Datakey®

The power of memory. Secured.

Series III Programmer

Operation Manual

Table of Contents

Hardware Background

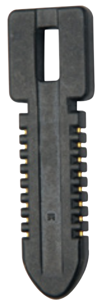
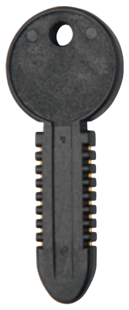
Datakey Serial Memory devices each contain a commercially available memory chip. The memory chip is designed to:

- operate within a specified supply voltage range,
- communicate via a specific protocol (SPI Flash, SPI EEPROM, I2C, or Microwire), and
- have a specified memory capacity (i.e. size) – usually measured in bits.

The portable memory devices are available are available in several different form factors: key, Slimline token, Plug token, and Bar token. Each shape has a corresponding receptacle that allows for quick insertion/removal and electrical connection to the device. See the appropriate protocol's interface specification document for the various device shapes, their associated receptacle pin numbering conventions, and signal locations.

The Series III Programmer is a microcontroller-based product that contains the firmware and interface circuitry necessary to allow a host computer to communicate (via USB) with a Datakey key or token that uses the SPI Flash, SPI EEPROM, I^{2}C, or Microwire protocol. Once power is applied to the Series III Programmer (via the host computer's USB port), its two-color LED will illuminate. The LED will be red while reading, writing, or erasing of the key is in progress; green, otherwise. If the token is withdrawn from the Series III Programmer while the LED is red, data corruption might occur.

Datakey memory devices have a family name embossed on them that indicates their communications protocol, memory technology and form factor. In addition, they have a size code, a voltage code, and a date code embossed on them. See chart on page 3.

KEY **SQUARE HEAD**	**KEY** **ROUND HEAD**	**SLIMLINE** **TOKEN**
		
DK	IIK, ISK, LCK, SFK, SFK5V, SSK	IIT, IST, LCS, SFT, SLT, SST
EXTENDED SLIMLINE TOKEN	**PLUG**	**BAR**
IIX, ISX, LCX, SFX, SSX	ISP, KSD, SSP	ISB, LCB, SSB

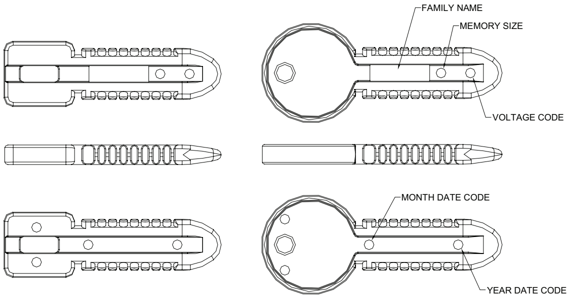
Key/ Token/ Plug Size Codes

CODE	SIZE IN BITS	BYTES
A	1 Kb	128 bytes
B	2 Kb	256 bytes
C	4 Kb	512 bytes
D	8 Kb	1 KB
E	16 Kb	2 KB
F	32 Kb	4 KB
G	64 Kb	8 KB
H	128 Kb	16 KB
I	256 Kb	32 KB
J	512 Kb	64 KB
K	1 Mb	128 KB
L	2 Mb	256 KB
M	4 Mb	512 KB
N	8 Mb	1 MB
O	16 Mb	2 MB
P	32 Mb	4 MB
Q	64 Mb	8 MB
R	128 Mb	16 MB
S	256 Mb	32 MB
T	512 Mb	64 MB
U	1 Gb	128 MB
V	2 Gb	256 MB
W	4 Gb	512 MB
X	8 Gb	1 GB
Y	16 Gb	2 GB
Z	32 Gb	4 GB

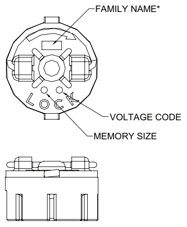
NFX and RUGGEDrive Size Codes

CODE	SIZE IN BITS	BYTES
A	1 Gb	128 MB
B	2 Gb	256 MB
C	4 Gb	512 MB
D	8 Gb	1 GB
E	16 Gb	2 GB
F	32 Gb	4 GB
G	64 Gb	8 GB
H	128 Gb	16 GB
I	256 Gb	32 GB
J	512 Gb	64 GB
K	1 Tb	128 GB
L	2 Tb	256 GB
M	4 Tb	512 GB
N	8 Tb	1 TB

KEY MARKING

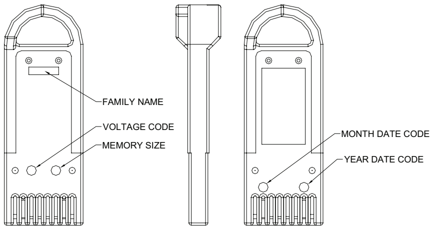


PLUG MARKING

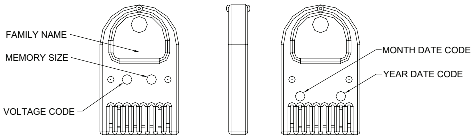


* LEGACY INFORMATION: NO MARKS OR BLACK TOP IS 1 KB, 5 V MICROWIRE INTERFACE

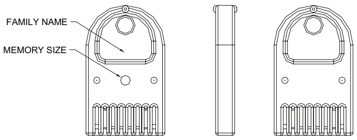
EXTENDED TOKEN MARKING



TOKEN MARKING

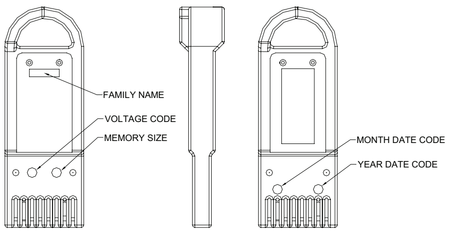


LEGACY TOKEN

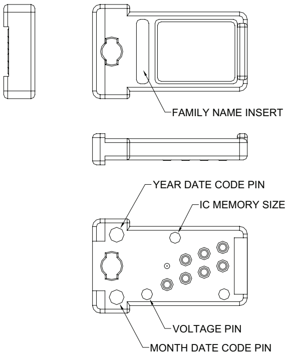


APPLIES TO MANUFACTURING DATES OF 4/2003 - 3/2005

RUGGEDRIVE TOKEN MARKING



BAR MARKING



Date Codes

MONTH		YEAR			
1	Jan	1	2001	2011	2021
2	Feb	2	2002	2012	2022
3	Mar	3	2003	2013	2023
4	Apr	4	2004	2014	2024
5	May	5	2005	2015	2025
6	Jun	6	2006	2016	2026
7	Jul	7	2007	2017	2027
8	Aug	8	2008	2018	2028
9	Sep	9	2009	2019	2029
0	Oct	0	2010	2020	2030
N	Nov				
D	Dec				

Voltage Codes

CODE	VOLTAGE RANGE
Blank	4.5 - 5.5 V
5	3.0 - 3.6 V
4	2.7 - 5.5 V
3	2.7 - 3.6 V
2	1.8 - 3.6 V

Although the memory chip within a device has a range of voltages over which it can operate, the Series III Programmer can only be configured to apply either 3.3 volts or 5 volts to the device.

Sample Applications

There are two different designs within the sample applications included on the flash drive/download: the C++ implementation and the C#/Visual Basic implementation. The applications have many features in common and some features that are unique to one or the other. The operation of each of the applications is described separately. Refer to the correct corresponding section.

C++ Implementation

To start the application, do the following:

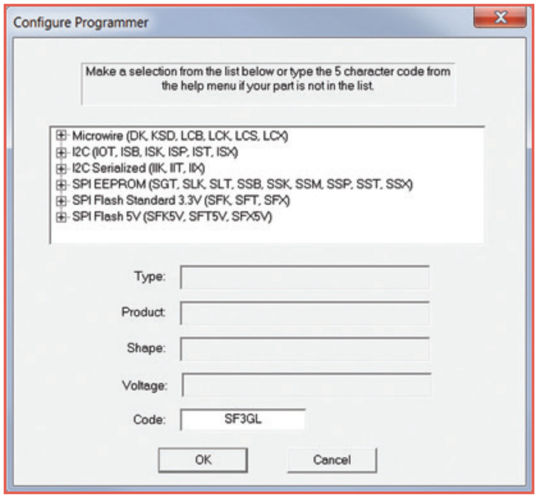
- attach the Series III Programmer to the host computer's USB 2.0 port, then
- double-click the Series III Programmer desktop icon on the host computer.

Configuration

Each time the sample application is started, it will prompt the user to configure the attached Series III Programmer for the type of device that is to be expected. This is a crucial step, it tells the Series III Programmer:

- what protocol is to be used for communication,
- what supply voltage to apply to the device (either 3.3 or 5 volts), and
- the memory capacity of the device.

The sample application cannot query this information from an attached device, it must be told what to expect. The Series III Programmer does remember the type of device for which it was last configured and the corresponding 5-character code is displayed in the **Code** text box of the **Configure Programmer** form.


Configure Programmer

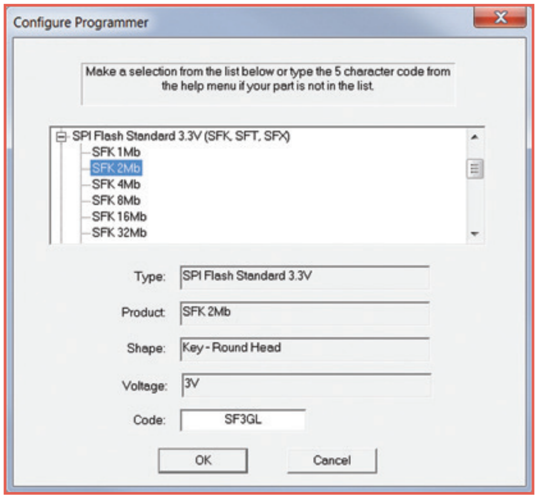
Make a selection from the list below or type the 5 character code from the help menu if your part is not in the list.

- ⊞ Microwire (DK, KSD, LCB, LCK, LCS, LCX)
- ⊞ I2C (IOT, ISB, ISK, ISP, IST, ISX)
- ⊞ I2C Serialized (IIK, IIT, IIX)
- ⊞ SPI EEPROM (SGT, SLK, SLT, SSB, SSK, SSM, SSP, SST, SSX)
- ⊞ SPI Flash Standard 3.3V (SFK, SFT, SFX)
- ⊞ SPI Flash 5V (SFK5V, SFT5V, SFX5V)

Type:
Product:
Shape:
Voltage:
Code: SF3GL

Expand the appropriate protocol family and select the specific device to be used in the Series III Programmer. Note that the same 5-character code may be assigned to more than one particular device in the list. That's because the electrical interface to the devices may be the same, even though their physical shapes are different.

Once the desired configuration is selected, click **OK**. In this example, the Series III Programmer is being configured for an SFK 2Mb device.

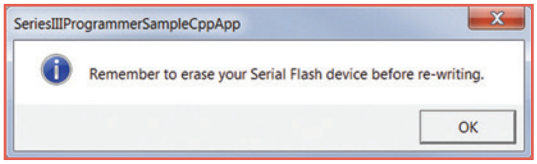

Configure Programmer

Make a selection from the list below or type the 5 character code from the help menu if your part is not in the list.

- ⊟ SPI Flash Standard 3.3V (SFK, SFT, SFX)
  - SFK 1Mb
  - SFK 2Mb
  - SFK 4Mb
  - SFK 8Mb
  - SFK 16Mb
  - SFK 32Mb

Type: SPI Flash Standard 3.3V
Product: SFK 2Mb
Shape: Key - Round Head
Voltage: 3V
Code: SF3GL

Note that if the Series III Programmer is configured for a Serial Flash device, a message will be displayed, reminding the user to manually erase the device before overwriting any existing data. EEPROM devices implement an automatic erase feature whenever new data is written; Serial Flash does not.


SeriesIIIProgrammerSampleCppApp

Remember to erase your Serial Flash device before re-writing.

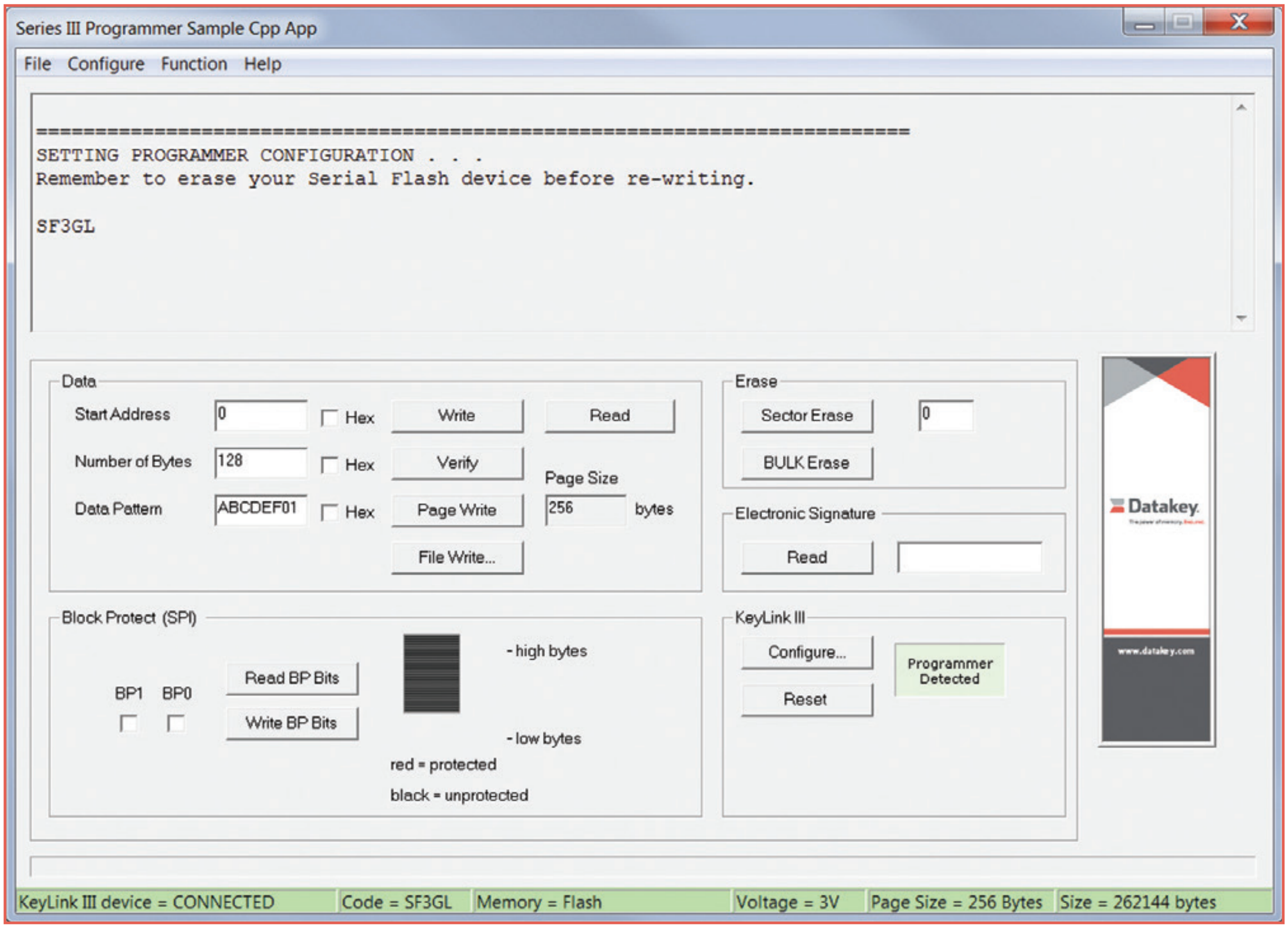
All of the Datakey devices that utilize Serial Flash memory have family names that begin with the letters "SF" (e.g., SFK, SFT, SFX).

Overview

The Sample Cpp App pictured below has:

- a menu bar at the top that offers a choice of **File**, **Configure**, **Function**, and **Help**,
- a message window that shows the results of operations recently performed and the general status of the application,
- a **Data** group box where address limits and data patterns can be specified and where read, write, and compare operations can be performed,
- an **Erase** group box where individual sectors or the entire device can be targeted for erasure,
- an **Electronic Signature** group box where the signature of a Serial Flash device can be displayed,
- a **Block Protect (SPI)** group box where the block protect bits of an SPI device can be read and set/cleared,
- a **KeyLink III** group box where the Series III Programmer can be reconfigured or reset and where the on-line/off-line status of the Series III Programmer can easily be determined,
- a status bar that indicates success (green), failure (red), or progress (blue) of the pending operation, and
- an information bar where information about the software, firmware, configuration, and hardware status are displayed.

Note that not all controls are available or appropriate for all types of devices.



```
===========================================================================
SETTING PROGRAMMER CONFIGURATION . . .
Remember to erase your Serial Flash device before re-writing.

SF3GL
```

Start Address 0 □ Hex

Number of Bytes 128 □ Hex

Data Pattern ABCDEF01 □ Hex

Page Size 256 bytes

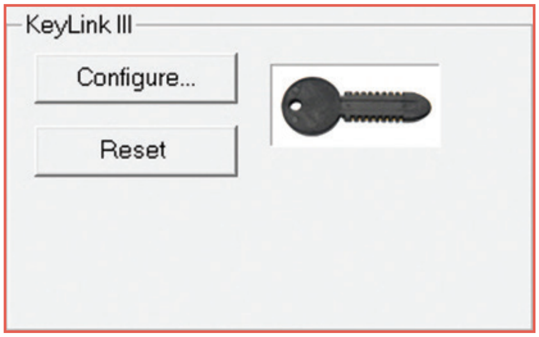
Sector Erase 0

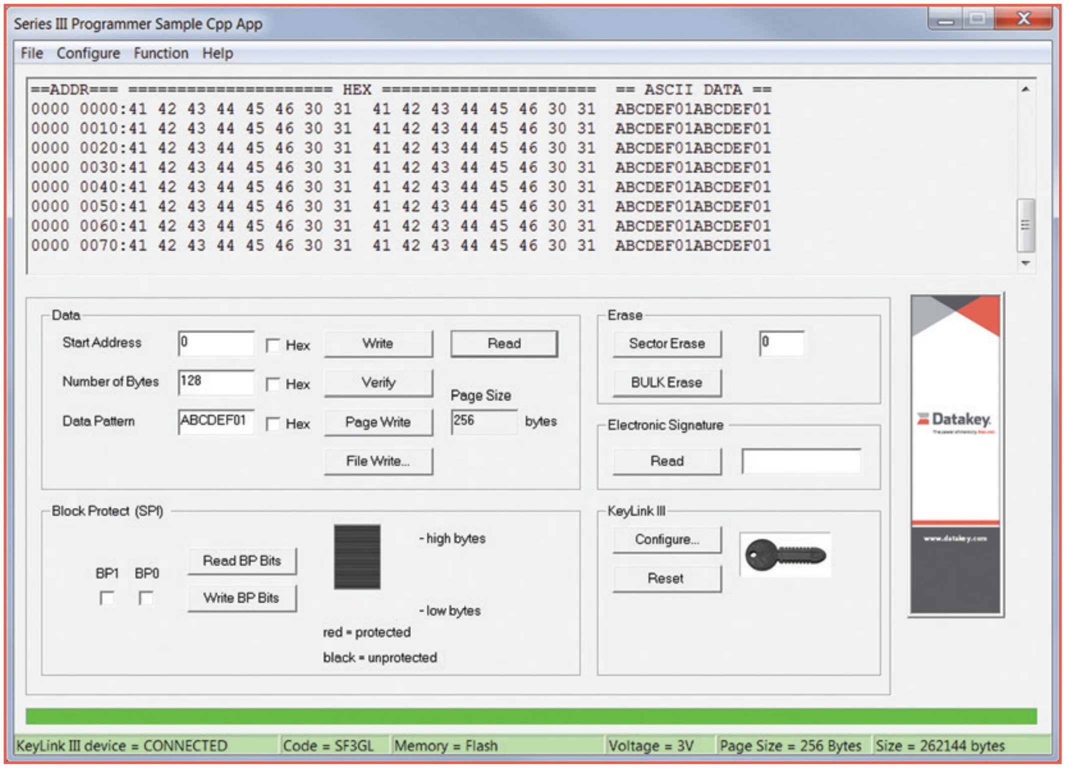
BP1 □ BP0 □

KeyLink III device = CONNECTED | Code = SF3GL | Memory = Flash | Voltage = 3V | Page Size = 256 Bytes | Size = 262144 bytes

Insertion

When a key is inserted into the receptacle, an image of a key will be displayed within the **KeyLink III** group box. (Note that for an actual key type of device, the key must be inserted into the receptacle and turned a quarter turn clockwise before the image of a key will appear.) Behind the scenes, the application is polling the status of the LOFO (last on, first off) signal and when it goes low, the image is displayed.





```
==ADDR=== ===================== HEX ======================  == ASCII DATA ==
0000 0000:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0010:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0020:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0030:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0040:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0050:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0060:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0070:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
```

Start Address: 0 □ Hex

Number of Bytes: 128 □ Hex

Data Pattern: ABCDEF01 □ Hex

Page Size: 256 bytes

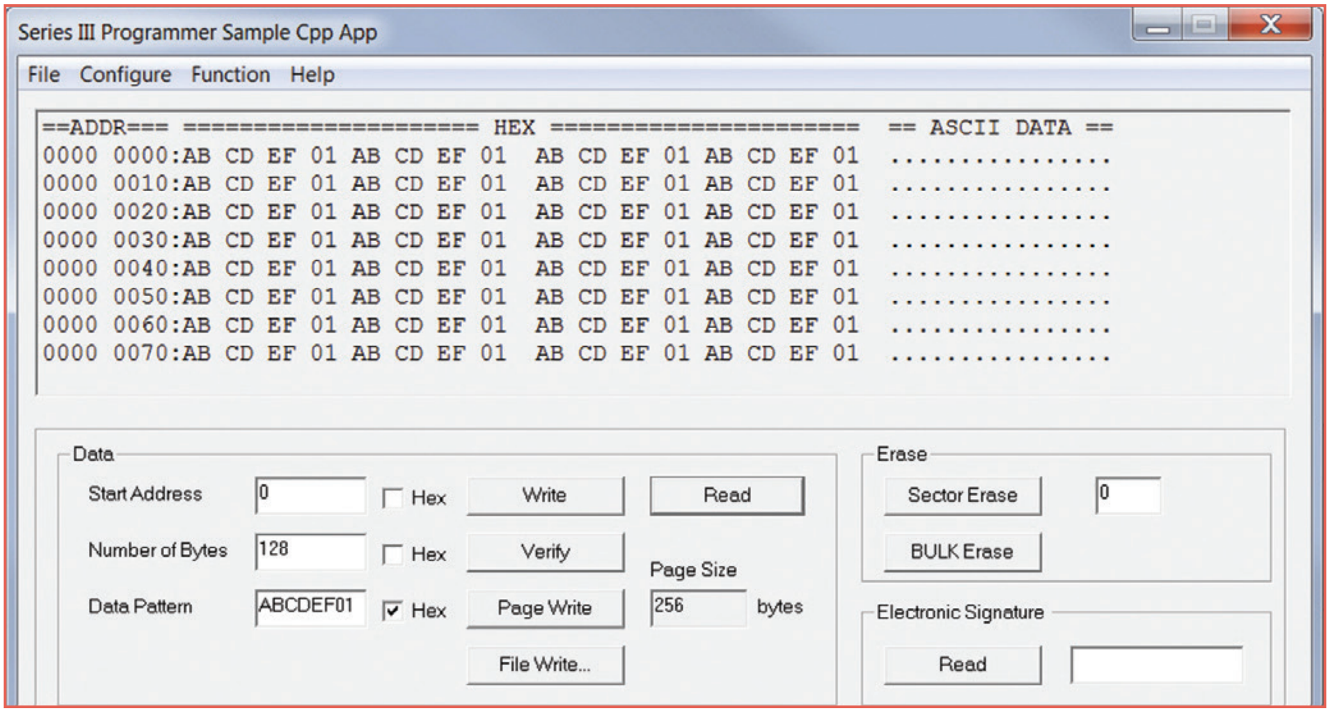
Sector Erase: 0

BP1 □ BP0 □

KeyLink III device = CONNECTED | Code = SF3GL | Memory = Flash | Voltage = 3V | Page Size = 256 Bytes | Size = 262144 bytes

Utilizing the default values in the **Data** group box and clicking the **Write** button and then the **Read** button, a repeating data pattern is first written to and then read back from the inserted key. As per the default values, writing/reading begins at address zero and continues sequentially for 128 bytes.

Because none of the **Hex** checkboxes in the Data group box were checked, the specified **Start Address** and **Number of Bytes** are interpreted as decimal values and the **Data Pattern** is converted to ASCII data.



```
==ADDR=== ====================== HEX ======================  == ASCII DATA ==
0000 0000:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0010:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0020:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0030:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0040:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0050:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0060:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0070:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
```

Start Address: 0 □ Hex

Number of Bytes: 128 □ Hex

Data Pattern: ABCDEF01 ☑ Hex

Page Size: 256 bytes

Sector Erase: 0

Utilizing the same default values in the **Data** group box, but checking the **Hex** checkbox associated with **Data Pattern** and then clicking the **Write** button and then the **Read** button, would display the data shown above. Note that in this case the **Data Pattern** would be interpreted as a sequence of hexadecimal values and written directly to the key as binary, without being converted to its equivalent ASCII codes.

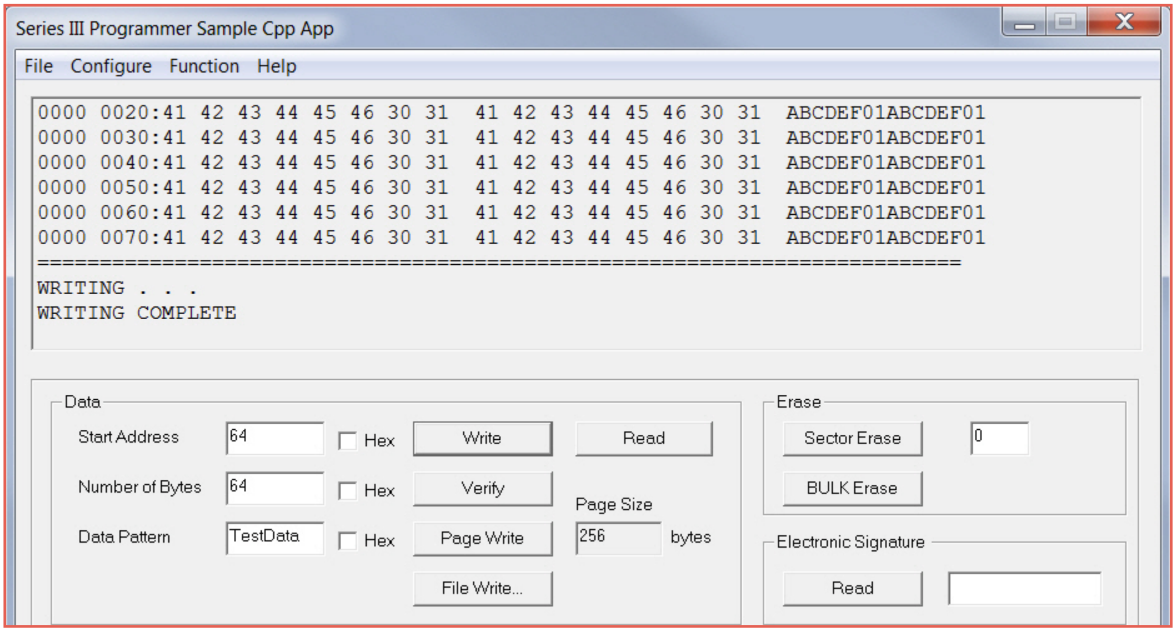
CAUTION: Attempting to send general text as the **Data Pattern** and telling the application to interpret it as hexadecimal characters will yield seemingly unpredictable behavior.

If, within the **Data** group box, the **Page Write** button is clicked instead of the **Write** button, the **Number of Bytes** to be written must be less than or equal to the **Page Size** of the device. The **Page Size** is displayed within the **Data** group box.

If, within the **Data** group box, the **File Write**... button is clicked instead of the **Write** button, the application will prompt for and allow browsing to the file to be written.

Manual Erasure (Serial Flash only)

Here's an example of the type of behavior that can be expected if a Serial Flash device is not manually erased before being rewritten. It is assumed that the first example of writing 128 bytes of data to the key was, in fact, performed.



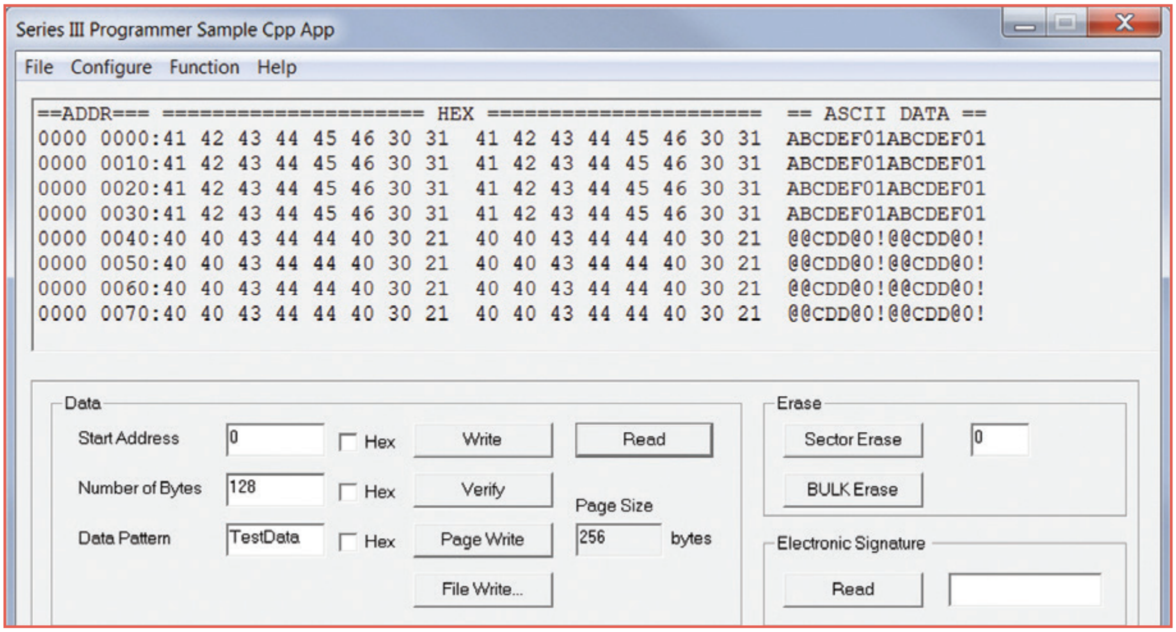
```
0000 0020:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0030:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0040:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0050:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0060:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0070:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
=========================================================================
WRITING . . .
WRITING COMPLETE
```

Data
Start Address: 64 ☐ Hex
Number of Bytes: 64 ☐ Hex
Data Pattern: TestData ☐ Hex
Page Size: 256 bytes

Erase
Sector Erase: 0

Without erasing the initial data, perform the following additional operations:

- enter the values shown above in the **Data** group box for **Start Address**, **Number of Bytes**, and **Data Pattern**,
- click the **Write** button,



```
==ADDR=== ===================== HEX ======================  == ASCII DATA ==
0000 0000:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0010:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0020:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0030:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0040:40 40 43 44 44 40 30 21  40 40 43 44 44 40 30 21  @@CDD@0!@@CDD@0!
0000 0050:40 40 43 44 44 40 30 21  40 40 43 44 44 40 30 21  @@CDD@0!@@CDD@0!
0000 0060:40 40 43 44 44 40 30 21  40 40 43 44 44 40 30 21  @@CDD@0!@@CDD@0!
0000 0070:40 40 43 44 44 40 30 21  40 40 43 44 44 40 30 21  @@CDD@0!@@CDD@0!
```

Data
Start Address: 0 ☐ Hex
Number of Bytes: 128 ☐ Hex
Data Pattern: TestData ☐ Hex
Page Size: 256 bytes

Erase
Sector Erase: 0

- enter the values shown above in the **Data** group box for **Start Address** and **Number of Bytes**, and
- click the **Read** button.

Now look at the 128 bytes of data that was read from the device. The first 64 bytes still exhibit the same repeating pattern of data that was written to them initially, but the final 64 bytes show neither the initial nor the subsequent data pattern. What's going on here?

The answer is that the following rules are obeyed when overwriting data in a Serial Flash device:

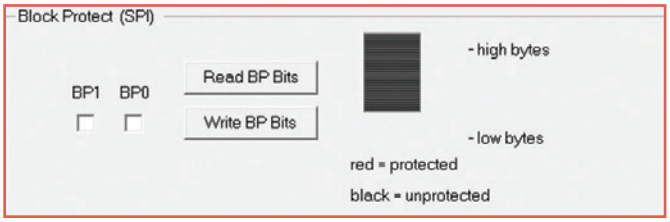
- binary ones in the initial data can remain as binary ones in the subsequent data,
- binary ones in the initial data can be demoted to binary zeroes in the subsequent data,
- binary zeroes in the initial data can remain as binary zeroes in the subsequent data, but
- binary zeroes in the initial data cannot be promoted to binary ones in the subsequent data.

More often than not, what is wanted is to manually erase the previous data before writing new data to any given memory location. That is the only way to promote binary zeroes in the initial data back to binary ones. When any byte is erased, all its bits go to binary ones.

The scope of an erasure can be limited to a single sector at a time or applied to the entire memory space simultaneously. This can be done using the controls in the **Erase** group box. To erase a single sector, specify the sector number and then click the **Sector Erase** button. To erase the entire memory space, click the **Bulk Erase** button.

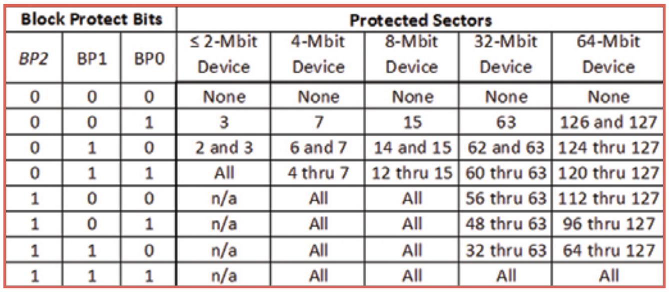
Block Protect (SPI only)

Sections of an SPI EEPROM/Flash memory space can be protected from writing and erasure by utilizing the Block Protect bits, BP0 and BP1. Depending on the memory capacity of the device, there may even be a third Block Protect bit, BP2. Only SPI memory offers Block Protect bits.



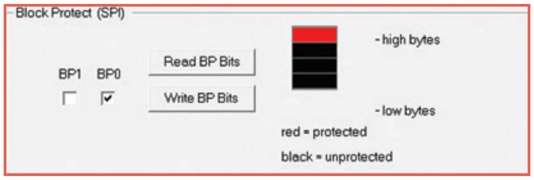
red = protected
black = unprotected

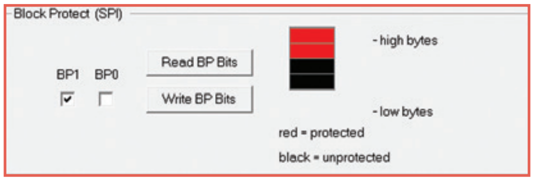
SPI EEPROM and SPI Flash memory devices have their memory space organized into sectors (i.e., blocks or chunks) of uniform size. That is, the size is uniform throughout any given device, but the number of sectors and the size of a sector may vary from one device to another – depending on the total memory capacity of the device. Setting various combinations of the Block Protect bits write-protects various combinations of sectors in the memory space. When all of the Block Protect bits are cleared, none of the sectors is write-protected. Setting all of the available Block Protect bits write-protects the entire memory space of the device. The table to the right shows which sectors are write-protected (for devices of various memory capacities) based on the combination of Block Protect bits that are set/cleared.

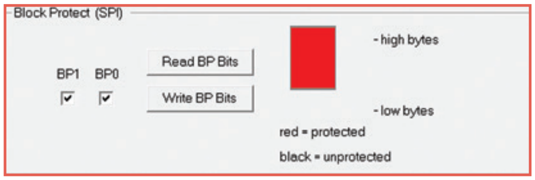


Block Protect Bits			Protected Sectors				
BP2	BP1	BP0	≤ 2-Mbit Device	4-Mbit Device	8-Mbit Device	32-Mbit Device	64-Mbit Device
0	0	0	None	None	None	None	None
0	0	1	3	7	15	63	126 and 127
0	1	0	2 and 3	6 and 7	14 and 15	62 and 63	124 thru 127
0	1	1	All	4 thru 7	12 thru 15	60 thru 63	120 thru 127
1	0	0	n/a	All	All	56 thru 63	112 thru 127
1	0	1	n/a	All	All	48 thru 63	96 thru 127
1	1	0	n/a	All	All	32 thru 63	64 thru 127
1	1	1	n/a	All	All	All	All

Sector numbering begins at zero – corresponding to the lowest memory address. Sectors are protected in a top-down fashion. The top-most sector(s) represents the smallest portion of the total memory that can be protected.





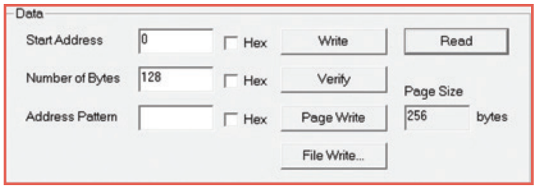


Note that when initially displayed, the **Block Protect (SPI)** group box indicates that neither (none) of the Block Protect bits is set – this indication is not necessarily accurate. Click on the **Read BP Bits** button to update the status of the BP0, BP1 (and BP2) checkboxes so that they will be accurate. A checked box indicates that the corresponding bit is set (i.e., high, one) and an unchecked box indicates that the corresponding bit is cleared (i.e., low, zero). The status of the Block Protect bits can be set manually by checking/clearing the BP0, BP1 (and BP2) checkboxes as desired and then clicking the **Write BP Bits** button.

Note that when a checkbox is checked/cleared, the graphics are updated immediately, but the Block Protect bits remain unaltered until the **Write BP Bits** button is clicked.

Address Pattern

If no data is placed within the **Data Pattern** text box, the application will automatically generate a sequence of increasing data values for use as the data pattern and will re-label the text box **Address Pattern**.

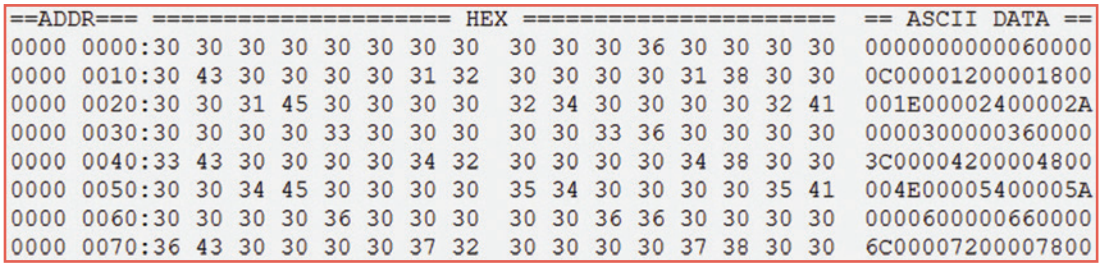

Data

Start Address: 0 □ Hex

Number of Bytes: 128 □ Hex

Address Pattern: □ Hex

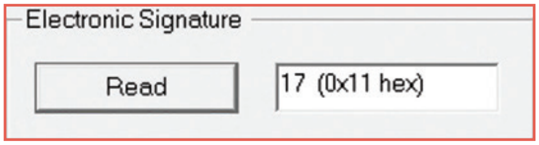
Page Size: 256 bytes


```
==ADDR=== ===================== HEX ======================  == ASCII DATA ==
0000 0000:30 30 30 30 30 30 30 30  30 30 30 36 30 30 30 30  0000000000060000
0000 0010:30 43 30 30 30 30 31 32  30 30 30 30 31 38 30 30  0C00001200001800
0000 0020:30 30 31 45 30 30 30 30  32 34 30 30 30 30 32 41  001E00002400002A
0000 0030:30 30 30 30 33 30 30 30  30 30 33 36 30 30 30 30  0000300000360000
0000 0040:33 43 30 30 30 30 34 32  30 30 30 30 34 38 30 30  3C00004200004800
0000 0050:30 30 34 45 30 30 30 30  35 34 30 30 30 30 35 41  004E00005400005A
0000 0060:30 30 30 30 36 30 30 30  30 30 36 36 30 30 30 30  0000600000660000
0000 0070:36 43 30 30 30 30 37 32  30 30 30 30 37 38 30 30  6C00007200007800
```

Shown here is a sample of the address pattern that the application automatically generates.

Electronic Signature (Serial Flash only)

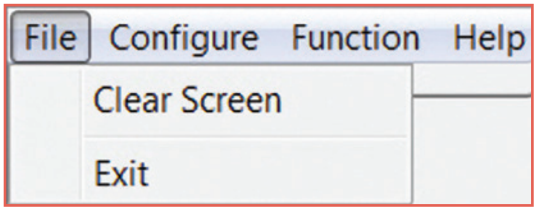
Only Serial Flash devices have electronic signatures. To read the electronic signature of a Serial Flash device, click the **Read** button within the **Electronic Signature** group box. (See the SPI Flash Interface Specification for more information.)


Electronic Signature

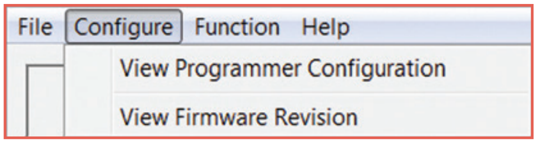
17 (0x11 hex)

Menu Bar

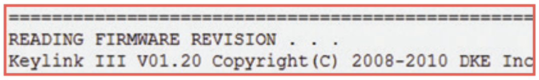
From within the **File** menu, the only two options are clearing the message window (**Clear Screen**) or closing the Sample Cpp App (**Exit**).



From within the **Configure** menu, the 5-character configuration code (which is always visible in the information bar at the bottom of the GUI) can be displayed in the message window by clicking **View Programmer Configuration**.

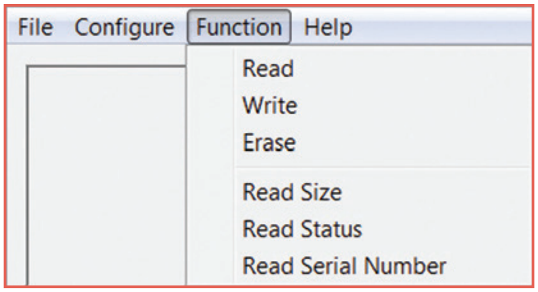


Clicking **View Firmware Revision** will display a message similar to what is shown here.

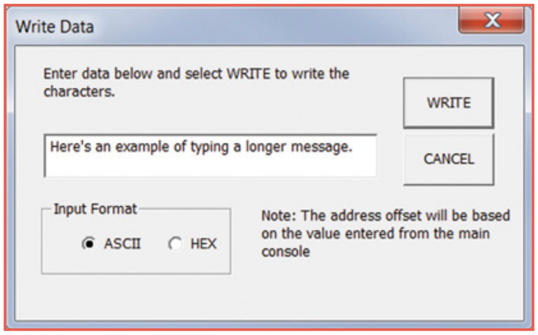


```
=================================================
READING FIRMWARE REVISION . . .
Keylink III V01.20 Copyright(C) 2008-2010 DKE Inc
```

From within the **Function** menu, the usual operations as well as some additional operations are available. **Read Serial Number** is only applicable to IIK/IIT devices. **Write** will permit entering a longer sequence of values, be they ASCII or hexadecimal (see the next screen shot).



This is the window displayed by clicking **Write** within the **Function** menu. An example of a longer ASCII character string is shown.



Enter data below and select WRITE to write the characters.

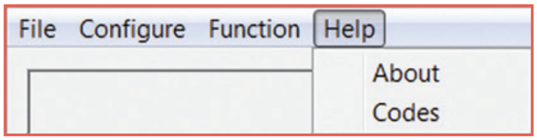
Here's an example of typing a longer message.

Input Format

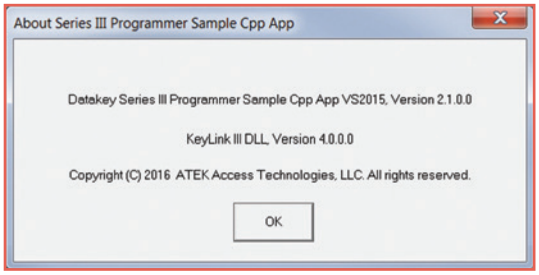
● ASCII ○ HEX

Note: The address offset will be based on the value entered from the main console

For information about the Sample Cpp App, click **About** within the **Help** menu (see the next screen shot). To display a list of all device descriptors and their associated 5-character codes, click **Codes**.



Clicking About in the Help menu will display a message similar to this.



Datakey Series III Programmer Sample Cpp App VS2015, Version 2.1.0.0

KeyLink III DLL, Version 4.0.0.0

Copyright (C) 2016 ATEK Access Technologies, LLC. All rights reserved.

C# and Visual Basic Implementation

To start the application, do the following:

- attach the Series III Programmer to the host computer's USB 2.0 port, then
- double-click the Series III Programmer desktop icon on the host computer.

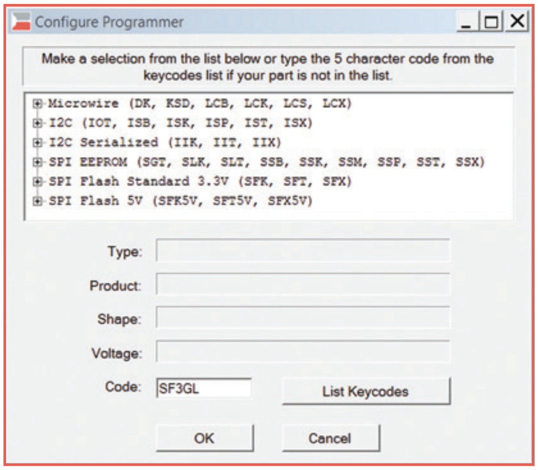
Series III
Programmer

Configuration

Each time the sample application is started, it will prompt the user to configure the attached Series III Programmer for the type of device that is to be expected. This is a crucial step, since it tells the Series III Programmer:

- what protocol is to be used for communication,
- what supply voltage to apply to the device (either 3.3 or 5 volts), and
- the memory capacity of the device.

The sample application cannot query this information from an attached device, it must be told what to expect. The Series III Programmer does remember the type of device for which it was last configured and the corresponding 5-character code is displayed in the **Code** text box of the **Configure Programmer** form.



Configure Programmer

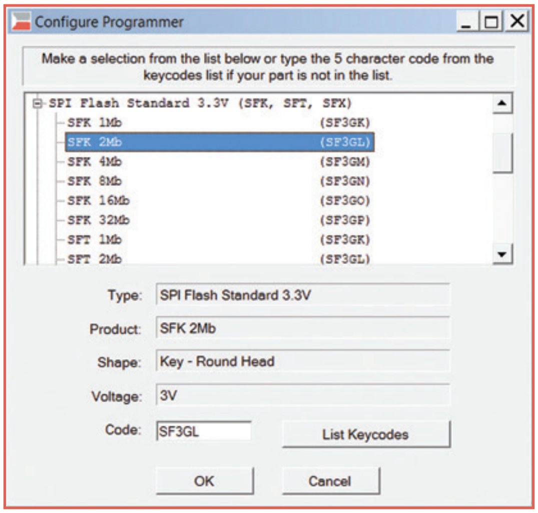
Make a selection from the list below or type the 5 character code from the keycodes list if your part is not in the list.

```
⊞ Microwire (DK, KSD, LCB, LCK, LCS, LCX)
⊞ I2C (IOT, ISB, ISK, ISP, IST, ISX)
⊞ I2C Serialized (IIK, IIT, IIX)
⊞ SPI EEPROM (SGT, SLK, SLT, SSB, SSK, SSM, SSP, SST, SSX)
⊞ SPI Flash Standard 3.3V (SFK, SFT, SFX)
⊞ SPI Flash 5V (SFK5V, SFT5V, SFX5V)
```

Type:
Product:
Shape:
Voltage:
Code: SF3GL

Expand the appropriate protocol family and select the specific device to be used in the Series III Programmer. Note that the same 5-character code may be assigned to more than one particular device in the list. That's because the electrical interface to the devices may be the same, even though their physical shapes are different.

Once the desired configuration has been selected, click the **OK** button. In this example the Series III Programmer is being configured for an SFK 2Mb device.



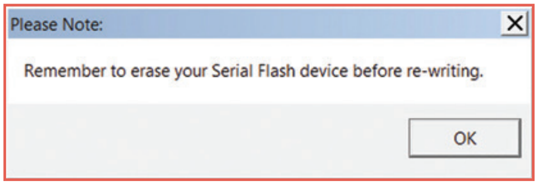
Configure Programmer

Make a selection from the list below or type the 5 character code from the keycodes list if your part is not in the list.

```
⊟ SPI Flash Standard 3.3V (SFK, SFT, SFX)
   SFK 1Mb                        (SF3GK)
   SFK 2Mb                        (SF3GL)
   SFK 4Mb                        (SF3GM)
   SFK 8Mb                        (SF3GN)
   SFK 16Mb                       (SF3GO)
   SFK 32Mb                       (SF3GP)
   SFT 1Mb                        (SF3GK)
   SFT 2Mb                        (SF3GL)
```

Type: SPI Flash Standard 3.3V
Product: SFK 2Mb
Shape: Key - Round Head
Voltage: 3V
Code: SF3GL

Note that if the Series III Programmer is configured for a Serial Flash device, a message will be displayed, reminding the user to manually erase the device before overwriting any existing data. EEPROM devices implement an automatic erase feature whenever new data is written; Serial Flash does not.

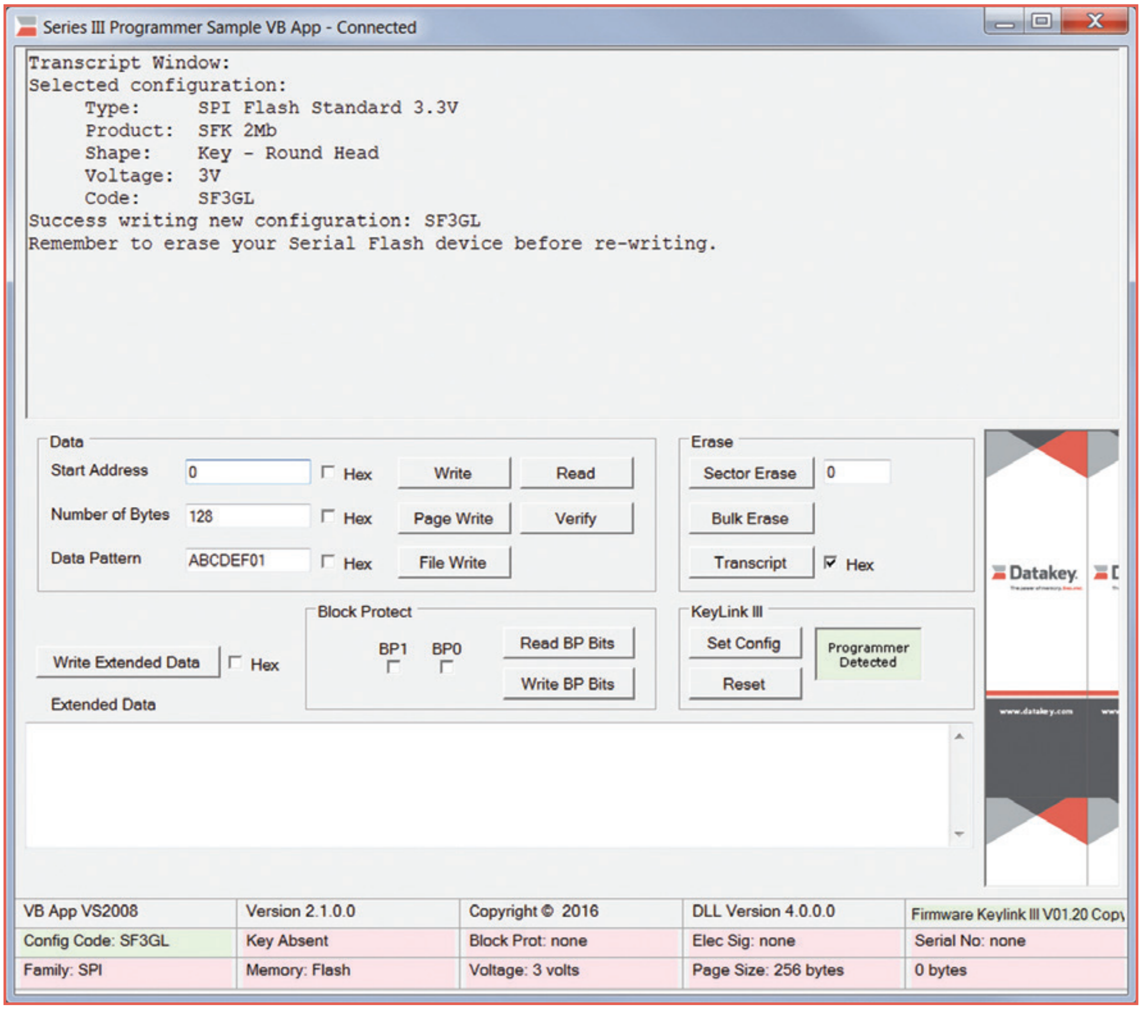


Please Note:

Remember to erase your Serial Flash device before re-writing.

All of the Datakey devices that utilize Serial Flash memory have family names that begin with the letters "SF" (e.g., SFK, SFT, SFX).

Overview



Series III Programmer Sample VB App - Connected

```
Transcript Window:
Selected configuration:
     Type:     SPI Flash Standard 3.3V
     Product:  SFK 2Mb
     Shape:    Key - Round Head
     Voltage:  3V
     Code:     SF3GL
Success writing new configuration: SF3GL
Remember to erase your Serial Flash device before re-writing.
```

Data
Start Address: 0 □ Hex
Number of Bytes: 128 □ Hex
Data Pattern: ABCDEF01 □ Hex

Erase
Sector Erase: 0
Transcript ☑ Hex

Block Protect
BP1 □ BP0 □

Write Extended Data □ Hex
Extended Data

VB App VS2008	Version 2.1.0.0	Copyright © 2016	DLL Version 4.0.0.0	Firmware Keylink III V01.20 Copy
Config Code: SF3GL	Key Absent	Block Prot: none	Elec Sig: none	Serial No: none
Family: SPI	Memory: Flash	Voltage: 3 volts	Page Size: 256 bytes	0 bytes

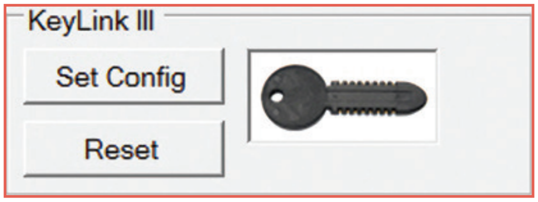
The **Sample VB App** pictured above has:

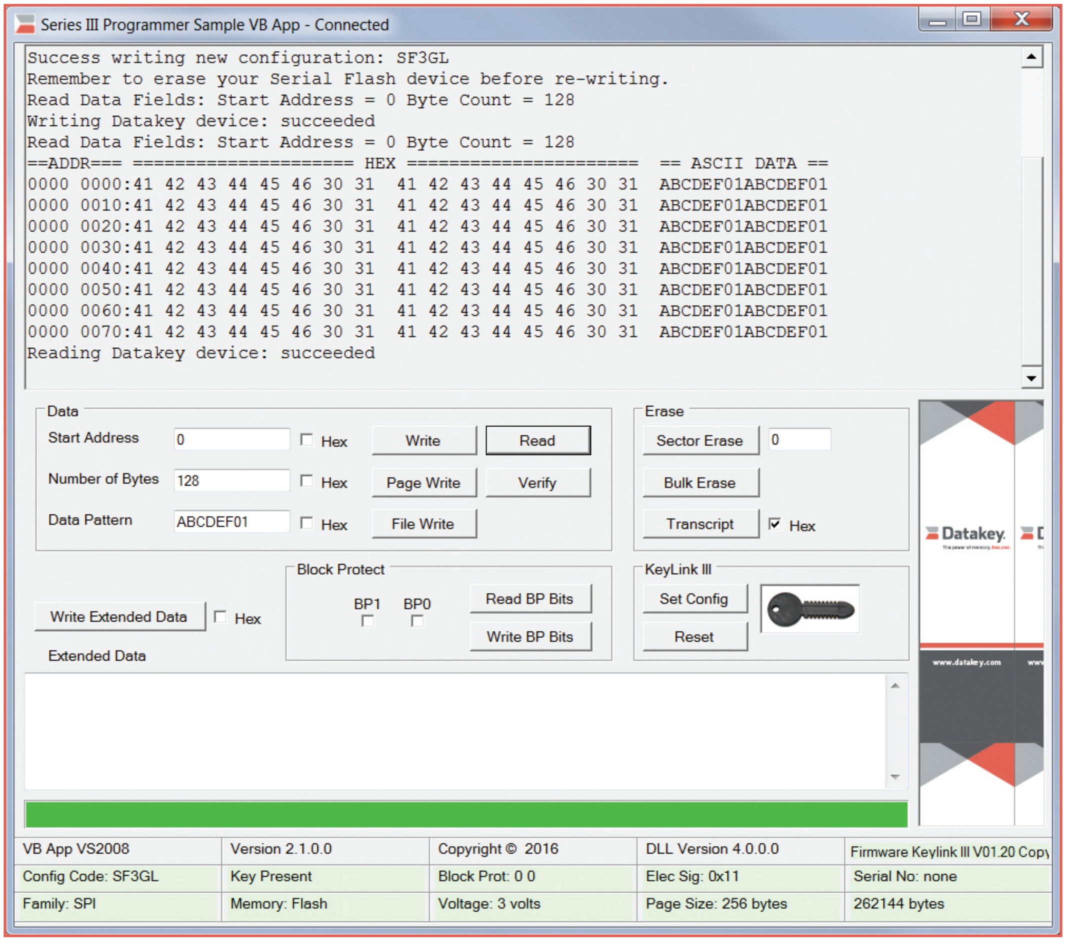
- a **Transcript Window** at the top that shows the results of operations recently performed and the general status of the application,
- a **Data** group box where address limits and data patterns can be specified and where read, write, and compare operations can be performed,
- an **Erase** group box where individual sectors or the entire device can be targeted for erasure,
- a **Block Protect** group box where the block protect bits of an SPI device can be read and set/cleared,
- a **KeyLink III** group box where the Series III Programmer can be reconfigured or reset and where the on-line/off-line status of the Series III Programmer can easily be determined,
- an **Extended Data** window where larger amounts of data can be entered and written to the device, and
- a status grid where information about the software, firmware, configuration, and hardware status are displayed – including an electronic signature for Serial Flash devices (see the SPI Flash Interface Specification).

Note that not all controls are available or appropriate for all types of devices.

Insertion

When a key is inserted into the receptacle, an image of a key will be displayed within the **KeyLink III** group box. (Note that for an actual key type of device, the key must be inserted into the receptacle and turned a quarter turn clockwise before the image of a key will appear.) Behind the scenes, the application is polling the status of the LOFO (last on, first off) signal and when it goes low, the image is displayed.





```
Success writing new configuration: SF3GL
Remember to erase your Serial Flash device before re-writing.
Read Data Fields: Start Address = 0 Byte Count = 128
Writing Datakey device: succeeded
Read Data Fields: Start Address = 0 Byte Count = 128
==ADDR=== ===================== HEX =====================  == ASCII DATA ==
0000 0000:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0010:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0020:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0030:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0040:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0050:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0060:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0070:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
Reading Datakey device: succeeded
```

Data
Start Address: 0 ☐ Hex
Number of Bytes: 128 ☐ Hex
Data Pattern: ABCDEF01 ☐ Hex

Erase
Sector Erase: 0
☑ Hex

Block Protect
BP1 ☐ BP0 ☐

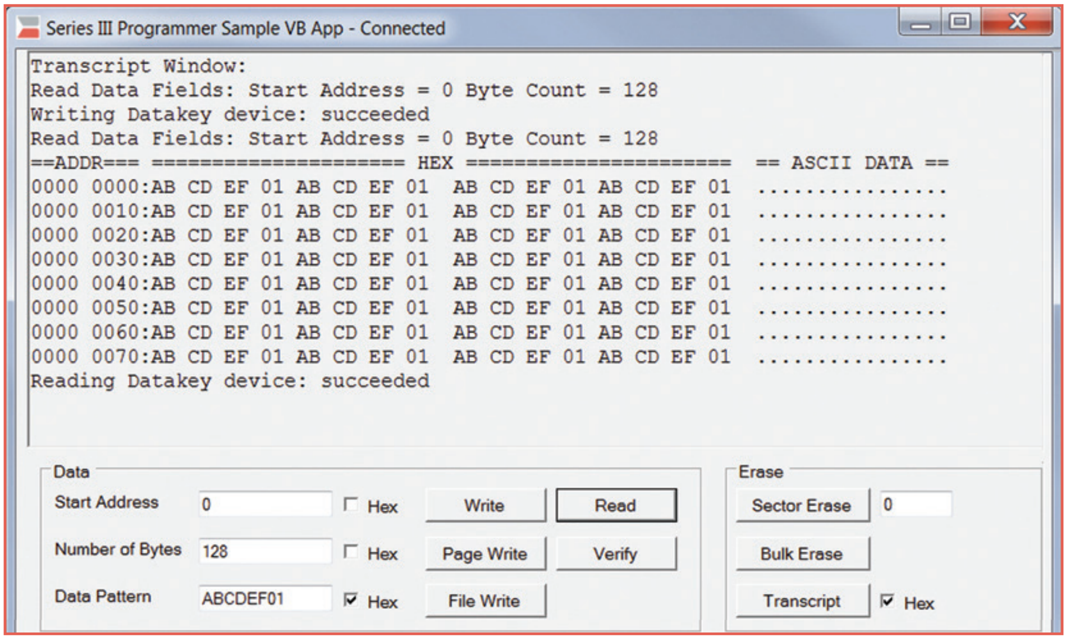
☐ Hex
Extended Data

VB App VS2008	Version 2.1.0.0	Copyright © 2016	DLL Version 4.0.0.0	Firmware Keylink III V01.20 Copy
Config Code: SF3GL	Key Present	Block Prot: 0 0	Elec Sig: 0x11	Serial No: none
Family: SPI	Memory: Flash	Voltage: 3 volts	Page Size: 256 bytes	262144 bytes

Note that in addition to the key image, a green status bar is also displayed. Additionally, none of the status grid blocks are pink – most of them are now green.

Utilizing the default values in the **Data** group box and clicking the **Write** button and then the **Read** button, a repeating data pattern is first written to and then read back from the inserted key. As per the default values, writing/reading begins at address zero and continues sequentially for 128 bytes.

Because none of the **Hex** checkboxes in the **Data** group box were checked, the specified **Start Address** and **Number of Bytes** are interpreted as decimal values and the **Data Pattern** is converted to ASCII data.



Series III Programmer Sample VB App - Connected

```
Transcript Window:
Read Data Fields: Start Address = 0 Byte Count = 128
Writing Datakey device: succeeded
Read Data Fields: Start Address = 0 Byte Count = 128
==ADDR=== ===================== HEX =====================  == ASCII DATA ==
0000 0000:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0010:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0020:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0030:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0040:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0050:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0060:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
0000 0070:AB CD EF 01 AB CD EF 01  AB CD EF 01 AB CD EF 01  ................
Reading Datakey device: succeeded
```

Data

Start Address: 0 ☐ Hex

Number of Bytes: 128 ☐ Hex

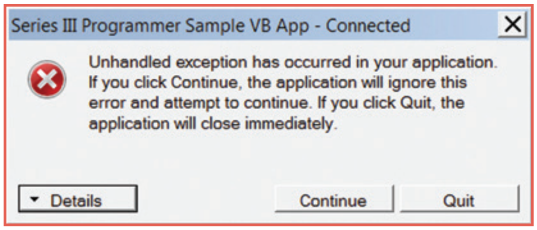
Data Pattern: ABCDEF01 ☑ Hex

Erase

Sector Erase: 0

☑ Hex

Utilizing the same default values in the **Data** group box, but checking the **Hex** checkbox associated with **Data Pattern** and then clicking the **Write** button and then the **Read** button, would display the data shown above. Note that in this case the **Data Pattern** would be interpreted as a sequence of hexadecimal values and written directly to the key as binary, without being converted to its equivalent ASCII codes.



Series III Programmer Sample VB App - Connected

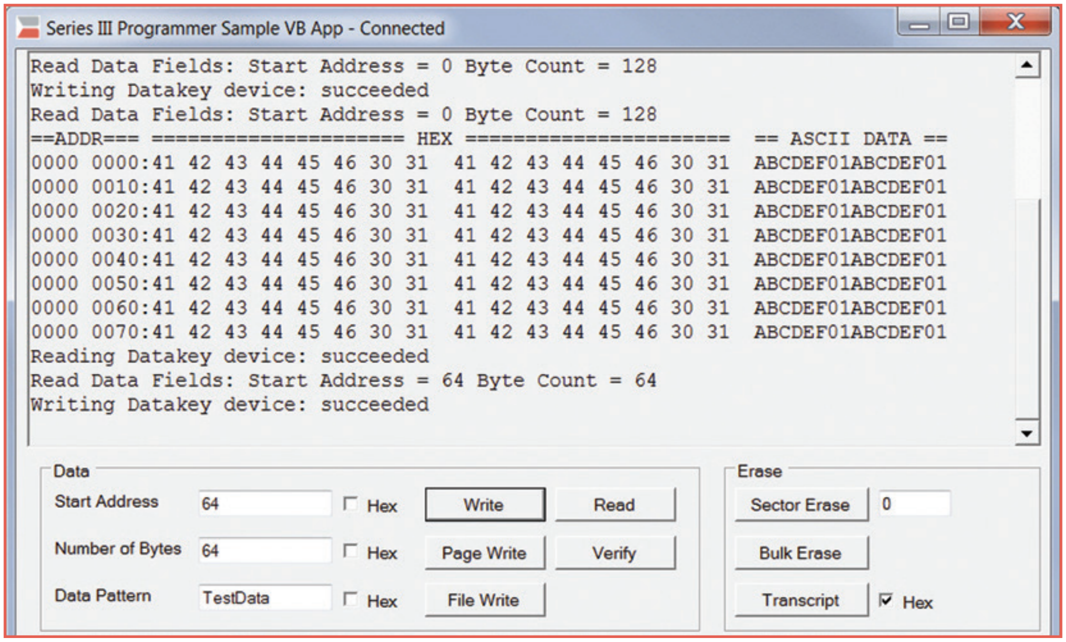
Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately.

CAUTION: Attempting to send "TestData" as the **Data Pattern** and telling the application to interpret it as hexadecimal characters will throw an unhandled exception. If this happens, click the **Continue** button.

If, within the **Data** group box, the **Page Write** button is clicked instead of the **Write** button, the **Number of Bytes** to be written must be less than or equal to the **Page Size** of the device. The **Page Size** is displayed in the bottom row of the status grid.

If, within the **Data** group box, the **File Write** button is clicked instead of the **Write** button, the application will prompt for and allow browsing to the file to be written.

Manual Erasure (Serial Flash only)



```
Read Data Fields: Start Address = 0 Byte Count = 128
Writing Datakey device: succeeded
Read Data Fields: Start Address = 0 Byte Count = 128
==ADDR=== ===================== HEX ======================  == ASCII DATA ==
0000 0000:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0010:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0020:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0030:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0040:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0050:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0060:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0070:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
Reading Datakey device: succeeded
Read Data Fields: Start Address = 64 Byte Count = 64
Writing Datakey device: succeeded
```

Start Address: 64 □ Hex

Number of Bytes: 64 □ Hex

Data Pattern: TestData □ Hex

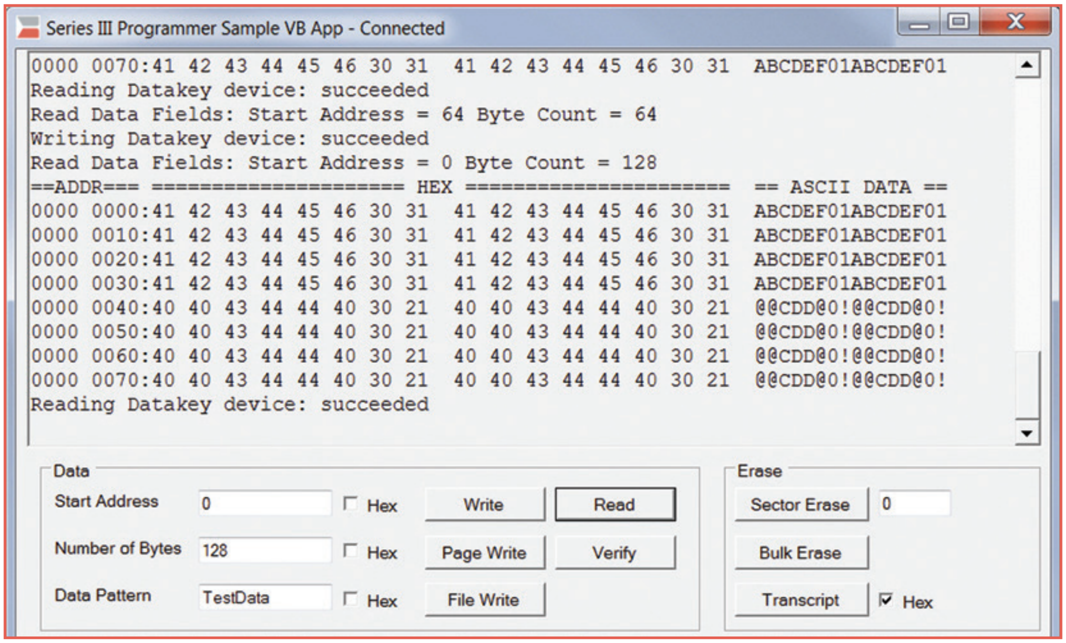
Sector Erase: 0

☑ Hex

Here's an example of the type of behavior that can be expected if a Serial Flash device is not manually erased before being rewritten. It is assumed that the first example of writing 128 bytes of data to the key was, in fact, performed.

Now (without erasing the initial data) perform the following additional operations:

- enter the values shown above in the **Data** group box for **Start Address, Number of Bytes**, and **Data Pattern**,
- click the **Write** button,



```
0000 0070:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
Reading Datakey device: succeeded
Read Data Fields: Start Address = 64 Byte Count = 64
Writing Datakey device: succeeded
Read Data Fields: Start Address = 0 Byte Count = 128
==ADDR=== ===================== HEX ======================  == ASCII DATA ==
0000 0000:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0010:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0020:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0030:41 42 43 44 45 46 30 31  41 42 43 44 45 46 30 31  ABCDEF01ABCDEF01
0000 0040:40 40 43 44 44 40 30 21  40 40 43 44 44 40 30 21  @@CDD@0!@@CDD@0!
0000 0050:40 40 43 44 44 40 30 21  40 40 43 44 44 40 30 21  @@CDD@0!@@CDD@0!
0000 0060:40 40 43 44 44 40 30 21  40 40 43 44 44 40 30 21  @@CDD@0!@@CDD@0!
0000 0070:40 40 43 44 44 40 30 21  40 40 43 44 44 40 30 21  @@CDD@0!@@CDD@0!
Reading Datakey device: succeeded
```

Data
Start Address: 0 □ Hex
Number of Bytes: 128 □ Hex
Data Pattern: TestData □ Hex

Erase
Sector Erase: 0

☑ Hex

- enter the values shown above in the **Data** group box for **Start Address** and **Number of Bytes**, and
- click the **Read** button.

Now look at the 128 bytes of data that was read from the device. The first 64 bytes still exhibit the same repeating pattern of data that was written to them initially, but the final 64 bytes show neither the initial nor the subsequent data pattern. What's going on here?

The answer is that the following rules are obeyed when overwriting data in a Serial Flash device:

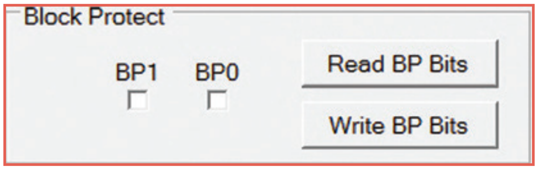
- binary ones in the initial data can remain as binary ones in the subsequent data,
- binary ones in the initial data can be demoted to binary zeroes in the subsequent data,
- binary zeroes in the initial data can remain as binary zeroes in the subsequent data, but
- binary zeroes in the initial data cannot be promoted to binary ones in the subsequent data.

More often than not, what is wanted is to manually erase the previous data before writing new data to any given memory location. That is the only way to promote binary zeroes in the initial data back to binary ones. When any byte is erased, all its bits go to binary ones.

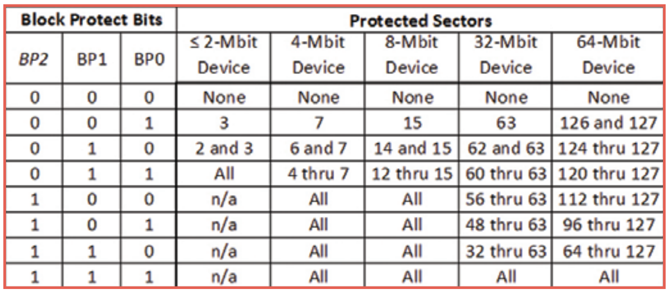
The scope of an erasure can be limited to a single sector at a time or applied to the entire memory space simultaneously. This can be done using the controls in the **Erase** group box. To erase a single sector, specify the sector number and then click the **Sector Erase** button. To erase the entire memory space, click the **Bulk Erase** button.

Block Protect (SPI only)

Sections of an SPI EEPROM/Flash memory space can be protected from writing and erasure by utilizing the Block Protect bits, BP0 and BP1. Depending on the memory capacity of the device, there may even be a third Block Protect bit, BP2. Only SPI memory offers Block Protect bits.



SPI EEPROM and SPI Flash memory devices have their memory space organized into sectors (i.e., blocks or chunks) of uniform size. That is, the size is uniform throughout any given device, but the number of sectors and the size of a sector may vary from one device to another – depending on the total memory capacity of the device. Setting various combinations of the Block Protect bits write-protects various combinations of sectors in the memory space. When all of the Block Protect bits are cleared, none of the sectors is write-protected. Setting all of the available Block Protect bits write-protects the entire memory space of the device. The table to the right shows which sectors are write-protected (for devices of various memory capacities) based on the combination of Block Protect bits that are set/cleared.



Block Protect Bits			Protected Sectors				
BP2	BP1	BP0	≤ 2-Mbit Device	4-Mbit Device	8-Mbit Device	32-Mbit Device	64-Mbit Device
0	0	0	None	None	None	None	None
0	0	1	3	7	15	63	126 and 127
0	1	0	2 and 3	6 and 7	14 and 15	62 and 63	124 thru 127
0	1	1	All	4 thru 7	12 thru 15	60 thru 63	120 thru 127
1	0	0	n/a	All	All	56 thru 63	112 thru 127
1	0	1	n/a	All	All	48 thru 63	96 thru 127
1	1	0	n/a	All	All	32 thru 63	64 thru 127
1	1	1	n/a	All	All	All	All

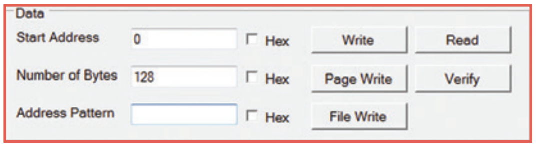
Sector numbering begins at zero – corresponding to the lowest memory address. Sectors are protected in a top-down fashion. The top-most sector(s) represents the smallest portion of the total memory that can be protected.

Note that when initially displayed, the **Block Protect** group box indicates that neither (none) of the Block Protect bits is set – this indication is not necessarily accurate. Click on the **Read BP Bits** button to update the status of the BP0, BP1 (and BP2) checkboxes so that they will be accurate. A checked box indicates that the corresponding bit is set (i.e., high, one) and an unchecked box indicates that the corresponding bit is cleared (i.e., low, zero). The status of the Block Protect bits can be set manually by checking/clearing the BP0, BP1 (and BP2) checkboxes as desired and then clicking the **Write BP Bits** button.

Note that when a checkbox is checked/cleared, the graphics are updated immediately, but the Block Protect bits remain unaltered until the **Write BP Bits** button is clicked.

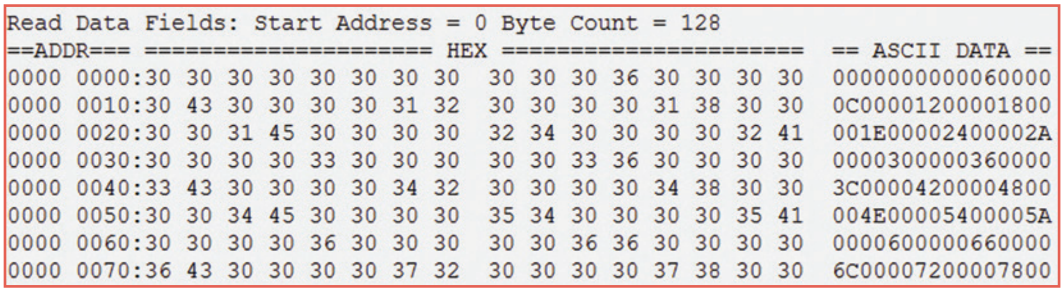
Address Pattern

If no data is placed within the **Data Pattern** text box, the application will automatically generate a sequence of increasing data values for use as the data pattern and will re-label the text box **Address Pattern**.



Start Address 0 □ Hex
Number of Bytes 128 □ Hex
Address Pattern □ Hex

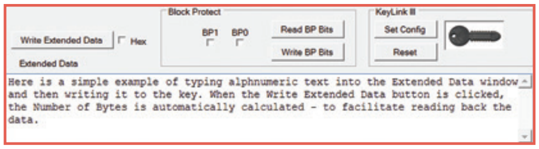
Shown here is a sample of the address pattern that the application automatically generates.



```
Read Data Fields: Start Address = 0 Byte Count = 128
==ADDR=== ===================== HEX ======================  == ASCII DATA ==
0000 0000:30 30 30 30 30 30 30 30  30 30 30 36 30 30 30 30  0000000000060000
0000 0010:30 43 30 30 30 30 31 32  30 30 30 30 31 38 30 30  0C00001200001800
0000 0020:30 30 31 45 30 30 30 30  32 34 30 30 30 30 32 41  001E00002400002A
0000 0030:30 30 30 30 33 30 30 30  30 30 33 36 30 30 30 30  0000300000360000
0000 0040:33 43 30 30 30 30 34 32  30 30 30 30 34 38 30 30  3C00004200004800
0000 0050:30 30 34 45 30 30 30 30  35 34 30 30 30 30 35 41  004E00005400005A
0000 0060:30 30 30 30 36 30 30 30  30 30 36 36 30 30 30 30  0000600000660000
0000 0070:36 43 30 30 30 30 37 32  30 30 30 30 37 38 30 30  6C00007200007800
```

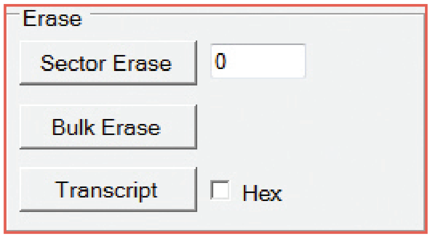
Extended Data

The application's **Extended Data** window allows manual entry of larger amounts of data or a simple copy-and-paste operation. Click the **Write Extended Data** button to copy the window's contents to the key. The **Number of Bytes** written will automatically be calculated.

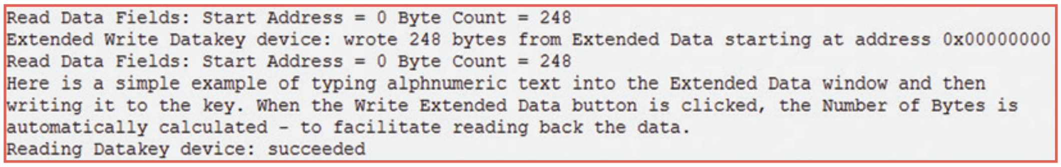


```
Here is a simple example of typing alphnumeric text into the Extended Data window
and then writing it to the key. When the Write Extended Data button is clicked,
the Number of Bytes is automatically calculated - to facilitate reading back the
data.
```

In order to see a plain text display of the data read from the key, clear the **Hex** checkbox in the **Erase** group box. That will prevent the application from displaying the address labels and hexadecimal values associated with the data it displays in the **Transcript Window**.



Sector Erase 0
□ Hex



```
Read Data Fields: Start Address = 0 Byte Count = 248
Extended Write Datakey device: wrote 248 bytes from Extended Data starting at address 0x00000000
Read Data Fields: Start Address = 0 Byte Count = 248
Here is a simple example of typing alphnumeric text into the Extended Data window and then
writing it to the key. When the Write Extended Data button is clicked, the Number of Bytes is
automatically calculated - to facilitate reading back the data.
Reading Datakey device: succeeded
```

Above is an example of what the plain text will look like in the Transcript Window.Click the **Read** button in the Data group box to display the text. To clear text from the **Transcript Window**, click the **Transcript** button in the **Erase** group box.

Acronyms/Abbreviations

ASCII	American Standard Code for Information Interchange
CD	Compact Disc
EEPROM	Electrically Erasable Programmable Read-Only Memory
Gbits/bytes	Gigabits/bytes – may indicate a multiplier of 10^9 (1,000,000,000) or 2^{30} (1,073,741,824) bits/bytes
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
Kbits/bytes	Kilobits/bytes – may indicate a multiplier of 10^3 (1,000) or 2^{10} (1,024) bits/bytes
LED	Light-Emitting Diode
Mbits/bytes	Megabits/bytes – may indicate a multiplier of 10^6 (1,000,000) or 2^{20} (1,048,576) bits/bytes
SPI	Serial Peripheral Interface
USB	Universal Serial Bus

223-0100-000 Rev. E 7/21

Microsoft is a registered trademark of Microsoft Corporation. Windows is a trademarks of Microsoft Corporation. All other brand names and product names used in this manual are trademarks, registered trademarks, or trade names of their respective holders.

ATEK Access Technologies
10025 Valley View Road, Ste. 190
Eden Prairie, MN 55344 U.S.A.

PH: 1.800.523.6996
FAX: 1.800.589.3705
+1.218.829.9797

www.atekaccess.com

ATEK Access Technologies
Access the power of technology.