

KDT750

APL: AML Price Lookup Protocol



For models:
KDT750-XXXX



Revision History

September 2008 - Initial revision

November 2008 - Add POST documentation, reformat and restructure

KDT750 APL Guide

© 2009 American Microsystems, Ltd. All rights reserved.

American Microsystems, Ltd. reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult American Microsystems, Ltd. to determine whether any such changes have been made. The information in this publication does not represent a commitment on the part of American Microsystems, Ltd.

American Microsystems, Ltd. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

This document contains proprietary information which is protected by copyright.

All rights are reserved. No part of this document may be photocopied, reproduced or translated into another language without the prior written consent of American Microsystems, Ltd.

American Microsystems, Ltd.
2190 Regal Parkway • Euless, TX 76040
Phone 800.648.4452 • Fax 817.685.6232
www.amltd.com

Table of Contents

About This Document

Introduction

Chapter Descriptions

Chapter 1 – Getting Started

Transaction Overview

Protocol Overview

AML Price Lookup Protocol

APL via TCP

APL via HTTP

Chapter 2 – AML Price Lookup Protocol

Transaction Overview

General Data Format

Lookup Request from Client to Server

Client to Server Event Names

Example APL Client to Server Events

Lookup Reply from Server to Client

Output Pairs and Action Pairs

Special Output Pairs

Available Action Pairs

Formatting Text Output

Example Field Format Pair

Chapter 3 – Using APL Over HTTP

Overview

HTTP POST

POST Reply

URL Encoding Data

Example Transactions

Example Server Source Code

Chapter 4 – Using APL Over a TCP Socket

Overview

General Packet Format

Example Transactions

Example TCP Server Design

TCP Server Source Code

Chapter 5 – PCDemo

Introduction to PCDemo

PCDemo Operational Loop

Command Line Interface

Local Image Slide Show

Lookup Screen

Lookup Screen Background and Error Images

Text Fields

PRC Field Font

Colors

Example PCDemo Implementation

PCDemo on the KDT750

Appendix A – APL HTTP POST Transaction Examples with Screenshots

About this Document

Introduction

This document describes the software interface protocol and system requirements for use of the AML Price Lookup protocol (APL) and application software.

This document is provided as **PRELIMINARY ONLY** and is not intended as a complete product reference. AML reserves the right to make changes to this document and to any hardware, software or product it describes.

Chapter Descriptions

Chapter 1 – gives brief overview of APL and its intended uses

Chapter 2 – describes the AML Price Lookup (APL) protocol, which provides communication between AML kiosk devices and a database lookup server or some other network device

Chapter 3 – describes how to implement APL via HTTP POST

Chapter 4 – describes using APL over a raw TCP socket connection

Chapter 5 – describes the KDT750 APL client application PCDemo

1

Getting Started

This chapter gives a brief overview of APL and its intended uses.

Theory of Operation

APL was designed as a network based protocol for use in price verification and data lookup environments. It is tailored for use where some given input data will be sent to a network server followed by a meaningful response. Generally, this response will be parsed into text based fields and displayed on an AML price checking kiosk terminal such as the KDT750.

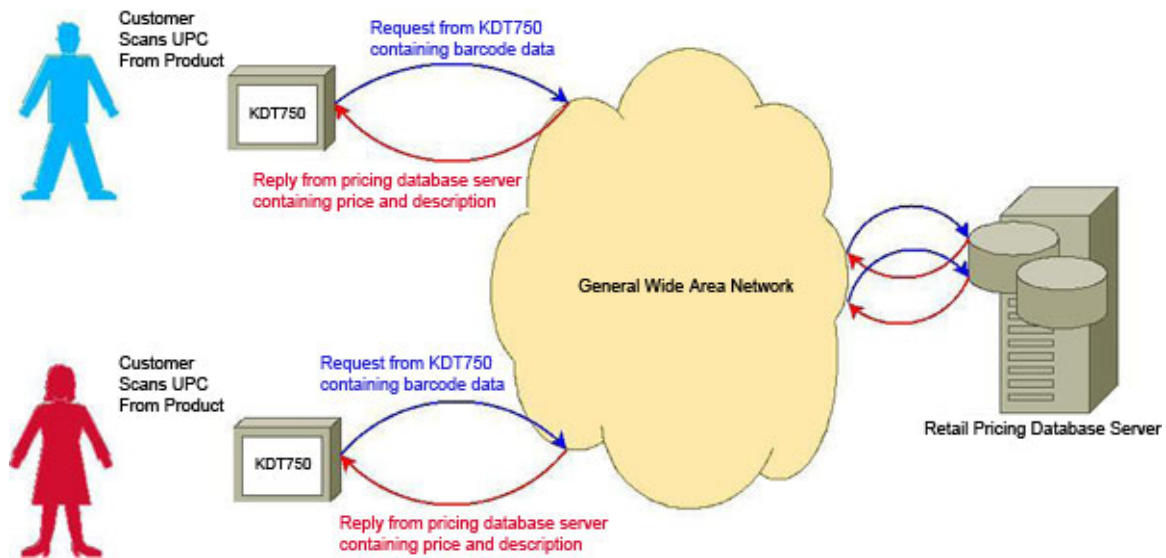


Figure 1.1 - High Level Price Checking Implementation

Communication Protocol Overview

Both the kiosk client software and the lookup server must understand a common interfacing protocol. To allow for easy integration into existing environments and to give the maximum functionality to AML price checking systems, APL supports two high level communication protocol methods – HTTP forms POST and raw TCP sockets. Each of these APL methods contain the ability to send and receive the same data; only the format of the request and reply differ.

APL: AML Price Lookup Protocol

APL was designed in an effort to simplify communication between price checking clients and database servers. Compared to other price verification protocols, APL drastically reduces the complexity of the server side software by limiting the client-server command set to only the necessary instructions. By simplifying the protocol down to only the bare necessities, integration time is drastically reduced.

APL via TCP

APL can utilize a standard raw TCP stream socket connection initiated by the client device (KDT750) to a database server. In this method, the client opens the socket to the server, sends the bar code data, receives a reply packet from the server and closes the connection. This reply is then parsed and the data is displayed on the LCD.

APL via TCP Features

- Fast communication with low overhead
 - Simple, human readable packet format
 - Extensible model for future custom enhancements
 - Easily parsed with server-side scripts
 - Fully functional server application source code provided
-

APL via HTTP

To allow for integration in environments where adding network services or opening new TCP ports is not an option, APL can utilize the standard Hypertext Transfer Protocol's (HTTP) POST operation. Here, APL acts as if it was a simple web client connecting to an HTTP service on the server.

APL via HTTP POST Features

- Easily integrates into existing web server environments
- Simple web based development for server-side software
- Easily parsed with server-side scripts
- Fully functional server application source code provided

2

APL

This chapter describes the AML Price Lookup (APL) protocol, which provides communication between AML kiosk devices and a database lookup server or some other network device.

Transaction Overview

APL is a packet based, connection oriented protocol. For each lookup to the server, a new connection is created, the request made, a reply sent and the connection destroyed.

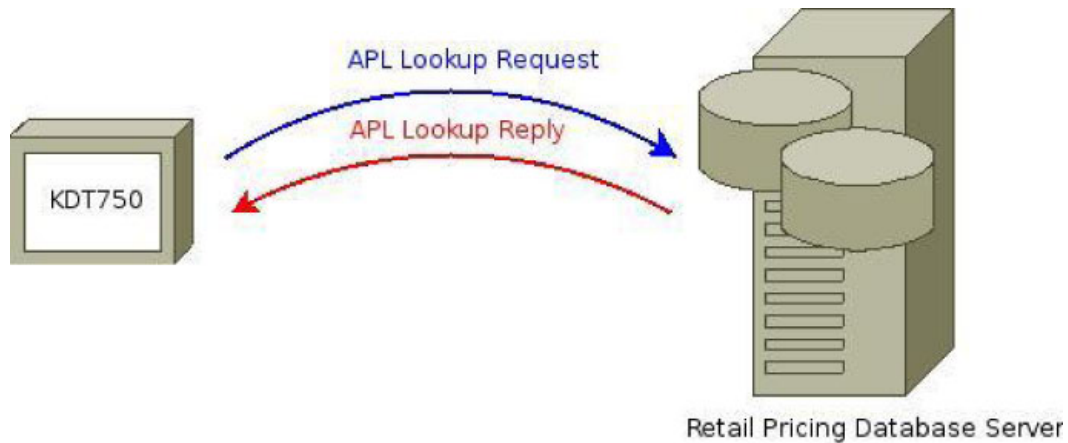


Figure 2.1 - Typical AP Transaction

General Data Format

APL defines all messages between two devices as item/value pairs. Item/value pairs are a collection of bytes defining a variable's name and its associated data, and are used to communicate a variety of different actions, commands, and information between the host and client.

Normally, the item name is separated from the value data by a standard ASCII equals ("=") sign. Data to the left of the equals sign is defined as the item's or action's name, whereas to the right of the equals sign is defined as the value. The 'ITEM=VALUE' pair is normally ASCII encoded, but depending upon the implementation, some characters may be required to be encoded in a different format.

Here, the data is named "BAR", and its value is "1234567890":

B	A	R	=	1	2	3	4	5	6	7	8	9	0
0x42	0x41	0x52	0x3D	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x30

Lookup Request from Client to Server

A general request from a client device will have a single item/value pair. The ITEM (left of equals sign) will denote the source of some event, and the VALUE (right of the equals sign) will give the server the event's data. An event could be a user scanning a bar code with an internal scanner, a swipe of a magnetic stripe card, a press of a pushbutton, or some other external or internal action.

In the example given above, a bar code reader, denoted by the "BAR" ITEM was used to scan a bar code that contained the data "1234567890".

Client to Server Event Names

APL defines certain events that server applications should be aware of:

Event	ITEM Data	VALUE Description
Internal bar code reader read a symbol	BAR	The bar code's data, as received from the bar code scanner.
External magnetic stripe reader read a card	MSR	The magnetic card's data, as received from the MSR. NOTE: This data may contain track separator characters.
External proximity or smart card reader read a card	PRX	The proximity or smart card's data, as received from the Weigand interface.
A front panel pushbutton was pressed	BUT	A single ASCII character defining which button was pressed (normally, buttons are expressed as capital letters "A" through "D").
Data was read from a keyboard	KBD	The data that was read from an external keyboard within a certain amount of time or until an ASCII carriage return was seen. The method of handling raw keyboard data is device dependent. Note that many USB handheld bar code readers will be treated as keyboards.
A digital input has changed state	INP	The input number denoting which digital input has changed state followed by a comma, followed by the new value. For example, if digital input number 1 changed state from digital low to digital high, the data would contain 1,1 . If the input then fell back to digital low, the data would contain 1,0 .

Example APL Client to Server Events

A user scans a bar code with data of "1234567890" with the internal bar code reader:

B	A	R	=	1	2	3	4	5	6	7	8	9	0
0x42	0x41	0x52	0x3D	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x30

A user swipes a magnetic stripe card with data of "John Doe" with the external MSR:

M	S	R	=	J	o	h	n		D	o	e
0x4D	0x53	0x52	0x3D	0x4A	0x6F	0x68	0x6E	0x20	0x44	0x6F	0x65

A user presses the left-most pushbutton on the front panel:

B	U	T	=	A
0x42	0x55	0x54	0x3D	0x41

**Note: From left to right, the buttons on the front of the KDT750 are conventionally called buttons A, B, C and D. Other devices may use different conventions.*

A door bell switch connected to a digital input 2 was pressed (input going high):

I	N	P	=	2	,	1
0x49	0x4E	0x50	0x3D	0x32	0x2C	0x31

Lookup Reply from Server to Client

In response to a lookup request, the server should reply with meaningful data. In the most basic of situations, the server will lookup the given data in a retail price database and reply with the price of the item to be drawn on the LCD.



Figure 2.2 - Example Lookup Reply Screen

A server reply, like a client request, uses 'ITEM=VALUE' elements, but unlike the request, a reply payload will typically contain multiple pairs. Each ITEM=VALUE pair will denote an action that the client device should take (i.e., Turning on a digital output) or a text based field that should be drawn on the LCD screen.

In the example lookup screen above, two output fields are present: one containing the price (12.99) and the other containing the description (Designer Flower Pot...). The text at the top of the screen is part of the background image and is not being actively drawn by APL.

Output Pairs and Action Pairs

Normally, in a lookup reply, the ITEM element corresponds to a text field on the screen and the VALUE element to the text that should be displayed. Each valid element pair received from the server will be drawn on the KDT750 LCD as a new text field. These output pairs can have any non-restricted ITEM name as long as it contains only alpha-numeric characters. This ITEM name is only used internally and must be unique.

A restricted ITEM name is one that is used for a special purpose action (turning on a digital

output). These action pairs will tell the client device to act on some hardware, and will not result in any data being drawn on the screen. A list of restricted names can be found in the *Available Action Pairs* section of this document.

Special Output Pairs

Two special output names exist: "PRC" and "DES". These output field names are predefined with certain enhancements to allow more flexibility in "price+description" environments.

Special Text Output Field	ITEM Name	Description
Price of an item	PRC	<p>The price of an item. This field will use images of the numerals in place of a text based font to allow much more customization with respect to size, shape, color and orientation.</p> <p>Note that the "\$" (or other monetary symbol) will be automatically prefixed to the price for display.</p> <p>The PRC field can only contain the characters "0" - "9" and ".".</p>
Description	DES	<p>A description of the item. When using APL in basic "Price+Description" environments, the DES field should be used. The DES field has a default text formatting designed for item descriptions.</p>

Available Action Pairs

Action pairs are 'ITEM=VALUE' pairs that denote special actions or commands that the client device should take. These element pairs will NOT be drawn on the LCD display in any way.

Action	ITEM Name	Description															
Enable/Disable a digital output	RELAY	<p>The value of the RELAY tag should be the desired output number to act on, followed by a comma, the mode in which to act, a second comma and finally a time in seconds.</p> <p>The digital outputs can be set to modes of ON, OFF, ON_WITH_TIMEOUT or OFF_WITH_TIMEOUT by using the following values:</p> <table border="1"> <thead> <tr> <th>RELAY Action</th> <th>Mode</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>OFF</td> <td>0</td> <td>Turn off the output (open relay)</td> </tr> <tr> <td>ON</td> <td>1</td> <td>Turn on the output (close relay)</td> </tr> <tr> <td>OFF_WITH_TIMEOUT</td> <td>2</td> <td>Turn off the output (open relay) for a given number of seconds, then turn back on (close relay)</td> </tr> <tr> <td>ON_WITH_TIMEOUT</td> <td>3</td> <td>Turn on the output (close relay) for a given number of seconds, then turn back off (open relay)</td> </tr> </tbody> </table> <p>For example, to turn on output 1 for five seconds:</p> <p style="text-align: center;">RELAY=1,3,5</p>	RELAY Action	Mode	Description	OFF	0	Turn off the output (open relay)	ON	1	Turn on the output (close relay)	OFF_WITH_TIMEOUT	2	Turn off the output (open relay) for a given number of seconds, then turn back on (close relay)	ON_WITH_TIMEOUT	3	Turn on the output (close relay) for a given number of seconds, then turn back off (open relay)
RELAY Action	Mode	Description															
OFF	0	Turn off the output (open relay)															
ON	1	Turn on the output (close relay)															
OFF_WITH_TIMEOUT	2	Turn off the output (open relay) for a given number of seconds, then turn back on (close relay)															
ON_WITH_TIMEOUT	3	Turn on the output (close relay) for a given number of seconds, then turn back off (open relay)															
Execute a system command		<p>The value of the COMMAND tag should be some arbitrary system command to be executed by the operating system.</p> <p>WARNING: Care must be taken when using the COMMAND tag. To allow for more functionality, the system command will be executed with administrative privileges on the client device, and therefore, it is possible to execute commands that delete, remove, and corrupt files in the nonvolatile file systems (SD, USB, etc...).</p>															

Formatting Text Output

To define a format for any given text output field, a special '@ITEM=VALUE' pair is used. The name element will be the '@' symbol (ASCII 0x40) followed by the field name. For example, the format name element for a 'PRC' field will be simply '@PRC'.

If no format '@ITEM=VALUE' pair exists in the packet, the client device will either ignore the 'ITEM=VALUE' pair or attempt to draw it using the following rules:

- 1.) If the PRC and DES fields exist, they will be drawn at their default locations with default attributes. All other fields will be ignored.)
- 2.) If no PRC or DES fields exist and no fields contain '@ITEM=VALUE' pairs, the device will draw the first two fields in the packet at the default PRC and DES locations and discard all other field definitions.

The VALUE element will be a comma separated list of the field format variables in readable ASCII text with all variables represented as decimal numbers:

@ITEM=<X>,<Y>,<W>,<H>,<FG>,<BG>

The available format variables are:

Variable	Location	Minimum	Maximum	Description
X Location	<X>	0	639	The X location of the upper left corner of the field. X=0 (zero) is defined as the left-hand side of the LCD.
Y Location	<Y>	0	479	The Y location of the upper left corner of the field. Y=0 (zero) is defined as the top of the LCD.
Width	<W>	1	640	The width, in pixels, of the field.
Height	<H>	1	480	The height, in pixels, of the field.
Text Color	<FG>	0	255	The text foreground color.
Field Color	<BG>	0	255	The field background color.

Example Field Format Pair

The DES field will be located at x=70, y=200, width=500, height=150, foreground color is 0 and background of 120:

@	D	E	S	=	7	0	,	2	0	0	,	5	0	0
0x40	0x44	0x45	0x53	0x3D	0x37	0x30	0x2C	0x32	0x30	0x30	0x2C	0x35	0x30	0x30
,	1	5	0	,	0	,	1	2	0					
0x2C	0x31	0x35	0x30	0x2C	0x30	0x2C	0x31	0x32	0x30					

3

Using APL Over HTTP

This chapter describes how to implement APL via HTTP POST.

Overview

HTTP (Hypertext Transfer Protocol) is the most common of all Internet protocols. Although it is used primarily to fetch pages from web servers to display in a browser, it is actually a general purpose transmission protocol. Since many large scale networks do not support opening new TCP sockets but already host web based portals, APL has been designed to use the HTTP POST method of sending and receiving data to and from a server.

HTTP POST

HTML forms already define a VALUE=ITEM style interface that works well when integrated with APL. Following the APL guidelines, the client application will create POST data with the appropriate request event name (BAR, INP, etc...) as a field name and the data value being the supplied data.

For example, if an internal bar code reader was used to scan a bar code containing the data "1234567890", APL would post data to the server in the form of an input field named "BAR" with a value of "1234567890". If this data was actually coming from an HTML form, it would have been formatted as such in HTML:

```
< input name="BAR" type="text" value='1234567890' />
```

POST Reply

The server side application or script should perform the desired action on the data and supply a meaningful response to the user. ITEM=VALUE pairs in the reply should be sent as plain text in the normal "ITEM=VALUE" format.

More often than not, this reply will contain multiple action or text output pairs.

When using HTTP POST, APL defines the separation between ITEM=VALUE pairs by a carriage return or new line.

It is also important that the reply contains a valid HTTP header.

Example server reply:

```
Content-type: text/html

PARTNUM=P/N: 123789
@PARTNUM=200,200,240,35,35,50
```

URL Encoding Data

APL will URL encode data being sent to the server-side script, and the server should URL encode all replies. That is, all data in a single ITEM=VALUE pair should be URL encoded, leaving the new line between ITEM=VALUE pairs intact as ASCII.

URL encoding is necessary to allow passing special characters that are normally not allowed in URL data. These characters include: **space \$ & < > ? ; # : = , " ' ~ + %**

To URL encode a special character, replace the character with a '%' followed by the two ASCII characters that represent the hex encoded value of the character to URL encode. For example, to URL encode an ampersand, the **&** would be replaced with **%26**.

URL encoded carriage return characters within a VALUE of an output field will be translated into a new line when displayed.

Example Transactions

A bar code containing the data "1641115" was scanned with the internal bar code reader. The server responds with a price field and description field.

Note that all data strings are URL encoded. See Appendix A for more examples.

Request:

```
BAR=164%2d1115
```

Reply:

```
Content-type: text/html
```

```
PRC=0.187
```

```
DES=Screw.%20Torx%2C%20Plastic%20Thrd%2C%200-4
```

Example Server Source Code

```
import os
import cgi
import urllib

form = cgi.FieldStorage()

inputdata = ""

### Note that urllib.quote method is used to URL encode all outbound data

# Handle button presses (just reply with the button name)

if form.has_key('BUT') :
    button = form["BUT"]
    print "Content-type: text/html\n\n" # Header is important
    print "FIELD1=" + urllib.quote ( "You pressed button " + button.value )
    print "@FIELD1=" + urllib.quote ( "100,165,440,35,200,220" )
# Handle MSR read
elif form.has_key('MSR') :
    fld = form["MSR"]
    inputdata = fld.value
    print "Content-type: text/html\n\n"
    print "MAGDATA=" + urllib.quote( inputdata )
    print "@MAGDATA=" + urllib.quote ( "20,250,600,45,0,50" )
# Handle internal bar code read
elif form.has_key('BAR') :
    fld = form["BAR"]
    inputdata = fld.value
    print "Content-type: text/html\n\n"
    # Handle the database lookup
    price = do_database( "SELECT Cost FROM 'Inventory' WHERE PN='" + inputdata )
    des = do_database( "SELECT Desc FROM 'Inventory' WHERE PN='" + inputdata )
    if price != "" :
        # URL encode the data and dump it out
        print "PRC=" + urllib.quote ( price )
        print "DES=" + urllib.quote ( des )
        print "@DES=" + urllib.quote ( "45,150,500,35,255,220" )
        print "PN=" + urllib.quote ( "P/N: " + inputdata )
        print "@PN=" + urllib.quote ( "200,200,240,35,35,50" )
        print "RELAY=" + urllib.quote ( "1,3,2" )
    else :
        # Display an error to the user
        print "FIELD1=" + urllib.quote ( " I'm sorry, I couldn't" + '\n' +
        " find that item." + '\n' + '\n' + "Would you like assistance?" );
        print "@FIELD1=" + urllib.quote ( "125,145,510,250,219,255" )
        print "FIELD2=" + urllib.quote ( "Press for assistance" + '\n' + "v" )
        print "@FIELD2=" + urllib.quote ( "15,370,340,75,255,220" )
else :
    # No valid ITEM names, send blank reply to show an error
    print "Content-type: text/html\n\n"
```

4

Using APL Over a TCP Socket

This chapter describes using APL over a raw TCP socket connection.

Packet Format

To use APL over a TCP socket, the following general packet format is used:

NOTE: All wireless SECURITY settings are configured in the wifi.conf file described in Chapter 5.



Field	Data	Description
<STX>	0x02	ASCII Start-of-Text character
PAYLOAD		The transmission's APL data payload
<ETX>	0x03	ASCII End-of-Text

General Payload Format

The payload of an APL TCP packet consists of 'ITEM=VALUE' pairs in readable ASCII format. Multiple pairs can be in any given payload with each pair being separated by an ASCII 0x00 (NULL) character. For example, in a server response where the ITEM element corresponds to a text field name and the VALUE element gives the text to display, a payload containing two fields, one named "PRC" with the data "9.99" and one named "DES" with the data "Flower Pot", would look like the following:

P	R	C	=	9	.	9	9	<NULL>	D	E	S	=	F	I
0x50	0x52	0x43	0x3D	0x39	0x2E	0x39	0x39	0x00	0x44	0x45	0x53	0x3D	0x46	0x6C
o	w	e	r		P	o	t							
0x6F	0x77	0x65	0x72	0x20	0x50	0x6F	0x74							

Note that this example is of a payload only and does not contain the packet start and stop characters.

Lookup Request Example Packets

A user scans a bar code with data of "1234567890" with the internal bar code reader:

<STX>	B	A	R	=	1	2	3	4	5	6	7	8	9	0	<ETX>
0x02	0x42	0x41	0x52	0x3D	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x30	0x03

A user swipes a magnetic stripe card with data of "John Doe" with the external MSR:

<STX>	M	S	R	=	J	o	h	n		D	o	e	<ETX>
0x02	0x4D	0x53	0x52	0x3D	0x4A	0x6F	0x68	0x6E	0x20	0x44	0x6F	0x65	0x03

A user presses the left-most pushbutton on the front panel:

<STX>	B	U	T	=	A	<ETX>
0x02	0x42	0x55	0x54	0x3D	0x41	0x03

Note: From left to right, the buttons on the front of the KDT750 are conventionally called buttons A, B, C and D.

Lookup Reply Example Packets

The server responds to a bar code read with an item price of "1.99" and a description of "Tomato Soup" on the first line and "8 oz." on the second.

<STX>	P	R	C	=	1	.	9	9	<NULL>	D	E	S	=	T
0x02	0x50	0x52	0x43	0x3D	0x31	0x2E	0x39	0x39	0x00	0x44	0x45	0x53	0x3D	0x54
o	m	a	t	o		s	o	u	p	<CR>	8		o	z
0x6F	0x6D	0x61	0x74	0x6F	0x20	0x53	0x6F	0x75	0x70	0x0D	0x38	0x20	0x6F	0x7A
.	<ETX>													
0x2C	0x03													

KDT750 APL Guide

The server responds to a magnetic stripe read with a screen containing three fields: an employee's name, their employee number and their clock-in status.

<STX>	N	a	m	e	=	J	a	n	e		D	o	e	<NULL>
0x02	0x4E	0x61	0x6D	0x65	0x3D	0x4A	0x61	0x6E	0x65	0x20	0x44	0x6F	0x6D	0x00
N	u	m	=	1	8	2	<NULL>	C	I	k	=	I	N	<ETX>
0x4E	0x75	0x6D	0x3D	0x31	0x38	0x32	0x00	0x43	0x6C	0x6B	0x3D	0x49	0x4E	0x03

Example APL Server Design

As a general case, the host server application can be realized very simply:

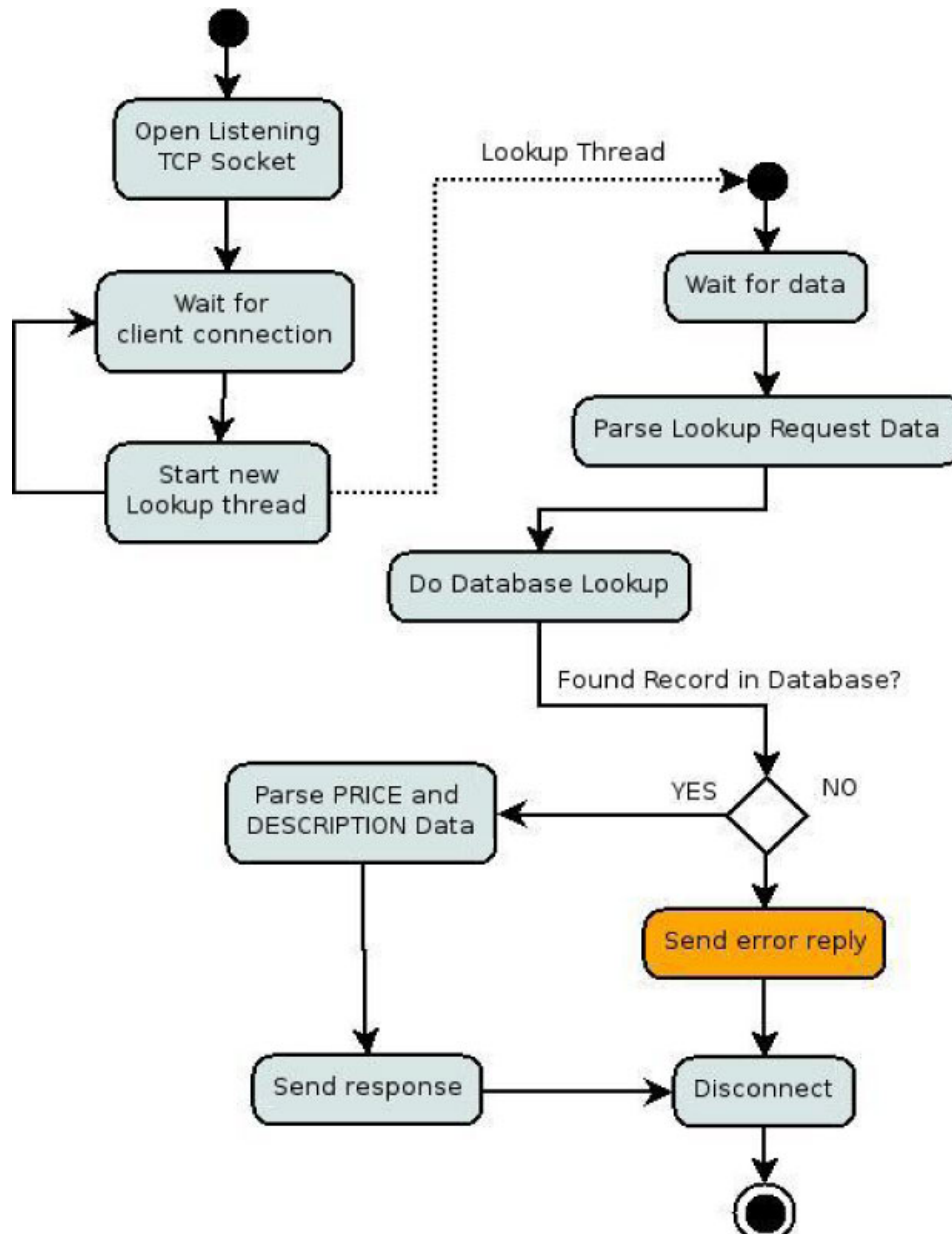


Figure 4.1 - Sample Server Flowchart

Server-side Source Code

```
# Use threads and sockets
import socket
import thread

# Lookup thread function to be called when a client connects
def lookup_thread ( client_socket, address ):
    # Read the request up to 512 bytes
    data = client_socket.recv ( 512 )
    # We don't care about which input device it came from for this example
    data = data.lstrip( '\002' ) # Remove <STX>
    data = data.rstrip( '\003' ) # Remove <ETX>
    device, barcode = data.split ( '=' ) # Split the parts
    # Do database work
    prc = do_database ( "SELECT Price FROM MyDatabase WHERE Barcode IS " barcode )
    des = do_database ( "SELECT Description FROM MyDatabase WHERE Barcode IS "
                        barcode )

    # Send "Unknown Item" on no record found
    if prc == ''
        reply = '\002' "FIELD1=Unknown Item" '\000'
                "@FIELD1=125,145,510,250,219,255" '\003'
    else :
        reply = '\002' "PRC=" prc '\000' "DES=" des '\003'
    # Send the reply and disconnect the client
    client_socket.send ( reply )
    client_socket.close ( )
    return

# Create the INET streaming socket
server_socket = socket.socket ( socket.AF_INET, socket.SOCK_STREAM )
# Bind to an unused TCP port - say 5000...
server_socket.bind ( ( "", 5000 ) )
# Allow 5 concurrent connections at a time
server_socket.listen ( 5 )

# Loop forever, handing new clients off to their own thread
while 1:
    client_socket, address = server_socket.accept ( )
    thread.start_new_thread ( lookup_thread, ( client_socket, address ) )
```

5

PCDemo - APL Client Application

This chapter describes the KDT750 APL client application PCDemo.

Overview

PCDemo is a versatile client application designed to run natively on the AML KDT750 price checking unit. PCDemo handles data capture and processing, network communication with a database server, and all user interface and display attributes for data lookup and verification applications.

The PCDemo software runs atop the KDT750's Linux® operating system based on kernel version 2.6.22 along with other associated software.

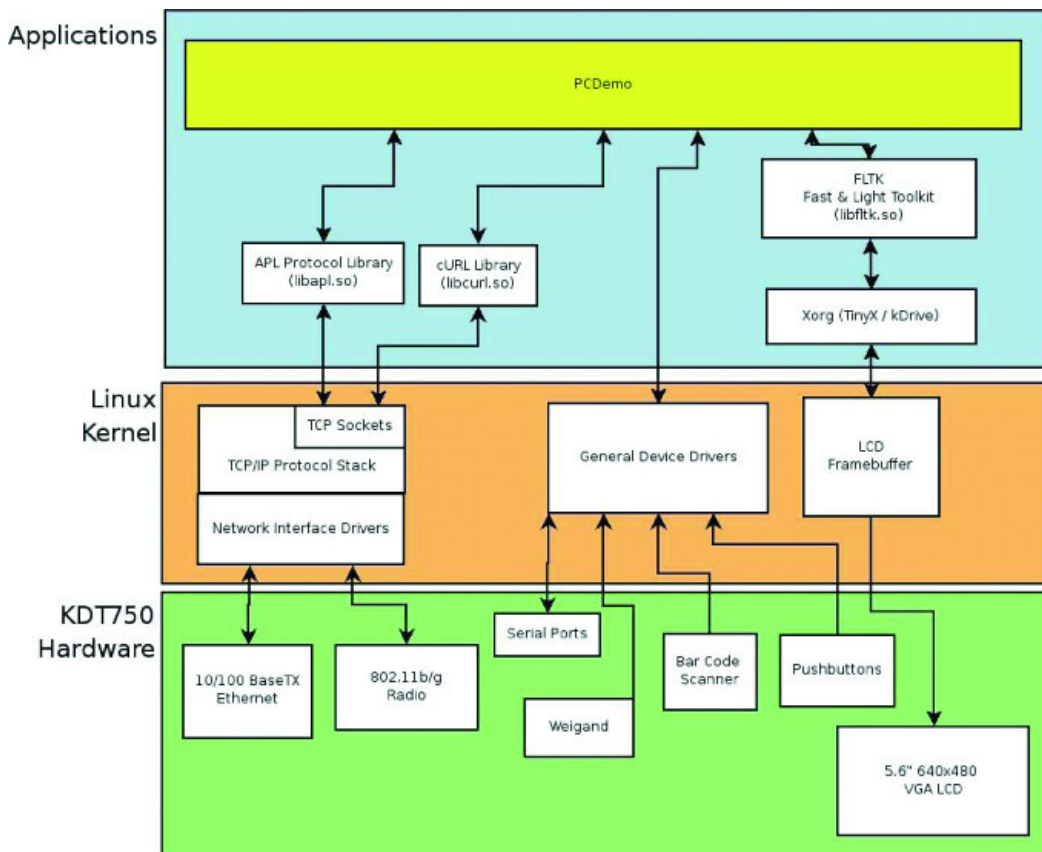


Figure 5.1 - PCDemo Associated Software

In most environments, the PCDemo software is all that is necessary to implement price checking, stock verification, employee time and attendance and customer self-services with the KDT750.

PCDemo Operational Loop

PCDemo's internal software structure is event based around clock timeouts and data input events.

KDT750 APL Guide

When the device is idle (i.e., No input device is being utilized), PCDemo displays a slide show of images. In the most general case, when input is received from any input device (internal bar code scanner, magnetic stripe reader, pushbuttons, etc...), the data lookup will be performed with the host server and a lookup screen will be shown containing the server response. If the server responds with a lookup failure, an error screen can be displayed. After a given amount of time, PCDemo will return to the slide show loop. This functionality can be extended through server side scripting and modifications to the runtime application and source code of PCDemo.

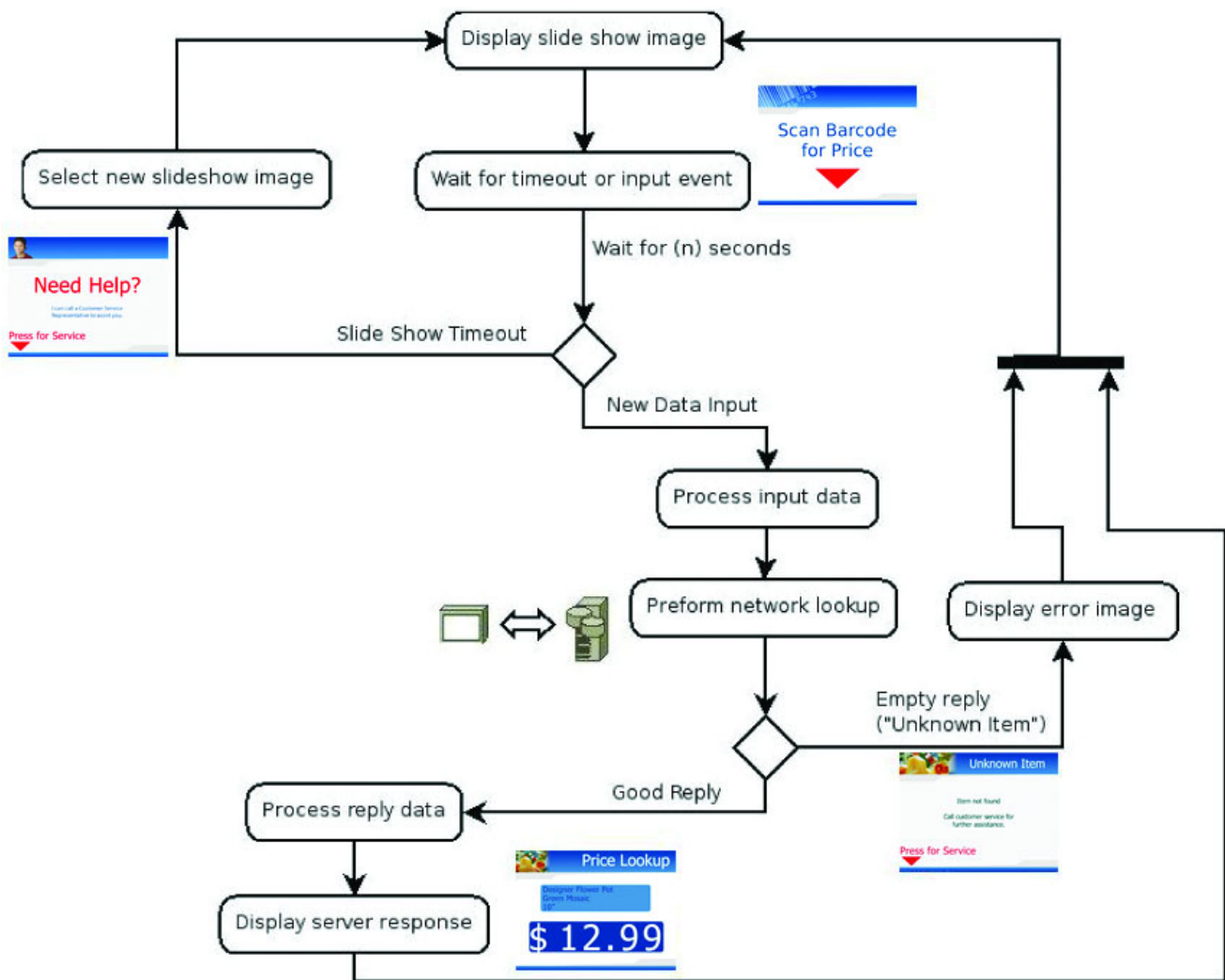


Figure 5.2 - PCDemo Basic Loop

Command Line Interface

PCDemo is normally located at `/usr/bin/pcdemo` of the KDT750 Linux file system. Note that the application binary name is all lowercase, and Linux has a case sensitive operating environment.

The following is a list and explanation of the command line flags for PCDemo.

Flag	Use	Description	Example
-H <Address>	Host Server Address	IP Address of lookup server	-H 192.168.100.1
-P <Port>	Host Server Port	TCP Port to connect to for lookups	-P 23
-m	Mode POST	Use HTTP POST	-m
-o <Timeout>	Network Timeout	Network connection and host reply timeout in seconds	-o 3
-u <Image Path>	Unknown item Background	Background JPEG image to use for lookup errors	-u /mnt/sd/error.jpg
-l <Image Path>	Item Lookup Background	Background JPEG image to use for normal lookups	-l /mnt/sd/lookup.jpg
-n <Directory>	Numeral Image Directory	Directory that stores the PRC field PNG numeral images. Defaults to SD card "numerals" directory (/mnt/sd/numerals)	-n /mnt/sd/numerals
-t <Timeout>	Slide show timeout	Time to display each slide show image. Defaults to 5	-t 5
-s <Timeout>	Lookup timeout	Time to display lookup screens in seconds. Defaults to 5	-s 5
-i <Directory>	Slide show image directory	Directory that contains only JPEG images to be displayed in the slide show. Defaults to SD card "slide show" directory (/mnt/sd/slideshow)	-i /mnt/sd/slideshow

Flag	Use	Description	Example
-p (012hb)	Input Device List	<p>Devices to use for server lookups</p> <p>0: Internal Bar code reader (/dev/ttySAC0) 1: User RS-232 Port (/dev/ttySAC1) 2: MSR Serial Port (/dev/ttySAC2) h: Weigand (HID) port b: Pushbuttons</p> <p>NOTE: PCDemo requires that all ports be configured and ready. For example, if a magnetic stripe reader is connected to serial port 2, then it must be configured at the correct baud rate, parity, and bit setup before executing PCDemo.</p>	-p 02b

< > - Denotes mandatory flag argument

() - Denotes optional flag argument

Local Image Slide Show

PCDemo will display a slide show of images when idle. These images will normally reside on the user supplied internal SD card and will be loaded into memory for fast display during operation. PCDemo will only load the first eight (8) images listed in the slide show image directory. The images are arbitrary, but must meet the following stipulations:

- Must be a standard JPEG (JPEG compression is optional)
- Must be no larger than 640x480 pixels in size (nominal is 640x480)
- All slide show images must be located in the same directory with no other files
- The directory that the images reside in must be passed properly to PCDemo via the command line interface (defaults to using a "slide show" directory of the internal SD card).

Lookup Screen Basics

When displaying a server response to a lookup request, the lookup screen of PCDemo will contain a basic background image along with any number of customizable text based fields.



Figure 5.3 - Lookup Screen

Lookup Screen Background and Error Images

The background image of the standard lookup screen and image to display on lookup error can be customized with standard 640x480 pixel JPEGs following the same stipulations as the slide show images. To change these images, the runtime command line arguments should be changed to point to the desired image.

Text Fields

The basic element of any lookup screen is the text field. Each text field has an arbitrary name that is only used internally and never displayed. This name is used as a reference for formatting the position, color and size of the field.

Field Name Requirements

- *Must contain ONLY alphanumeric characters*
- *Must be unique*

Special Text Fields

Two special field names exist: the PRC field and the DES field. These fields are predefined with certain enhancements to allow more flexibility in “price+description” environments.

Special Field	Name	Description
Price of an item	PRC	<p>The price of an item. This field will use images of the numerals in place of a text based font to allow much more customization with respect to size, shape, color and orientation.</p> <p>Note that the “\$” (or other monetary symbol) will be automatically prefixed to the price for display.</p> <p>The PRC field can only contain the characters “0” - “9” and “.”. The look of the numbers can be changed by adding a custom set of numeral images.</p>
Description	DES	<p>A description of the item. This field uses special text handling to ensure proper alignment of the text, as well as correct carriage returns and line spacing. The DES field will always be drawn with the largest possible text font.</p>

PRC Field Fonts

The PRC field uses PNG format graphics to display the numbers, decimal point and currency mark. Portable Network Graphics (PNG) format is used because it supports transparency, allowing for customization and integration within the background.

Each character available in the PRC field (“0” - “9”, “.” and a currency mark) is contained in its own image located in the directory pointed to by the Numeral Image Directory command line argument. The names of the files are simply “1.png”, “2.png”, etc... for the numerals and “dollar.png” for the currency mark. The decimal point should be named “point.png”, and a blank image that should be used for spaces should be called “blank.png”. The default images are shown below.

Default PRC Numeral Images

1.png	1	2.png	2
3.png	3	4.png	4
5.png	5	6.png	6
7.png	7	8.png	8
9.png	9	0.png	0
dollar.png	\$	point.png	.
blank.png			

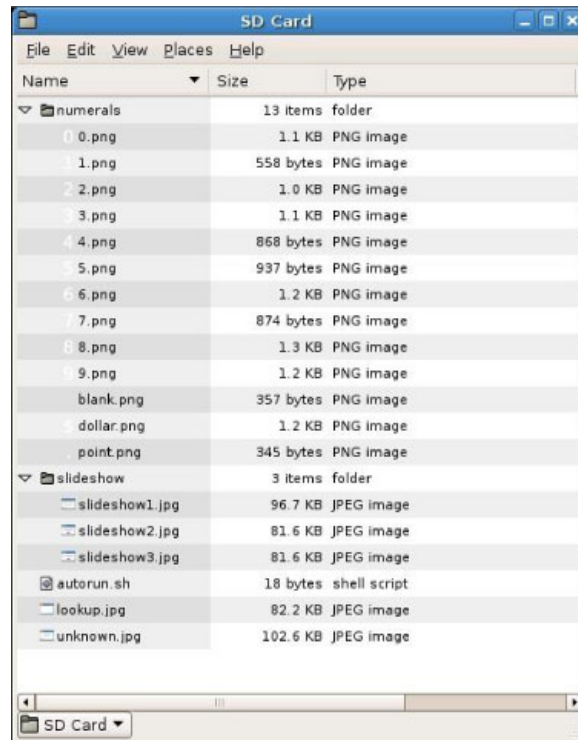


Figure 5.5 - Example SD Card Contents

With this SD card installed into a KDT750 unit, the `autorun.sh` script will be executed upon boot (see the KDT750 User's Guide for more information). This script contains the following PCDemo execution command:

```
/usr/bin/pcdemo -H 192.168.100.1 -P 5000 -u /mnt/sd/error.jpg -l /mnt/sd/lookup.jpg -n /mnt/sd/numerals -i /mnt/sd/slideshow -p 02b
```

Appendix A

Packet Examples with Screenshots

Example 1 - Price Lookup via HTTP Post



Client Lookup POST:

```
BAR=1641-115
```

Server Reply:

```
Content-type: text/html
```

```
PRC=0.187
```

```
DES=Screw. Torx, Plastic Thrd 04
```

```
@DES=45,150,500,35,255,220
```

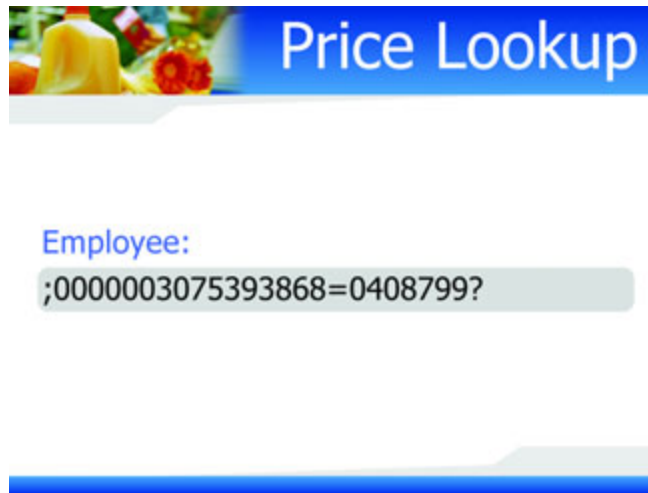
```
PN=P/N: 1641115
```

```
@PN=200,200,240,35,35,50
```

```
RELAY=1,3,2
```

NOTE: To enhance readability, these APL messages are not URL encoded

Example 2 – MSR Read via HTTP POST



Client Lookup POST:

```
MSR=;0000003075393868=0408799?
```

Server Reply:

```
Content-type: text/html
```

```
FIELD1=Employee:  
FIELD2= ;0000003075393868=0408799?  
@FIELD1=20,200,250,150,219,255  
@FIELD2=20,250,600,45,0,50
```

NOTE: To enhance readability, these APL messages are not URL encoded