Retroswitch LLC

Flyer User's Guide

Version 1.2 (July 26, 2019)

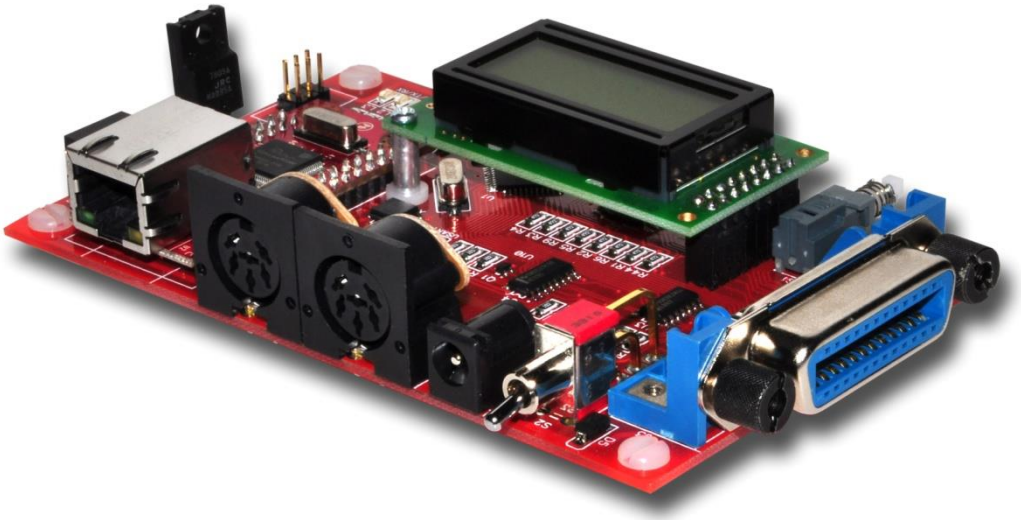

Table of Contents

Chapter 1:
Getting Started

The following topics will be discussed in this chapter:

- ✓ Description of the Flyer and its capabilities.
- ✓ Setting up your Flyer.
- ✓ How to access software included with your Flyer.

Overview

The Flyer is an internet modem and disk drive emulator compatible with most Commodore 8-bit computers having either an IEC or IEEE-488 hardware interface.

It is actually two peripherals in one. The control unit responds to device # 7 and is responsible for network communication, disk management, and overall device configuration. The disk unit responds to device #8-15 (selectable), and is a hybrid disk drive emulator which supports most standard Commodore disk drive commands and multiple disk formats.

The Flyer includes 4MB of flash memory, of which 3.5MB is available for caching disk images.

Setup

Connect the Flyer to your Commodore computer of choice, either via the IEC or the IEEE-488 connectors. Please make sure your computer and any connected peripherals are turned off before connecting! Also note that having both the IEC and IEEE-488 buses connected at the same time may give unpredictable results and is not recommended.

If you are using your own power adapter for Flyer, it is recommended you use a 9VDC 1A power source with 2.1mm connector, **center positive**. Using the wrong polarity adaptor may damage the Flyer, and more importantly, may damage your Commodore computer and any other connected peripherals!

Lastly, connect the Flyer to your LAN using a standard ethernet cable. Although the Flyer does support DHCP, it is disabled by default. The Flyer's IP address will be configured as

192.168.1.10 when you first receive it. Network settings can be changed by using the included configuration utility (described in the next section), or by issuing configuration commands to the control unit's command channel (described in chapter 2).

Power up the Flyer (and any other connected peripherals), then your Commodore computer. The disk unit is set to initially respond to device #10. To change this, change to the *device selection mode* on the Flyer using long presses of the button on the front left of the unit. Once the "Device #" screen appears on the LCD display, quick "taps" of the button will change the displayed value. Once the desired value is selected, no further steps are necessary - the changes are immediate.

Included Software

Although no software is required to use the Flyer, software is included to help you configure your Flyer as well as manage locally stored disks. The included software will be updated along with regular firmware updates.

The software is made available via a secondary disk drive emulated by the Flyer. This secondary disk is accessed using the standard Commodore syntax for multiple drive units.

For example, if your disk unit is set as device #10, a directory of the included software may be loaded by issuing the following BASIC command:

```
load"$1",10
```

The configuration program allows you to edit network settings (including enabling/disabling DHCP), cloud server configurations and aliases. It can be loaded as follows:

```
load"1:config",10
```

The disk manager allows you to view disks currently stored on the Flyer, create new blank disks, format disks, and more. It can be loaded as follows:

```
load"1:diskmgr",10
```

Note that the secondary software drive is not a full disk drive emulation, and does not support standard disk drive commands for initialization, copying, etc., nor does it support direct access or even command channel operations.

Also note that both programs were designed to run on all Commodore computers, from the PET to the 128.

A Note About Case

Since the Flyer is meant to interface with "the outside world", character case becomes an issue. Most Commodore users are used to working in upper case (at least with the VIC-20 and onward), only rarely switching to lower case for certain programs.

There are 2 character modes on Commodore computers: graphics mode and text mode. Graphics mode is what most Commodore users are used to. In this mode, unshifted characters appear as uppercase, and shifted characters appear as graphics symbols.

When changing to text mode (pressing the C= and shift keys simultaneously, or **`poke 59468,14`** on the PET), unshifted characters appear as lower case, and shifted characters appear upper case.

This is what is assumed when working with the Flyer: regardless of which mode you are in, unshifted characters will be interpreted as lower case *always*, and shifted characters will be interpreted as upper case *always*. Again, it doesn't matter which mode you are in, although it will be most natural if you are in text (lowercase) mode when working with the Flyer.

Again, to toggle between lower/upper case on most Commodore machines, simply press the Commodore and shift keys simultaneously.

To switch to graphics (uppercase) mode on a PET: **`poke 59468,12`**

To switch to text (lowercase) mode on a PET: **`poke 59468,14`**

All examples in this guide assume you are working in text (lowercase) mode.

Commands (such as **`load"$",10`**) are always displayed in lowercase/unshifted for this reason.

Chapter 2:
The Control Unit

The following topics will be discussed in this chapter:

- ✓ Overview and purpose of the control device.
- ✓ Commands and protocols supported by the control device.
- ✓ How to perform common tasks such as firmware updates.

Overview

The control unit provides all the functionality which is unique to the Flyer, including network communication, high level disk management, firmware updating, and more.

The control unit is exposed as a complete peripheral that responds to device #7. This address cannot currently be changed.

The control unit is also responsible for interfacing to *cloud servers* (HTTP servers which support the Flyer's specific queries for exchanging disks and files).

Commands may be issued to the control unit via the dedicated command channel (#15) with the remaining channels (2-14) available for general communication, very similar to standard Commodore disk drives. Channels 0 and 1 are reserved by the Commodore kernel for LOAD and SAVE.

Here are some examples:

```
load"http:somesite.com/programs/demo.prg",7
```

This loads the program "demo.prg" directly from the specified http server.

```
save"hello",7
```

Assuming a valid cloud server is selected, this will seamlessly save the current program in memory to the cloud server. If an invalid response or other error is encountered, it will be

displayed on the Flyer's LCD screen.

```
open2,7,2,"tcp:someserver.com,1234":print#2,"hello":close2
```

This will open a TCP/IP connection to the specified server and port, send the data "hello" and close the connection. Obviously no error handling is being performed.

```
open7,7,15:input#7,a,b$,c,d:print a,b$,c,d:close7
```

This will read the current error status from the control unit's command channel. This is identical to a standard Commodore disk drive.

Firmware Updating

Firmware updates are performed using the standard LOAD command. The default firmware server is retroswitch.com, and although this may be changed via the included configuration program, there is no reason to do so currently.

A directory listing of all available firmware images may be loaded and displayed by issuing the following commands. Note that for all of these examples, '^' indicates the up arrow.

```
load "$^",7
list
```

To see a verbose firmware listing (includes release notes for each version), the ',v' option may be included after the file pattern. The previous example didn't use a file pattern, so we'll just use '*' in this example, which will match all the files:

```
load "$^:*,v",7
list
```

In order to perform the actual firmware update, simply LOAD with the firmware '^' protocol specifier and a valid file specification. '*' will always load the latest firmware update. After the LOAD operation completes, the Flyer will reboot and apply the firmware update. In the meantime, a list of release notes may be seen for the update just received by typing LIST.

Here are a couple of examples:

```
load "^:*",7
```

This will download the latest firmware release, reboot the Flyer and apply the update.

```
load "^:1.0.4",7
```

This will download a specific firmware release (1.0.4), reboot the Flyer and apply the update.

IMPORTANT: If you are ever "downgrading" firmware, there may be certain cases where the data on the Flyer is incompatible since it was written with a newer version of the firmware. In this case, follow the full recovery procedure described in the next section which will erase all data on the Flyer and reset it to factory conditions. Then upgrade to the firmware version of your choice.

Recovery

In the rare case anything goes wrong during a firmware update (device is switched off, etc), the Flyer should automatically detect this and reinstall the factory firmware (the original version shipped on the unit) the next time it is powered up.

Factory firmware may also be restored manually by switching on the Flyer with the button held down.

If held for 5 seconds, “RESTORE FW ONLY” will appear on the display. If the button is released at this point, the original firmware will be rewritten to the device and verified. At this point the Flyer will restart and may be used normally, or upgraded to a newer version of firmware.

If held for 10 seconds, “RESTORE FW+CLEAR” will appear on the display. Releasing the button at this point will ERASE ALL DATA on the Flyer before restoring the original firmware on the device. This includes network and login settings as well. After this, the device will be in the exact state it was when you first received it.

The pending recovery procedure can be aborted by switching off the Flyer BEFORE letting go of the button.

Configuration

Configuration is performed by issuing the CONFIG: command over the command channel, followed by the particular configuration setting to change or retrieve.

Note that all of these settings may also be configured by using the included CONFIG program. See Included Software in Chapter 1 for instructions on how to access the bundled software.

Syntax for **retrieving** parameter:

```
open 7,7,15
print#7, “config:setting”
input#7, a, b$, c, d
close 7
```

The setting will be returned in the message portion of the command channel status (B$ in this example) unless otherwise noted. The returned value is only considered valid if the error code (A in this example) is zero. Otherwise, the message will contain a description of the error.

Syntax for **setting** parameter:

```
open 7,7,15
print#7, "config:setting=value"
close 7
```

Although omitted from this example, the status channel should normally be read to determine if the setting was updated properly.

Network Configuration Settings

The following setting names may be abbreviated to the first two letters. The examples will reflect this.

Setting	Description	Example
IP	IP Address	`print#7, "config:ip=192.168.1.10"`
SUBNETMASK	Subnet Mask	`print#7, "config:su=255.255.255.0"`
GATEWAY	Gateway Address	`print#7, "config:ga=192.168.1.1"`
DNS	DNS Server	`print#7, "config:dn=8.8.8.8"`
FIRMWARESERVER	Firmware Server[1]	`print#7, "config:fi=retroswitch%2ecom"`
DHCP	Enable/disable DHCP[2]	`print#7, "config:dh=1"`

[1] Values for these settings are URL encoded.

[2] 1 = enabled, 0 = disabled.

Alias Configuration Settings

Setting	Description	Example
A0 – A3	Alias Values[1]	`print#7, "config:a0=http%3aserver%2ecom%2f"`

[1] Values for these settings are URL encoded.

Cloud Server Configuration Settings

Setting	Description	Example
CLDNAM0 – CLDNAM3	Cloud Server 0 – 3 Name[1]	`print#7, "config:cldnam0=Xyzzy"`
CLDSRV0 – CLDSRV3	Cloud Server 0 – 3 Address[1]	`print#7, "config:cldsrv0=somesite%2ecom"`
CLDPRT0 – CLDPRT3	Cloud Server 0 – 3 Port Number	`print#7, "config:cldprt0=2112"`
CLDUSR0 – CLDUSR3	Cloud Server 0 – 3 Username[1]	`print#7, "config:cldusr0=geddy"`
CLDPWD0 – CLDPWD3	Cloud Server 0 – 3 Password[1]	`print#7, "config:cldpwd0=baseball"`

[1] Values for these settings are URL encoded.

Disk Emulator Configuration Settings

Setting	Description	Example
DI	Drive ID (Address)	`print#7, "config:di=9"`
JIFFYDOS	Enable/disable JiffyDOS[1]	`print#7, "config:jiffydos=0"`

[1] 1 = enabled, 0 = disabled.

Status

Status queries are performed by issuing the STATUS: command over the command channel, followed by the particular status identifier to retrieve. STATUS is similar to CONFIG, except it is used to retrieve information only. More specifically, it returns the current "live" state of certain properties. For example, if DHCP is enabled, the device's assigned IP address will likely differ from the IP address previously CONFIGured. STATUS could be used to return the IP address currently active, whereas CONFIG would still return the previously configured IP address.

Syntax for retrieving status information:

```
open 7,7,15
print#7, "status:identifier"
input#7, a, b$, c, d
close 7
```

The status value will be returned in the message portion of the command channel status (B$ in this example) unless otherwise noted. The returned value is only considered valid if the error code (A in this example) is zero. Otherwise, the message will contain a description of the error.

Network Status Identifiers

The following identifiers may be abbreviated to the first two letters. The examples will reflect this.

Setting	Description	Example
IP	IP Address[1]	`print#7, "status:ip"`
SUBNETMASK	Subnet Mask[1]	`print#7, "status:su"`
GATEWAY	Gateway Address[1]	`print#7, "status:ga"`

[1] The returned IP addresses reflect the **current state** of the Flyer. If DHCP is enabled, they will always return the dynamically assigned addresses which may or may not match the current CONFIG values.

Aliases

Aliases are used by prefixing the file specification for a LOAD or OPEN command with "A0:" through "A3:".

The value of the alias (if any) will be used to replace the alias prefix. For example, imagine you have a number of programs stored on a website and you are loading them with something similar to the following:

```
load"http:myserver.com/programs/ufogame.prg",7
```

Let's configure an alias to make our lives easier. Remember, the alias value must be URL encoded:

```
open 7,7,15
print#7, "config:a2=http%3amyserver%2ecom%2fprograms%2f"
close 7
```

This will configure alias 2 ("A2:") to "http:myserver.com/programs/". A URL encoding table can be found in **Appendix A**.

Now all we need to do to load our program is this:

```
load”a2:ufogame.prg”,7
```

Disk Management

The Flyer contains 4MB of flash memory, of which approximately 3.5MB is available for disk storage. The Flyer was designed around the concept of cloud storage and was not intended to be a mass storage device. As such, the flash memory should be considered more of a cache rather than a general disk storage area.

A directory of all the disks currently cached on the Flyer, along with the amount of free space remaining, may be retrieved and displayed with the following BASIC commands:

```
load”$$”,7
list
```

This will return a listing similar to the following:

```
0 “local disks      “ 00 00
1 “Blue Max”          d64
2 “Stuff”             d64
3 “Atari Games”       d64
4 “PET Games 1”       d64
0 -------------------------
0 used: 703k/3424k
0 free: 2721k
0 -------------------------
```

Downloading/uploading disks to/from the Flyer is covered in the next two sections. In this section we will just cover commands used for managing disks already cached on the device, as well as creating brand new disks locally.

Disk commands are issued over the command channel (#15).

Adding a Disk

A new disk can be added to the Flyer at any time with the following command:

```
open 7,7,15
print#7, ”disk-add:d64,Disk Label”
close 7
```

This example will create a new d64 disk image on the Flyer with the label "Disk Label". Other currently supported disk image types are d71, d81, d80 and d82.

It is important to remember that this is the equivalent of creating a new blank unformatted diskette, and the label can be thought of as the handwritten label sticker on the exterior of the diskette. The label is also displayed on the Flyer's LCD screen when the disk is currently selected.

The disk may be formatted using the Flyer's disk unit (device #8 - #15) after the disk is made active by selecting it on the Flyer's display. Note that the newly added disk will be automatically selected as the active disk.

Relabeling a Disk

Relabeling a disk can be performed with the following command:

```
open 7,7,15
print#7, "disk-relabel:INDEX=New Name"
close 7
```

where INDEX is the index of the disk to relabel (the first disk is index 1). The disk indices are also displayed to the left of the disk labels when listing the disks currently on the device with `load"$$",7` (see the example at the beginning of this section).

Removing a Disk

Removing (scratching) a disk is performed with the following command:

```
open 7,7,15
print#7, "disk-scratch:INDEX"
close 7
```

where INDEX is the index of the disk to remove (the first disk is index 1). **This operation cannot be undone!** Double check you have specified the correct index before removing a disk, and remember that indices may shift after you remove a disk. This is very important if you are removing multiple disks. For example, after removing the first of three disks (index 1), the indices of disks 2 and 3 will shift down to 1 and 2, since the first disk on the Flyer will always be index 1.

Removing ALL Disks From the Flyer

Removing all disks from the Flyer is performed with the following command:

```
open 7,7,15
print#7, "disk-wipe:"
close 7
```

This is a very powerful command - use with caution!

Querying Disk Information

The following program will retrieve the number of disks currently cached on the Flyer:

```
10 open 7,7,15
20 print#7, ”disk-count”
30 input#7, a, b$, c, d
40 print c;” disks cached”
50 close7
```

When issuing the DISK-COUNT: command, the number of disks will be returned in the third status field.

The following program will return the name and type of a disk:

```
10 open 7,7,15
20 print#7, ”disk-query:INDEX”
30 input#7, a, b$, c, d
40 print “name: “;b$;” type: “;c
50 close7
```

where INDEX is the index of the disk to query (the first disk is index 1). The type of disk is returned in the third status field:

```
0 = d64
1 = d71
2 = d81
3 = d80
4 = d82
```

Mounting a Disk

Mounting a disk can be performed with the following command:

```
open 7,7,15
print#7, ”disk-mount:INDEX”
close 7
```

where INDEX is the index of the disk to mount (the first disk is index 1). The disk indices are also displayed to the left of the disk labels when listing the disks currently on the device with `load”$$”,7` (see the example at the beginning of this section).

Mounting a disk using this command is the same as physically selecting a particular disk using the button on the Flyer.

Accessing a Cloud Server

The default behavior of LOAD and SAVE for the control unit (device #7) is to exchange disk and program data with the currently selected cloud server.

Up to four cloud server configurations may be stored in the Flyer. The active cloud server can be chosen using the button on the Flyer. If the cloud selection page is not currently displayed on the LCD screen, use long button presses to change display modes. The active cloud can then be changed using short button taps.

Information on how to set up your own cloud server is given in a later chapter. Sample scripts are also available on retroswitch.com.

The Flyer is preconfigured with commodoreonline.com as the default cloud server. However, as authentication is required for this server, you will first need to register for a free account and update the Flyer with your chosen username and password before continuing.

Listing Programs and Disks

A listing of programs and disks available on the cloud server can be loaded just like a standard disk directory:

```
load”$”,7
list
```

Wildcards may be used to refine your search. Let’s only list programs and disks that start with the letters PET:

```
load”$:PET*”,7
list
```

Program Access

LOADing and SAVEing without any special parameters or prefixes will exchange program data with the cloud server. Let write a simple program and save it online:

```
new

10 print”Hello, World!”

save”Hello”,7
```

After the “ready” prompt appears, glance over to the LCD screen on the Flyer and make sure no error messages appear. If you do see an error message, pressing the button will clear it and return the display to normal. The most likely message you might see at this point is an

authentication failure, meaning you either forgot to configure the Flyer with your commodoreonline.com username/password, or they were entered incorrectly.

To retrieve your program back from the server, simply load it:

```
new
load”Hello”,7
list
```

To save over an existing program on the server, prefix “@:” to the filename, just as you would when saving over an existing program on a standard Commodore disk drive. For example:

```
save”@:Hello”,7
```

If you omit the “@:” and a program with the same name already exists on the server, the LCD screen will display the error message “FILE EXISTS” after you attempted to save it.

Disk Access

To upload/download actual disk images, specify “DISK:” followed by the label of the disk you want to operate on. Assuming we have a disk in our commodoreonline.com collection named “PET Games”, we would transfer it to the Flyer as follows:

```
load”disk:PET Games”,7
```

Saving disks to the server works the same way. Let’s say we removed a broken game from the PET Games disk and wish to save our updated disk image back to the server. Similar to programs, if you are saving over a disk image which already exists on the server, prefix the disk label with “@DISK:”.

```
save”disk:PET Games”,7
```

This should have resulted in the error message “DISK EXISTS” on the Flyer’s LCD display.

```
save”@disk:PET Games”,7
```

Now our updated disk image is saved properly.

HTTP Protocol

Overview

The HTTP protocol is available for use on all general communication channels. It is a bidirectional protocol, but since it is based around a single server transaction, an HTTP data channel will be in write-only mode prior to the server transaction, and read-only mode after the server transaction. Only one HTTP transaction may be performed after the channel is opened.

The command channel is used to control exactly when the server transaction occurs for a particular communication channel by sending the "HTTP-TRANSACT:" command. For example:

```
open 7,7,15
open 2,7,2,"http:somesite.com/script.php?f=3"
print#7,"http-transact:2"
close2:close7
```

We first open the command channel, then open general channel #2 using the http protocol, followed by the server address we want to load. Since we don't need to send any additional information (other than the simple query string "?f=3") we trigger the server transaction immediately by sending the "HTTP-TRANSACT:" command through the command channel, followed by the channel number of our HTTP transaction.

At this point, any data received back from the server will be buffered by the Flyer and could be read back from channel 2 by using INPUT# or GET#.

Note that HTTP header information is not returned. The Flyer will only return the content of the transaction.

Additional data may be buffered and sent as part of the HTTP transaction. Any data written to the channel before the transaction will be combined and sent as a single file object named "data" with the filename "data.bin".

In addition, POST data may be sent by issuing any number of HTTP-POST: commands through the command channel before the transaction is performed.

This next example will demonstrate sending this additional data, as well as what a server-side script might look like to process this data.

```
10 open7,7,15
20 open2,7,2,"http:somesite.com/httptest.php"
30 print#7,"http-post:2,username=slappy"
40 print#7,"http-post:2,password=dagnabbit"
50 print#2,"This is some generic data. "
```

```
60 print#2,”This is some more generic data.”
70 print#7,”http-transact:2”
80 input#2,a$
90 close2:close7
100 print “Response: “;a$
```

We open the command and communication channels just as we did in the last example. Then we set a couple of POST variables via the command channel by sending the “HTTP-POST:” command, supplying the HTTP channel to update (2), followed by the key/value pair.

- ❖ Note: The ‘key’ portion of the HTTP-POST: command will automatically be converted from PETSCII to ASCII by the Flyer.

General data is simply written to the HTTP channel itself.

Here is an example of what httptest.php might look like to handle this additional data:

```
<?php

 require_once 'Retroswitch.inc.php';

 $username = isset( $_POST['username'] ) ?
Retroswitch::PetsciiToAscii( $_POST['username'] ) : "";
 $password = isset( $_POST['password'] ) ?
Retroswitch::PetsciiToAscii( $_POST['password'] ) : "";

 if( strcmp( $username, "slappy" ) || strcmp( $password, "dagnabbit"
) )
 {
   echo Retroswitch::AsciiToPetscii( "Not Authorized" );
   exit;
 }

 if( isset( $_FILES['data'] ) )
 {
   $data = file_get_contents( $_FILES['data']['tmp_name'] );
   // Strip delimiters from our data (replace with spaces) so we can
   // return it in one chunk for testing...
   $data = str_replace( array( ",", "\n", "\r" ), " ", $data );
   echo $data;
   exit;
 }

 echo Retroswitch::AsciiToPetscii( "No Data" );

?>
```

If you would like to experiment with this example, this script has been uploaded to retroswitch.com/scripts/httptest.php.

Using HTTP with Load and Save

The HTTP protocol may also be used with LOAD and SAVE, thanks to some additional processing that happens automatically when HTTP is used on channels 0 or 1.

When using the HTTP protocol with LOAD, the data returned from the server will simply be loaded into program memory. As far as the computer knows, the data is coming from a standard storage device such as a disk or tape drive.

Loading a program from an HTTP server is simple:

```
load”http:somesite.com/programs/hello.prg”,7
```

We can also download disk images to store on the Flyer by appending ”,d=label”, where “label” is used to name and identify the virtual diskette. For example:

```
load”http:somesite.com/disks/disk045.d64,d=PET Games”,7
```

This will download the file disk045.d64 and treat it as a disk image to be stored on the Flyer, giving it the label “PET Games”.

Assuming all went well, after the load completes you will be able to switch to this disk using the button on the Flyer and start using it through the disk unit.

One last detail however - the LOAD command always expects *something* to be loaded into the computer’s memory, so if a disk is downloaded in this manner, the Flyer automatically returns a disk listing back to the computer showing all the disks currently stored and how much memory is currently in use / available. This is the same listing returned as if you typed:

```
load “$$”, 7
```

It should be noted that the programs and disks you specify don’t have to be actual files (such as hello.prg, or disk045.d64). They could just as easily be server scripts that return the appropriate data as binary content. Login credentials (or anything else you can think of!) could even be specified by using a query string.

SAVEing via the HTTP protocol is similar, however it requires some kind of script/processing on the server end, since we obviously can’t just save data to an ordinary HTTP server.

Saving/sending a program to an HTTP target is no different than loading:

```
save”http:somesite.com/scripts/receiver.php”,7
```

Internally, the data is sent as file content in a MIME multipart message. The file content is named “data”. In PHP, the received data would be available in the predefined variable $_FILES[‘data’].

Saving a disk image currently stored on the Flyer is once again identical to the loading example, however the specified label indicates which local disk to send:

```
save”http:somesite.com/scripts/receiver.php,d=PET Games”,7
```

TCP Protocol

Overview

The TCP protocol is available for use on all general communication channels. It is a bidirectional protocol, and up to 3 TCP channels may be open at any one time.

The format for opening a TCP channel is simple:

```
open 2,7,2,”tcp:192.168.1.115,23”
```

This particular example will open a TCP/IP connection to a server (presumably on a local network) on port 23 (presumably a telnet server).

Sending Data

Sending data is as simple as PRINT# ing data to the channel. Each block of data written will be sent immediately as a new packet.

Receiving Data

Receiving data again uses standard channel I/O protocol, namely GET# and INPUT#. However, in order to know if (and how much) data is available to read, the command channel must be used.

We use the command channel command NET-AVAIL: followed by the TCP channel number. This will return the number of queued bytes in the third status field when reading back the command channel status. As usual, if the first status field is not 0, an error has occurred.

The following example will open a connection to a custom server which expects a single character command, and returns a single character response (an unlikely, but very simple example). Note that no error checking is performed.

```
10 open 7,7,15
20 open 2,7,2,"tcp:someserver.com,2112"
30 print#2,chr$(197)
40 print#7,"net-avail:2":input#7,a,b$,c,d:if c=0 then goto 40
50 get#2,r$
60 print"Response: ";asc(r$)
70 close 2
80 close 7
```

Listening for Connections

The Flyer is able to listen for incoming connections as well. A maximum of 2 channels can be listening for connections at any one time.

To open a channel to listen for incoming connections, simply use the LISTEN: channel protocol, specifying the port you want to listen on:

```
open 2,7,2,"listen:5000"
```

The NET-STAT: (or simply N-S:) command may be issued via the command channel to determine if a connection has been accepted, as it returns the connection status as part of its response.

Assuming the command channel has already been opened and bound to file ID 7,

```
open 2,7,2,"listen:5000"
```

the connection status may be checked as follows:

```
print#7,"n-s:2"
input#7,a,b$,c,d
print"connected: ";d
```

'd' will be either 1 or 0, indicating whether or not a connection has been established on channel 2.

Once a connection has been established, normal communication can proceed as usual via PRINT#, GET# and INPUT#.

Connection Information

Information about a TCP connection can be retrieved using the C-I: (connection info) command. This is particularly useful when listening for connections, as it will also report information about the remote connection. For example:

```
print#7,”c-i:2”
input#7,a,b$,c,d
print”error: ”;a
print”remote ip: ”;b$
print”remote port: ”;c
print”connected: ”;d
```

As with N-S:, ‘d’ will be either 1 or 0, indicating connection status.

Chapter 3:
The Disk Unit

The following topics will be discussed in this chapter:

- ✓ Overview and purpose of the disk device.
- ✓ Review of basic disk drive commands.

Overview

The disk unit is a hybrid disk drive emulator which allows you to access and use the disks stored on the Flyer. The term "hybrid" is used since the emulated disk drive combines features of the 1541, 1571 and 1581 disk drives, with more planned for future updates.

Most commonly used commands and features are supported. This chapter will cover basic disk commands you will likely use the most.

The disk unit is initially set to device address #10, however this can be changed using the button on the front of the Flyer. If not already displayed, switch to the device selection mode using long button presses. Then use short button taps to change the device address. The changes take effect immediately.

The examples in the chapter will assume the device address has been changed to #8.

Also note that while almost all standard disk commands are supported (including direct access and memory read/write), we will only cover the very basics. Please refer to an actual disk drive manual (1541 for example) for more information.

Mounting Disks

Mounting disk images is done by selecting the desired disk using the button on the front of the Flyer. If not already displayed, switch to the disk selection mode using long button presses. Then use short button taps to change the current disk.

Each time the disk is changed, it is immediately mounted in the disk unit and ready for use.

Loading and Saving Programs

Loading programs is accomplished with the LOAD command. For example:

```
load”calculator”,8
load”game”,8,1
```

Note the extra ”,1” in the second example. This indicates that the program should be loaded into the exact same location in memory from which it came.

Otherwise, it will be loaded at the start of BASIC memory on your computer. This is how a BASIC program written for the VIC-20 may be loaded and run on the Commodore 64 for example (as long as it didn’t use any features specific to the VIC-20 that is).

Note that Commodore PETs do NOT support this feature. Programs will always be loaded as if ”,1” was supplied, meaning a program written for the Commodore 64 (or any other Commodore computer that doesn’t share the same memory layout as the PET) will not load correctly on the PET in any case.

Saving programs is similarly accomplished with the SAVE command. For example:

```
save”myprogram”,8
```

This simply saves the current BASIC program to disk with the name “myprogram”.

To see a list of all the files on a disk, you LOAD the disk directory into the computer and LIST it. “$” indicates you want to load the disk directory. For example:

```
load”$”,8
list
```

Basic Disk Management

Here we’ll cover a few of the more common disk management commands.

These commands are issued over communication channel #15, known as the ”command channel”. The examples will all follow the same basic pattern. The command channel is opened and given the command, then the command channel is closed. This is done using the BASIC commands OPEN and CLOSE as follows:

```
open 15,8,15,”<command>” : close 15
```

Formatting a Disk

Formatting a disk is performed with the N0: (new) command and takes one or two additional arguments – the new name of the disk, and the 2 character identifier for the disk. Omitting the 2 character identifier performs a "fast" format, where only the table of contents is initialized. Here are a couple of examples:

```
open 15,8,15,"n0:mydisk,00":close15
open 15,8,15,"n0:stuff":close15
```

The second example performs a fast format.

Renaming a File

Renaming a file is accomplished using the RENAME0: or R0: command as follows:

```
open 15,8,15,"r0:newname=oldname":close15
```

Deleting a File

Deleting (or "scratching") a file is accomplished using the SCRATCH0: or S0: command as follows:

```
open 15,8,15,"s0:filename":close15
```

Chapter 4: Examples

The following topics will be discussed in this chapter:

✓ Sample projects which demonstrate various aspects of the Flyer.

Writing a Fortune Cookie Program

This example will demonstrate how easy it is to write a simple internet-enabled program in BASIC using the Flyer. Our goal is to write a program to fetch one of potentially thousands of "fortune cookie" messages from an HTTP server and display them to the user.

The Server

The first thing we'll do is write a PHP script to serve messages back to the client. To keep things simple, we'll just choose one of three hard coded messages (a "real" solution would most likely parse/fetch data from real fortune files or a database, but that's outside the scope of this example).

```
<?php

require_once 'Retroswitch.inc.php';

$fortunes = array(
  "As a computer, I find your faith in technology amusing!",
  "If a pig loses its voice, is it disgruntled?",
  "Confidence: The feeling you have before you understand the
situation."
);

$index = rand( 0, count( $fortunes ) - 1 );
$petscii = Retroswitch::AsciiToPetscii( $fortunes[$index] );
echo urlencode( $petscii );

?>
```

This script has been placed at www.retroswitch.com/scripts/fortune.php.

A couple of things are worth pointing out...

First, since Commodore computers do not use standard ASCII (although it's very close), we must convert any text data before outputting it to ensure it will be displayed properly (this will always be an issue when communicating with "the outside world" from your Commodore). The Retroswitch utility class (available on retroswitch.com) contains PHP code for converting between ASCII and PETSCII.

Second, note that we are url encoding the returned fortunes, since we'll be reading them with the INPUT# instruction. INPUT# is designed to stop on various delimiters (such as a comma), and url encoding allows us to retrieve the entire response in one chunk. Otherwise we would have to use GET# and receive one character at a time, which is much slower. Of course, url decoding in BASIC is also quite slow and is the main cause for the delay when retrieving fortunes in this example. It is a perfect candidate for a speedy little machine language routine, however!

The Client

Next we'll write a BASIC program that will fetch and display a random message. This program should be entered and run in lower case mode, as with all the other examples in this manual.

This first section is the main loop of the program. It simply fetches and displays a fortune, then prompts the user whether they would like to continue:

```
10 gosub 100: rem returns random fortune in m$
20 print:print m$:print
30 print"Would you like another (Y/N)?"
40 get q$: if q$="" then goto 40
50 if q$="y" OR q$="Y" then goto 10
60 print:print"Have a nice day!"
70 end
```

Next is the subroutine which performs the HTTP request to retrieve the fortune. We first open the command channel so we can check error status and issue commands. Then we open a general communications channel (2 in this example), using the HTTP protocol and supplying the path to our PHP script.

We can add additional POST and FILE data to the HTTP request at this point since the server transaction has not yet occurred. However, for this example we're only interested in the server response so we initiate the transaction immediately by issuing the HTTP-TRANSACT command over the command channel, specifying the communication channel with the pending transaction (#2). This pattern should be very familiar to anyone that has done direct-access disk programming.

Once the transaction has occurred, the server's response can now be read. We read the response into E$, url decode it, then return the result back to the main loop in M$:

```
100 open 7,7,15: rem open command channel
110 open 2,7,2,"http:retroswitch.com/scripts/fortune.php"
120 print#7,"http-transact:2":gosub 500
130 input#2,e$:gosub 500
140 close2:close7
150 gosub 200: rem url decode e$, result in u$
160 m$ = u$
170 return
```

Next we have a couple of subroutines used for url decoding:

```
200 rem url decode from e$ to u$
210 sl=len(e$):u$="":ifsl=0thenreturn
220 fori=1tosl:ac=asc(mid$(e$,i,1))
230 ifac=43thenu$=u$+" ":goto280
240 ifac<>37thenu$=u$+chr$(ac):goto280
250 an=asc(mid$(e$,i+1,1)):gosub300:h0=dn
260 an=asc(mid$(e$,i+2,1)):gosub300:h1=dn
270 i=i+2:u$=u$+chr$(h0*16+h1)
280 nexti
290 return

300 rem convert ascii nybble to dec from an to dn
310 ifan>=48andan<=57thendn=an-48:return
320 ifan>=65andan<=90thendn=an-55:return
330 ifan>=97andan<=122thendn=an-87:return
340 dn=0:return
```

And finally a short subroutine which we call periodically during the network I/O to check for any error conditions. If an error is detected, we print the error code and message before quitting.

```
500 input#7,a,b$,c,d
510 if a=0 then return
520 close2:close7
530 print"Error: ";a
540 print"Message: ";b$
550 end
```

That's it! For your typing convenience, this sample program can be downloaded at the following location:

```
load"http:retroswitch.com/programs/fortune.prg",7
```

Appendix A: URL Encoding Table

space	%20
!	%21
"	%22
#	%23
$	%24
%	%25
&	%26
'	%27
(	%28
)	%29
*	%2A
+	%2B
,	%2C
-	%2D
.	%2E
/	%2F
0	%30
1	%31
2	%32
3	%33
4	%34
5	%35
6	%36
7	%37
8	%38
9	%39
:	%3A
;	%3B
<	%3C
=	%3D
>	%3E
?	%3F
@	%40
A	%41

B	%42
C	%43
D	%44
E	%45
F	%46
G	%47
H	%48
I	%49
J	%4A
K	%4B
L	%4C
M	%4D
N	%4E
O	%4F
P	%50
Q	%51
R	%52
S	%53
T	%54
U	%55
V	%56
W	%57
X	%58
Y	%59
Z	%5A
[	%5B
\	%5C
]	%5D
^	%5E
_	%5F
`	%60
a	%61
b	%62
c	%63

d	%64
e	%65
f	%66
g	%67
h	%68
i	%69
j	%6A
k	%6B
l	%6C
m	%6D
n	%6E
o	%6F
p	%70
q	%71
r	%72
s	%73
t	%74
u	%75
v	%76
w	%77
x	%78
y	%79
z	%7A
{	%7B
|	%7C
}	%7D
~	%7E
	%7F
€	%80
	%81
‚	%82
ƒ	%83
„	%84
…	%85

†	%86
‡	%87
ˆ	%88
‰	%89
Š	%8A
‹	%8B
Œ	%8C
	%8D
Ž	%8E
	%8F
	%90
‘	%91
’	%92
“	%93
”	%94
•	%95
–	%96
—	%97
˜	%98
™	%99
š	%9A
›	%9B
œ	%9C
	%9D
ž	%9E
Ÿ	%9F
	%A0
¡	%A1
¢	%A2
£	%A3
	%A4
¥	%A5
|	%A6
§	%A7

¨	%A8
©	%A9
ª	%AA
«	%AB
¬	%AC
¯	%AD
®	%AE
¯	%AF
°	%B0
±	%B1
²	%B2
³	%B3
´	%B4
µ	%B5
¶	%B6
·	%B7
¸	%B8
¹	%B9
º	%BA
»	%BB
¼	%BC
½	%BD
¾	%BE
¿	%BF
À	%C0
Á	%C1
Â	%C2
Ã	%C3
Ä	%C4
Å	%C5
Æ	%C6
Ç	%C7
È	%C8
É	%C9

Ê	%CA
Ë	%CB
Ì	%CC
Í	%CD
Î	%CE
Ï	%CF
Ð	%D0
Ñ	%D1
Ò	%D2
Ó	%D3
Ô	%D4
Õ	%D5
Ö	%D6
	%D7
Ø	%D8
Ù	%D9
Ú	%DA
Û	%DB
Ü	%DC
Ý	%DD
Þ	%DE
ß	%DF
à	%E0
á	%E1
â	%E2
ã	%E3
ä	%E4
å	%E5
æ	%E6
ç	%E7
è	%E8
é	%E9
ê	%EA
ë	%EB
ì	%EC
í	%ED
î	%EE
ï	%EF
ð	%F0

ñ	%F1
ò	%F2
ó	%F3
ô	%F4
õ	%F5
ö	%F6
÷	%F7
ø	%F8
ù	%F9
ú	%FA
û	%FB
ü	%FC
ý	%FD
þ	%FE
ÿ	%FF

Appendix B: Software Licenses

Flyer uses the following licensed technologies:

- RSA Data Security, Inc. MD5 Message-Digest Algorithm

```
**********************************************************************
** Copyright (C) 1990, RSA Data Security, Inc. All rights reserved.  **
**                                                                   **
** License to copy and use this software is granted provided that    **
** it is identified as the "RSA Data Security, Inc. MD5 Message-     **
** Digest Algorithm" in all material mentioning or referencing this  **
** software or this function.                                        **
**                                                                   **
** License is also granted to make and use derivative works          **
** provided that such works are identified as "derived from the RSA  **
** Data Security, Inc. MD5 Message-Digest Algorithm" in all          **
** material mentioning or referencing the derived work.              **
**                                                                   **
** RSA Data Security, Inc. makes no representations concerning       **
** either the merchantability of this software or the suitability    **
** of this software for any particular purpose.  It is provided "as  **
** is" without express or implied warranty of any kind.              **
**                                                                   **
** These notices must be retained in any copies of any part of this  **
** documentation and/or software.                                    **
**********************************************************************
```