end-to-end connectivity to support a wide range of services, including circuit-switched voice, circuit-switched data, and packet-switched data over the same local facility.

Internet

A worldwide interconnection of individual networks operated by government, industry, academia, and private parties.The Internet originally served to interconnect laboratories engaged in government research, and has now been expanded to serve millions of users and a multitude of purposes.

Internet protocol address

(IP address) A unique 32-bit number for a specific TCP/IP host on the Internet. IP addresses are normally printed in dotted decimal form such as 128.127.50.224.

Intranet

A private network that uses Internet software and Internet standards. It is generally reserved for use by people who have been given the authority and passwords necessary to use that network.

IP address

See Internet protocol address.

ISDN

See integrated services digital network

ISDN router

Connects an ISDN line to an Ethernet line.

Java

An object oriented programming language and platform. Popular on the Internet, because it runs on any operating system where a Java Virtual Machine exists.

Java Applets

Small programs written in the Java programming language which typically run inside a Web browser that supports a Java Virtual Machine.

Java Virtual Machine

Software that can interpret Java byte codes and run Java programs.

loyalty card

A card provided by an institition or business. It can allow users to change transactions to the card, or give users discounts on transactions. One typical example would be the Air Miles card in Canada.

menu

A list of choices that a user can select from.

multimedia

The combination of multiple elements of communication, such as sound, text, graphics, and animation.

NetVenue Manager

A central database which stores information and records for the NetVenue system.

operating system

(OS) The software that interprets instructions given by applications software programs and causes a computer's or telephone system's components (CPU, peripherals) to perform actions as per those instructions.

OS

See operating system.

Pay-Per-Use

When users pay for access to a service based on long much they use it.

Pixel

Abbreviation for picture element. The smallest unit of area of a video screen image. A single point on a CRT display.

PPU

See Pay-Per-Use.

Pre-paid integrated circuit card

Also known as a smart card, telecard, or chip card. Users buy the cards and use them to pay for telephone calls by inserting them into the card reader. They contain an electrially erasable programmable read-only memory (EEPROM) chip that records the value of the card. This value decreases as the card is used.

QuickTime

A commonly used format for storing videos and other media such as graphics and audio.

RAM

See random access memory

random access memory

(RAM) Memory into which data can be written and from which data can be read. A solid state memory device used for transient memory stores. Information can be entered and retrieved from any storage position.

secure sockets layer

(SSL) An encryption process which provides encrypts communications on the Internet.

server

A network device that provides service to the network users by managing shared resources.

smart card

See pre-paid integrated circuit card

software

Programs, applications, and utilities that exist as binary code and can be run on an operating system.

TCP/IP

See transmission control protocol/Internet protocol

touch screen

A transparent screen which fits over the monitor. Users can touch the screen to activate on-screen buttons, in the same way that clicking with a mouse would activate them.

transmission control protocol/Internet protocol

(TCP/IP) A protocol stack, designed to connect different networks, on which the Internet is based. It was designed to withstand a partial deterioration of the network.

uniform resource locator

(URL) A standardized way of representing different documents, media and network services on the World Wide Web.

Uninterruptible power supply

(UPS) A device that is inserted between a primary power source, such as a commercial utility, and the primary power input of equipment to be protected, e.g., a computer system, for the purpose of eliminating the effects of transient anomalies or temporary outages.

URL

See uniform resource locator.

validation

See card validiation.

workstation

A computer that allows technicians and administrators to interact with a database, server, terminal, or other computer.

Index

NetVenue
Application developer Handbook

Issue: 01.01
Status: Standard
Date: September 1999

Printed in Canada