PSM20i Bar Code Scanner Attachment

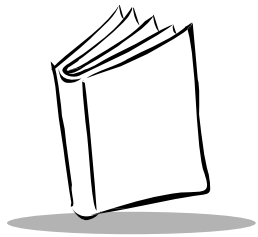
Programmer's Guide

PSM20i Bar Code Scanner Attachment Programmer's Guide

72E-59169-01
Revision A
December 2002

symbol®

Symbol Technologies, Inc.
One Symbol Plaza
Holtsville, New York 11742-1300
http://www.symbol.com



Contents

About This Guide

Introduction v
About the SDK vi
Java 2 Micro Edition (J2ME) vi
Integrated Development Environments and SDKs vi
Related Documents vi
Notational Conventions vii
Service Information vii
    Symbol Support Center viii

Chapter 1. Getting Started

Introduction 1-1
Required Item Checklist 1-1
Getting iDEN Java Application Loader and Handset Provisioning 1-2
Installing the PSM20i Bar Code Scanner Attachment SDK 1-3

Chapter 2. Function Definitions

Introduction 2-1
Supported Barcode Type 2-4
Basic Function Commands 2-5
    disable 2-5
    enable 2-6
    getCodeType 2-7
    isEnabled 2-9
    resetScanner 2-10
    setDataTransFormat 2-11
    setLaserOnTime 2-12
    setLinearSecurity 2-13
    setPrefix 2-15

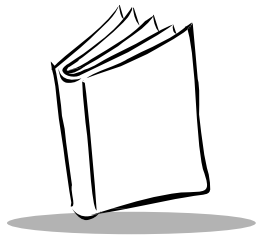
setSuffix1 . . . 2-16
setSuffix2 . . . 2-17
startDecode . . . 2-18
transmitCodeId . . . 2-19
Decode Configuration Commands . . . 2-20
booklandEAN . . . 2-20
codabar . . . 2-21
code128 . . . 2-23
code39 . . . 2-24
code39Trioptic . . . 2-26
code93 . . . 2-27
discrete2of5 . . . 2-28
ean128 . . . 2-30
ean13 . . . 2-31
ean8 . . . 2-32
interleaved2of5 . . . 2-33
isbt128 . . . 2-35
msiPlessey . . . 2-36
upcA . . . 2-38
upcE . . . 2-39
upcE1 . . . 2-40
upcEANCouponCode . . . 2-41
upcEANSecurity . . . 2-42
upcEANSupplemental . . . 2-43
upcEANSupplRedundancy . . . 2-44
Miscellaneous . . . 2-45
getSDKVersion . . . 2-45

Appendix A. Programming Reference

Symbol Code Identifiers . . . A-1
AIM Code Identifiers . . . A-2

Appendix B. Sample Program

Index



About This Guide

Introduction

The *PSM20i Bar Code Scanner Attachment Programmer's Guide* provides the Application Programming Interface (API) definition for the Bar Code Scanner Attachment Software Development Kit (SDK). The PSM20i Bar Code Scanner Attachment adds value to Motorola iDEN phones by adding fast, efficient data capture to the J2ME applications that run on these phones.

Note: *To guarantee an accurate decode, ensure that the phone is not transmitting while the scanner is attempting to decode data.*

The documentation and demo programs included in this package provide developers with resources necessary to create data capture applications.

This guide provides information on developing applications utilizing the PSM20i Bar Code Scanner Attachment for installation on the i88, i85s, i58, i55sr & i50sx multi-communication devices. It is targeted at application developers for these products.

This guide does not provide information on developing and packaging applications for installation on the i88, i85s, i58, i55sr & i50sx multi-communication devices or the Java 2 Micro Edition environment. This information can be obtained at: http://www.motorola.com/idendev.

About the SDK

The Symbol Attachment Scanner SDK provides API calls used to enable and configure the Symbol Technologies scan engine used with the Motorola iDEN phones. The SDK was developed using the J2ME Wireless Toolkit 1.0.4 provided by Sun Microsystems, Inc.

Java 2 Micro Edition (J2ME)

The Java 2 Micro Edition platform, or J2ME, is a highly optimized, scaled down version of the Java 2 Platform designed for deployment on small devices such as cell phones and pagers. For more information on J2ME, see http://java.sun.com/j2me.

Integrated Development Environments and SDKs

Various tools are available to emulate and develop MIDP compliant applications. Many of the tools, such as the J2ME Wireless Toolkit from Sun Microsystems (http://java.sun.com/products/j2mewtoolkit/download.html), include a compiler, MIDP API documentation, a byte code verifier, and a customizable emulation environment. In addition to simple emulation, tools such as MetroWerks CodeWarrior (http://www.metrowerks.com) integrate the MIDP API and emulation into a full development and debug environment. Like CodeWarrior, Forte for Java from Sun Microsystems (http://www.sun.com/forte/ffj/index.html) also includes a full IDE and debug environment for J2ME development.

Related Documents

- *PSM20i Bar Code Scanner Attachment Owner's Guide*, p/n 72-55997-xx
- *Motorola i85s & i50sx Multi-Communication Device Developers' Guide*, p/n NTN9880A - this guide provides information on developing and packaging J2ME compliant applications for the Motorola multi-communication devices. It also includes step-by-step procedures for setting up the debug environment on the devices.

Notational Conventions

The following conventions are used in this document:

- *Italics* are used to highlight specific items in the general text, and to identify chapters and sections in this and related documents.
- Bullets (•) indicate:
  - action items
  - lists of alternatives
  - lists of required steps that are not necessarily sequential
- Sequential lists (e.g., those that describe step-by-step procedures) appear as numbered lists.

Service Information

If you have a problem with your equipment, contact the *Symbol Support Center* for your region. See page viii for contact information. Before calling, have the model number, serial number, and several of your bar code symbols at hand.

Call the Support Center from a phone near the scanning equipment so that the service person can try to talk you through your problem. If the equipment is found to be working properly and the problem is symbol readability, the Support Center will request samples of your bar codes for analysis at our plant.

If your problem cannot be solved over the phone, you may need to return your equipment for servicing. If that is necessary, you will be given specific directions.

Note: *Symbol Technologies is not responsible for any damages incurred during shipment if the approved shipping container is not used. Shipping the units improperly can possibly void the warranty. If the original shipping container was not kept, contact Symbol to have another sent to you.*

Symbol Support Center

For service information, warranty information or technical assistance contact or call the Symbol Support Center in:

United States [1]
Symbol Technologies, Inc.
One Symbol Plaza
Holtsville, New York 11742-1300
1-800-653-5350

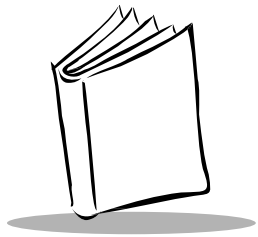
Canada
Symbol Technologies Canada, Inc.
2540 Matheson Boulevard East
Mississauga, Ontario, Canada L4W 4Z2
905-629-7226

Latin America Sales Support
7900 Glades Road
Suite 340
Boca Raton, Florida 33434 USA
1-800-347-0178 (Inside United States)
+1-561-483-1275 (Outside United States)

[1]Customer support is available 24 hours a day, 7 days a week.

If you purchased your Symbol product from a Symbol Business Partner, contact that Business Partner for service.

For the latest version of this guide go to:http://www.symbol.com/manuals.



Chapter 1
Getting Started

Introduction

This chapter provides the information you need to start developing with your PSM20i Bar Code Scanner Attachment. Ensure that you complete the following steps:

1. Obtain all the items in the Required Items Checklist.
2. Obtain iDEN Java Application Loader and handset provisioning.
3. Install the PSM20i Bar Code Scanner Attachment SDK.
4. Customize the decoding, system, and configuration parameters of the device. Refer to Chapter 2, *Function Definitions* for full descriptions of all available parameters.

Required Item Checklist

- Motorola iDEN phone (i88, i85s, i58, i55sr or i50sx)
- PC Data Cable (p/n NKN6544)
- Motorola iDEN Java Application Loader utility
- Sun Java 2 Platform Standard Edition
- Sun Java 2 Micro Edition Wireless Toolkit.

Getting iDEN Java Application Loader and Handset Provisioning

Note: *The following process was put in place by most of the current iDEN operators, but not all of them. You must complete ALL steps in order to be authorized by your operator to load you own applications.*

1. Obtain a J2ME-enabled handset (i85s, i50sx, or i55sr) from your iDEN operator.
   a. Nextel - For the United States, the operator mandates, you register for a develop rate plan at http://developer.nextel.com or your request for required tools may be rejected.
   b. Telus - Please contact your local representative.
   c. Southern LINC - Please contact your local representative.
2. Register with Motorola iDEN Developer Program via http://www.motorola.com/idendev (recommend using Internet Explorer 5.5 and above to access this website). You will be issued a unique iDEN Developer Program user ID and password which are required to access the developer tools.
3. Contact your operator requesting a developer rate plan (optional for operator) and data usage plan (optional for operator). Please include your Name, IMEI number, telephone number, and SIM ID for your phone along with your iDEN Developer Program user ID.
4. Obtain a Motorola PC data cable (p/n NKN6544) which allows you to connect your phone to your PC for loading applications, available through your iDEN operator and http://www.idenstore.com.
5. Download and install the iDEN Java Application Loader. You will need to register via the application registration process to use this tool.
6. Once you receive confirmation that you have been approved to use the iDEN Java Application Loader, you may run it from your PC, using your user ID and password.

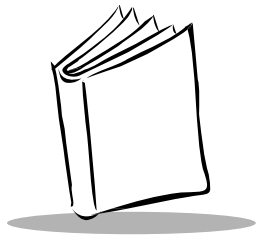
Note: *The iDEN Java Application Loader is intended for testing purposes only and not for commercial use, there is a restriction of up to 5 phones that can be used for loading applications. Once you reach this device limit, your developer access will be automatically deactivated.*

Installing the PSM20i Bar Code Scanner Attachment SDK

1. Download and install Sun Java 2, Platform Standard Edition (J2SE) from http://www.sun.com.
2. Download and install Sun Java 2 Micro Edition (J2ME) Wireless Toolkit from http://www.sun.com.
3. Place AScanner.jar into J2SE jre\lib\ext directory and J2ME Wireless Toolkit apps\lib directory.

In order to access the BarCodeReader class include the following import statement in the source file(s) of your midlet.

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
```



Chapter 2
Function Definitions

Introduction

This chapter details all of the available function calls and their descriptions (see Table 2-1 for a complete list). All CSP functions listed are broken down by functional group.

Table 2-1. Available Function Calls

Function Name	Description	Page
Basic Function Commands		
disable	Terminates a connection with the scanner.	2-5
enable	Establishes a connection with the scanner.	2-6
getCodeType	Returns the code type of barcode scanned.	2-7
isEnabled	Determines if the scanner is currently enabled and ready for use.	2-9
resetScanner	Reconfigures the scanner to the factory default settings.	2-10
setDataTransFormat	Determines the format of the decoded data which is returned to the application.	2-11
setLaserOnTime	Sets the amount of time the decoder will try to decode a barcode.	2-12
setLinearSecurity	Determines which linear code types and lengths must be read, depending on the security level.	2-13
setPrefix	Set the hex value of the character to be used as the prefix to any decoded data.	2-15

Table 2-1. Available Function Calls

Function Name	Description	Page
setSuffix1	Set the hex value of the character to be used as the first suffix to any decoded data.	2-16
setSuffix2	Set the hex value of the character to be used as the second suffix to any decoded data.	2-17
startDecode	Returns a string containing the decoded text.	2-18
transmitCodeId	Returns an ID character along with the decoded barcode data.	2-19
Decode Configuration Commands		
booklandEAN	Enable or disable decoding of Bookland-EAN barcodes and set label options.	2-20
codabar	Enable or disables decoding of Codabar bar codes and options.	2-21
code128	Enable or disables decoding of USS-128 bar codes.	2-23
code39	Enable or disables decoding of Code 39 bar codes and options.	2-24
code39Trioptic	Enable or disables decoding of Trioptic Code 39 bar codes.	2-26
code93	Enable or disables decoding of Code 93 bar codes and lengths.	2-27
discrete2of5	Enable or disables decoding of Discrete 2 Of 5 bar codes and lengths.	2-28
ean128	Enable or disables decoding of UCC/EAN-128 bar codes.	2-30
ean13	Enable or disable decoding of EAN-13 barcodes and set label options.	2-31
ean8	Enable or disable decoding of EAN-8 barcodes and set label options.	2-32
interleaved2of5	Enable or disables decoding of Interleaved 2 Of 5 bar codes and options.	2-33
isbt128	Enable or disables decoding of ISBT-128 bar codes.	2-35
msiPlessey	Enable or disables decoding of MSI Plessey bar codes and options.	2-36

Table 2-1. Available Function Calls

Function Name	Description	Page
upcA	Enable or disable decoding of UPC-A barcodes and set label options.	2-38
upcE	Enable or disable decoding of UPC-E barcodes and set label options.	2-39
upcE1	Enable or disable decoding of UPC-E1 barcodes and set label options.	2-40
upcEANCouponCode	Enable or disables decoding of UPC-A, UPC-A with 2 supplemental characters, UPC-A with 5 supplemental characters, and UPC-A/EAN-128 bar codes.	2-41
upcEANSecurity	Set the level of security for decoding UPC/EAN bar codes.	2-42
upcEANSupplemental	Set supplemental mode for UPC/EAN labels.	2-43
upcEANSupplRedundancy	Adjust the number of times a symbol without supplementals will be decoded before transmission.	2-44
Miscellaneous		
getSDKVersion	Returns the version of the SDK that was used in creating the midlet.	2-45

Please note, the returned data from each function, if false, indicates an error condition in executing the function, *except* for errors reported from Windows file operations. A true indicates success and/or returned data as detailed by each function description.

Supported Barcode Type

The following bar codes types can be scanned with the PSM20i Bar Code Scanner Attachment:

Barcode Type	Default Setting on Scanner
UPC-A	Enable
UPC-E	Enable
UPC-E1	Disable
EAN-8	Enable
EAN-13	Enable
EAN-128	Enable
Code 39	Enable
Trioptic Code 39	Disable
Code 93	Disable
Code 128	Enable
ISBT-128	Enable
Bookland EAN	Disable
Interleave 2 of 5	Enable
Discrete 2 of 5	Disable
MSI Plessey	Disable
Codabar	Disable
Coupon Code	Disable

Basic Function Commands

disable

Class

BarCodeReader

Description

Terminates a connection with the scanner.

Invocation

boolean disable()

Return Value

true	if connection to scanner was successfully disabled.
false	if connection to scanner failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.disable();
```

enable

Class

BarCodeReader

Description

Establishes a connection with the scanner. This function must be successfully executed prior to calling any other function calls.

Note: *To guarantee an accurate decode, ensure that the enable function is not called while the phone is transmitting.*

Invocation

boolean enable()

Return Value

true	if connection to scanner was successfully enabled.
false	if connection to scanner failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.enable();
```

getCodeType

Class

BarCodeReader

Description

Returns the code type of barcode scanned.

Invocation

byte getCodeType()

Return Value

Returns a byte representing the type of barcode scanned. Below is a list of associated hex values for each supported code type:

Barcode Type	Hex Value	Barcode Type	Hex Value
Not Applicable	0x00	EAN-8 + 5 Supps.	0x8A
Bookland EAN	0x16	Interleave 2 of 5	0x06
Codabar	0x02	ISBT 128 Concat.	0x21
Code 128	0x03	ISBT-128	0x19
Code 39	0x01	MSI Plessey	0x0E
Code 39 Full ASCII	0x13	Trioptic Code 39	0x15
Code 93	0x07	UPC-A	0x08
Coupon Code	0x17	UPC-A + 2 Supps.	0x48
Discrete 2 of 5	0x04	UPC-A + 5 Supps.	0x88
EAN-128	0x0F	UPC-E	0x09
EAN-13	0x0B	UPC-E + 2 Supps.	0x49
EAN-13 + 2 Supps.	0x4B	UPC-E + 5 Supps.	0x89
EAN-13 + 5 Supps.	0x8B	UPC-E1	0x10
EAN-8	0x0A	UPC-E1 + 2 Supps.	0x50
EAN-8 + 2 Supps.	0x4A	UPC-E1 + 5 Supps.	0x90

Value is 0x00 if the scanner is disabled or no barcode has been scanned.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
byte codeType = barCodeReader.getCodeType();
```

isEnabled

Class

BarCodeReader

Description

Determines if the scanner is currently enabled and ready for use.

Invocation

boolean isEnabled()

Return Value

true	if scanner is enabled
false	if scanner is not enabled

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.isEnabled();
```

resetScanner

Class

BarCodeReader

Description

Reconfigures the scanner to the factory default settings. Any application applied configurations will be lost.

Invocation

boolean resetScanner()

Return Value

true	scanner was reset successfully.
false	failed to reset scanner.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.resetScanner();
```

Sets the scanner back to its factory configuration.

setDataTransFormat

Class

BarCodeReader

Description

Determines the format of the decoded data which is returned to the application. Also see setPrefix, setSuffix1 and setSuffix2.

Invocation

boolean setDataTransFormat(byte val)

val	valid range 0-7	
	0	data as is
	1	<data><suffix1>
	2	<data><suffix2>
	3	<data><suffix1><suffix2>
	4	<prefix><data>
	5	<prefix><data><suffix1>
	6	<prefix><data><suffix2>
	7	<prefix><data><suffix1><suffix2>

Return Value

true	set data transmit format successfully.
false	failed to set data transmit format.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.setSuffix1((byte)0x07);
```

Sets the format of data to: <prefix><data><suffix2><suffix2>.

setLaserOnTime

Class

BarCodeReader

Description

Sets the amount of time the decoder will try to decode a barcode. The scanner must be enabled and configured before calling this function. By default, this value is 3.0 seconds.

Invocation

void setLaserOnTime(int seconds, int fraction)

seconds	the whole number of seconds; valid range 0-9
fraction	the tenths of second; valid range 0-9

Valid range is 0.5 seconds to 9.9 seconds.

Return Value

None

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
barCodeReader.setLaserOnTime((int )0x05, (int )0x09);
```

Sets the laser on time to 5.9 seconds.

setLinearSecurity

Class

BarCodeReader

Description

For security levels 1-3, determines which linear code types and lengths must be read twice before being decoded. For security level 4 determines which linear codes must be read three times before being decoded.

There is an inverse relationship between security and scanner aggressiveness, so be sure to set only that level of security necessary for any given application.

Invocation

boolean setLinearSecurity(byte level)

level	valid range 1-4		
	Level 1	Code Type	Length
		Codabar	All
		MSI Plessey	4 or less
		Discrete 2 of 5	8 or less
		Interleaved 2 of 5	8 or less
	Level 2	Code Type	Length
		All	All
	Level 3	Code Type	Length
		MSI Plessey	4 or less
		Discrete 2 of 5	8 or less
		Interleaved 2 of 5	8 or less
	Level 4	Code Type	Length
		All	All

Return Value

true	security level set successfully.
false	failed to set security level.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.setLinearSecurity((byte)0x02);
```

Sets the linear security level to 2 which requires all codes to be read twice before being decoded.

setPrefix

Class

BarCodeReader

Description

Set the hex value of the character to be used as the prefix to any decoded data. Note that setDataTransFormat determines if the prefix will actually be sent or not.

Invocation

boolean setPrefix(byte val)

val	hex value of the ASCII character to be used as the prefix to the decoded data. Default value = NULL.

Return Value

true	set prefix successfully.
false	failed to set prefix.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.setPrefix((byte)0x25);
```

Sets the prefix for decoded data to ’%’.

setSuffix1

Class

BarCodeReader

Description

Set the hex value of the character to be used as the first suffix to any decoded data. Note that setDataTransFormat determines if the first suffix will actually be sent or not.

Invocation

boolean setSuffix1(byte val)

val	hex value of the ASCII character to be used as the first suffix to the decoded data. Default value = LF (0x0a).

Return Value

true	set suffix1 successfully.
false	failed to set suffix1.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.setSuffix1((byte)0x21);
```

Sets the suffix1 for decoded data to '!'.

setSuffix2

Class

BarCodeReader

Description

Set the hex value of the character to be used as the second suffix to any decoded data. Note that setDataTransFormat determines if the second suffix will actually be sent or not.

Invocation

boolean setSuffix2 (byte val)

val	Set the hex value of the character to be used as the second suffix to any decoded data. Note that setDataTransFormat determines if the second suffix will actually be sent or not. Default value = CR (0x0d).

Return Value

true	set suffix2 successfully.
false	failed to set suffix2.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.setSuffix2((byte)0x24);
```

Sets the suffix2 for decoded data to ’$’.

startDecode

Class

BarCodeReader

Description

Returns a string containing the decoded text. Returns after a successful decode or timeout.

Note: *To guarantee an accurate decode, ensure that the startDecode function is not called while the phone is transmitting.*

Invocation

String startDecode()

Return Value

String containing decoded text.

Returns 'NR' if failed to decode, or timeout

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
String labeltext = barCodeReader.startDecode();
```

transmitCodeId

Class

BarCodeReader

Description

Returns an ID character along with the decoded barcode data. The code ID will be placed in front of the decoded barcode data. Note that if a prefix is being returned (see setDataTransFormat) the code ID will follow the prefix and immediately precede the decoded data. See Appendix A, *Programming Reference* for a list of code IDs.

Invocation

boolean transmitCodeId(byte val)

val	valid range 0-2	
	0	do not send Code ID.
	1	send AIM code ID.
	2	send Symbol code ID.

Return Value

true	code ID sent successfully.
false	failed to send code ID.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.transmitCodeId((byte)0x02);
```

Send the Symbol code ID along with the barcode data and any prefix and suffix character(s).

Decode Configuration Commands

booklandEAN

Class

BarCodeReader

Description

Enable or disable decoding of Bookland-EAN barcodes and set label options.

Invocation

boolean upcBooklandEAN(boolean enable)

enable	= false - Disable Bookland-EAN decoding
	= true - Enable Bookland-EAN decoding

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.booklandEAN (true);
```

Enable Bookland-EAN type barcodes.

codabar

Class

BarCodeReader

Description

Enable or disables decoding of Codabar bar codes and options.

Invocation

boolean codabar (boolean enable, byte options, byte len1, byte len2)

enable	= false - Disable Codabar decoding
	= true - Enable Codabar decoding
options	Sets Codabar bar code options. Bit mapped options logically OR'd as needed
	= 1 NOTIS
	= 2 CLSI
len1	See len2 description
len2	If len2 = 0 then len1 determines the discrete length of the bar code.
	If len1 > len2 then, two discrete lengths are acceptable, where len1 and len2 determine the actual lengths.
	If len1 < len2 then valid length of bar code falls within the range determined by len1 and len2
	If len1 = len2 = 0, then any length of Codabar bar code is accepted.

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. codabar
( true, (byte)0x03, (byte)0x00, (byte)0x00 );
```

Enable decoding Codabar labels, NOTIS and CLSI, and allow any length label.

code128

Class

BarCodeReader

Description

Enable or disables decoding of Code128 bar codes.

Invocation

boolean code128 (boolean enable)

enable	= false - Disable USS-128 decoding
	= true - Enable USS-128 decoding

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.code128 (true);
```

Enable decoding of Code128.

code39

Class

BarCodeReader

Description

Enable or disables decoding of Code 39 bar codes and options.

Invocation

boolean code39 (boolean enable, boolean checkdigit, byte options, byte len1, byte len2)

enable	= false - Disable Code 39 decoding
	= true - Enable Code 39 decoding
checkdigit	= false - Disable check digit verification
	= true - Enable check digit verification
options	Sets Code 39 bar code options. Bit mapped options logically OR'd as needed
	= 1 Convert Code 39 to Code 32
	= 2 Add prefix of "A" to Code 32 barcodes. Code 39 to 32 must be enabled
	= 4 Transmit check digit.
	= 8 Code 39 Full ASCII
len1	See len2 description
len2	If len2 = 0 then len1 determines the discrete length of the bar code.
	If len1 > len2 then, two discrete lengths are acceptable, where len1 and len2 determine the actual lengths.
	If len1 < len2 then valid length of bar code falls within the range determined by len1 and len2
	If len1 = len2 = 0, then any length of Code 39 bar code is accepted.

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. code39
( true, true, (byte)0x05, (byte)0x00, (byte)0x00 );
```

Enable decoding of Code 39 labels, enable check digit verification, convert Code 39 to Code 32, transmit check digit, and allow any length label.

code39Trioptic

Class

BarCodeReader

Description

Enable or disables decoding of Trioptic Code 39 bar codes.

Invocation

boolean code39Trioptic (boolean enable)

enable	= false - Disable Trioptic decoding
	= true - Enable Trioptic decoding

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. code39Trioptic (true);
```

Enable decoding of Trioptic Code39 labels.

code93

Class

BarCodeReader

Description

Enable or disables decoding of Code 93 bar codes and lengths.

Invocation

boolean code93 (boolean enable, byte len1, byte len2)

enable	= false - Disable Code 93 decoding
	= true - Enable Code 93 decoding
len1	See len2 description
len2	If len2 = 0 then len1 determines the discrete length of the bar code.
	If len1 > len2 then, two discrete lengths are acceptable, where len1 and len2 determine the actual lengths.
	If len1 < len2 then valid length of bar code falls within the range determined by len1 and len2
	If len1 = len2 = 0, then any length of Code 93 bar code is accepted.

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. code93 ( true, (byte)0x00, (byte)0x00 );
```

Enable decoding of Code 93 labels and allow any length label.

discrete2of5

Class

BarCodeReader

Description

Enable or disables decoding of Discrete 2 of 5 bar codes and lengths.

Note: *It is highly recommended to use discrete lengths greater than 8 on low security barcodes, such as Discrete 2of5 and Interleaved 2of5, to avoid poor system performance.*

Invocation

boolean discrete2of5 (boolean enable, byte len1, byte len2)

enable	= false - Disable Discrete 2 of 5 decoding
	= true - Enable Discrete 2 of 5 decoding
len1	See len2 description
len2	If len2 = 0 then len1 determines the discrete length of the bar code.
	If len1 > len2 then, two discrete lengths are acceptable, where len1 and len2 determine the actual lengths.
	If len1 < len2 then valid length of bar code falls within the range determined by len1 and len2
	If len1 = len2 = 0, then any length of Discrete 2 of 5 bar code is accepted.

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. discrete2of5 ( true, (byte)0x00, (byte)0x00 );
```

Enable decoding of Discrete 2of5 labels, and allow any length label.

ean128

Class

BarCodeReader

Description

Enable or disables decoding of UCC/EAN-128 bar codes.

Invocation

boolean ean128 (boolean enable)

enable	= false - Disable UCC/EAN-128 decoding
	= true - Enable UCC/EAN-128 decoding

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.ean128 (true);
```

Enable decoding of UCC/EAN-128.

ean13

Class

BarCodeReader

Description

Enable or disable decoding of EAN-13 barcodes and set label options.

Invocation

boolean upcEAN13(boolean enable)

enable	= false - Disable EAN-13 decoding
	= true - Enable EAN-13 decoding

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.ean13(true);
```

Enable EAN-13 type barcodes.

ean8

Class

BarCodeReader

Description

Enable or disable decoding of EAN-8 barcodes and set label options.

Invocation

boolean upcEAN8(boolean enable, boolean extend)

enable	= false - Disable EAN-8 decoding
	= true - Enable EAN-8 decoding
extend	= false - Do not convert EAN-8 to EAN-13
	= true - Convert EAN-8 to EAN-13

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.ean8(true, true);
```

Enable EAN-8 type barcodes, extend EAN-8 to 13 characters.

interleaved2of5

Class

BarCodeReader

Description

Enable or disables decoding of Interleaved 2 Of 5 bar codes and options.

Note: *It is highly recommended to use discrete lengths greater than 8 on low security barcodes, such as Discrete 2of5 and Interleaved 2of5, to avoid poor system performance.*

Invocation

boolean interleaved2of5 (boolean enable, byte checkdigit, byte options, byte len1, byte len2)

enable	= false - Disable Interleaved 2 of 5 decoding
	= true - Enable Interleaved 2 of 5 decoding
checkdigit	= 0 Disable check digit verification
	= 1 Enable USS check digit verification
	= 2 Enable OPCC check digit verification
options	Sets Interleaved 2 of 5 bar code options. Bit mapped options logically OR'd as needed
	= 1 Transmit check digit
	= 2 Convert to EAN-13
len1	See len2 description
len2	If len2 = 0 then len1 determines the discrete length of the bar code.
	If len1 > len2 then, two discrete lengths are acceptable, where len1 and len2 determine the actual lengths.
	If len1 < len2 then valid length of bar code falls within the range determined by len1 and len2
	If len1 = len2 = 0, then any length of Interleaved 2 of 5 bar code is accepted.

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. interleaved2of5
( true, (byte)0x01, (byte)0x03, (byte)0x00, (byte)0x00 );
```

Enable decoding of Interleaved 2of5 labels, enable check digit verification, transmit check digit, convert to EAN-13, and allow any length label.

isbt128

Class

BarCodeReader

Description

Enable or disables decoding of ISBT-128 bar codes.

Invocation

boolean isbt128 (boolean enable)

enable	= false - Disable ISBT-128 decoding
	= true - Enable ISBT-128 decoding

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. isbt128 (true);
```

Enable decoding of ISBT-128.

msiPlessey

Class

BarCodeReader

Description

Enable or disables decoding of MSI Plessey bar codes and options.

Invocation

boolean msiPlessey (boolean enable, byte checkdigit, byte options, byte len1, byte len2)

enable	= false - Disable MSI Plessey decoding
	= true - Enable MSI Plessey decoding
checkdigit	= 0 Apply 1 check digit
	= 1 Apply 2 check digits
options	Sets MSI Plessey bar code options. Bit mapped options logically OR'd as needed
	= 1 Transmit check digit
	= 2 Check digit algorithm, true = mod10/mod10; false = mod10/mod11
len1	See len2 description
len2	If len2 = 0 then len1 determines the discrete length of the bar code.
	If len1 > len2 then, two discrete lengths are acceptable, where len1 and len2 determine the actual lengths.
	If len1 < len2 then valid length of bar code falls within the range determined by len1 and len2
	If len1 = len2 = 0, then any length of Interleaved 2 of 5 bar code is accepted.

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. msiPlessey
( true, (byte)0x01, (byte)0x03, (byte)0x00, (byte)0x00 );
```

Enable decoding of MSI Plessey labels, apply 2 check digits, transmit check digit, mod10/mod10 check digit algorithm, convert to EAN-13, and allow any length label.

upcA

Class

BarCodeReader

Description

Enable or disable decoding of UPC-A barcodes and set label options.

Invocation

boolean upcA(boolean enable, boolean checkdigit, byte preamble)

enable	= false - Disable UPC-A decoding
	= true - Enable UPC-A decoding
checkdigit	= false - Do not send check digit
	= true - Send check digit
preamble	= 0 Do not apply preamble character to label
	= 1 Apply system character as preamble to label
	= 2 Apply country code and system character as preamble to label

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.upcA(true, false, (byte)0x02);
```

Enable UPC-A type barcodes, do not send the check digit, and apply country code and system characters as a preamble to the label.

upcE

Class

BarCodeReader

Description

Enable or disable decoding of UPC-E barcodes and set label options.

Invocation

boolean upcE(boolean enable, boolean checkdigit, byte preamble, boolean convert)

enable	= false - Disable UPC-E decoding
	= true - Enable UPC-E decoding
checkdigit	= false - Do not send check digit
	= true - Send check digit
preamble	= 0 Do not apply preamble character to label
	= 1 Apply system character as preamble to label
	= 2 Apply country code and system character as preamble to label
convert	= 0 Do not convert UPC-E to UPC-A
	= 1 Convert UPC-E to UPC-A

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.upcE
(true, true, (byte)0x00, true);
```

Enable UPC-E type barcodes, send the check digit, do not apply a preamble to the label, convert UPC-E to UPC-A.

upcE1

Class

BarCodeReader

Description

Enable or disable decoding of UPC-E1 barcodes and set label options.

Invocation

boolean upcE1(boolean enable, boolean checkdigit, byte preamble, boolean convert)

enable	= false - Disable UPC-E1 decoding
	= true - Enable UPC-E1 decoding
checkdigit	= false - Do not send check digit
	= true - Send check digit
preamble	= 0 Do not apply preamble character to label
	= 1 Apply system character as preamble to label
	= 2 Apply country code and system character as preamble to label
convert	= false - Do not convert UPC-E1 to UPC-A
	= true - Convert UPC-E1 to UPC-A

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader.upcE1
(true, true, (byte)0x00, true);
```

Enable UPC-E1 type barcodes, send the check digit, do not apply a preamble to the label, convert UPC-E1 to UPC-A.

upcEANCouponCode

Class

BarCodeReader

Description

Enable or disables decoding of UPC-A, UPC-A with 2 supplemental characters, UPC-A with 5 supplemental characters, and UPC-A/EAN-128 bar codes. Autodiscriminate UPC/EAN Supplementals must be enabled.

Invocation

boolean upcEANCouponCode (boolean enable)

enable	= false - Disable UPC/EAN Coupon Code
	= true - Enable UPC/EAN Coupon Code

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. upcEANCouponCode (true);
```

Enable UPC/EAN coupon codes.

upcEANSecurity

Class

BarCodeReader

Description

Set the level of security for decoding UPC/EAN bar codes. The security level goes up as the quality of the bar code diminishes.

There is an inverse relationship between security and scanner aggressiveness, so be sure to set only that level of security necessary for any given application.

Invocation

boolean upcEANSecurity (byte level)

count	= 0 Most aggressive state. Provides sufficient security in decoding "in-spec" UPC/EAN bar codes.
	= 1 Allows decoding of poorly printed barcodes whose mis-decodes are limited to characters 1, 2, 7, 8.
	= 2 Allows decoding of poorly printed barcodes whose mis-decodes are not limited to characters 1, 2, 7, 8.
	= 3 If level 2 fails then select this level. This level is an extreme measure against mis-decoding severely out of spec bar codes. The quality of the barcodes should really be improved if this level is necessary.

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. upcEANSecurity ((byte)0x02);
```

Set security level to 2 for decoding UPC/EAN labels.

upcEANSupplemental

Class

BarCodeReader

Description

Set supplemental mode for UPC/EAN labels.

Note: *The Auto discriminate mode, if enabled, will reduce aggressiveness on UPC/EAN labels without supplementals. If Auto discriminate is enabled, choose an appropriate upcEANSupplReduncancy (see page 2-44) value.*

Invocation

boolean upcEANSupplemental (byte mode)

mode	= 0 Ignore supplemental
	= 1 Decode UPC/EAN with supplementals
	= 2 Auto discriminate UPC/EAN supplementals

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. upcEANSupplemental ((byte)0x01);
```

Enable UPC/EAN with supplemental barcodes.

upcEANSupplRedundancy

Class

BarCodeReader

Description

With Auto discriminate UPC/EAN Supplementals selected, this call adjusts the number of times a symbol without supplementals will be decoded before transmission.

A minimum of 5 times is recommended.

Invocation

boolean upcEANSupplRedundancy (byte count)

count	valid range is 2 - 20

Return Value

true	if configuration was successful.
false	if configuration failed.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
boolean response = barCodeReader. upcEANSupplRedundancy ((byte)0x05);
```

Decode UPC/EAN label 5 times before returning it to the application.

Miscellaneous

getSDKVersion

Class

BarCodeReader

Description

Returns the version of the SDK that was used in creating the midlet.

Invocation

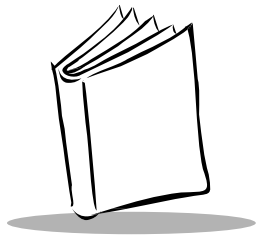
String getSDKVersion()

Return Value

Returns a string containing the version of the SDK.

Example

```
import com.symbol.j2me.midlets.ascanner.BarCodeReader;
BarCodeReader barCodeReader = new BarCodeReader();
String sversion  =  barCodeReader.getSDKVersion();
```



Appendix A
Programming Reference

Symbol Code Identifiers

Table A-1. Code Characters

Code Character	Code Type
A	UPC-A, UPC-E, UPC-E1, EAN-8, EAN-13
B	Code 39
C	Codabar
D	Code 128
E	Code 93
F	Interleaved 2 of 5
G	Discrete 2 of 5, or Discrete 2 of 5 IATA
J	MSI
K	UCC/EAN-128
L	Bookland EAN
M	Trioptic Code 39
N	Coupon Code

AIM Code Identifiers

Each AIM Code Identifier contains the three-character string **]cm** where:

] = Flag Character (ASCII 93)
c = Code Character (see Table A-2)
m = Modifier Character (see Table A-3)

Table A-2. Code Characters

Code Character	Code Type
A	Code 39
C	Code 128
E	UPC/EAN
F	Codabar
G	Code 93
H	Code 11
I	Interleaved 2 of 5
M	MSI
S	D2 of 5, IATA 2 of 5
X	Code 39 Trioptic
X	Bookland EAN
X	Coupon Code

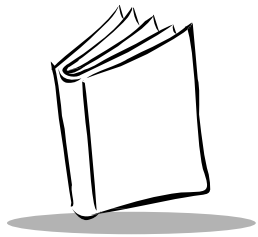
The modifier character is the sum of the applicable option values based on Table A-3.

Table A-3. Modifier Characters

Code Type	Option Value	Option
Code 39	0	No check character or Full ASCII processing.
	1	Reader has checked one check character.
	3	Reader has checked and stripped check character.
	4	Reader has performed Full ASCII character conversion.
	5	Reader has performed Full ASCII character conversion and checked one check character.
	7	Reader has performed Full ASCII character conversion and checked and stripped check character.
	Example:A Full ASCII bar code with check character W, **A+I+MI+DW**, is transmitted as **]A7**AIMID where 7 = (3+4).	
Trioptic Code 39	0	No option specified at this time. Always transmit 0.
	Example:A Trioptic bar code 412356 is transmitted as **]X0**412356	
Code 128	0	Standard data packet, no Function code 1 in first symbol position.
	1	Function code 1 in first symbol character position.
	2	Function code 1 in second symbol character position.
	Example:A Code (EAN) 128 bar code with Function 1 character in the first position, FNC1 AIMID is transmitted as **]C1**AIMID	
I 2 of 5	0	No check digit processing.
	1	Reader has validated check digit.
	3	Reader has validated and stripped check digit.
	Example:An I 2 of 5 bar code without check digit, 4123, is transmitted as **]I0**4123	
Codabar	0	No check digit processing.
	1	Reader has checked check digit.
	3	Reader has stripped check digit before transmission.
	Example:A Codabar bar code without check digit, 4123, is transmitted as **]F0**4123	

Table A-3. Modifier Characters (Continued)

Code Type	Option Value	Option
Code 93	0	No options specified at this time. Always transmit 0.
	Example:A Code 93 bar code 012345678905 is transmitted as **]G0**012345678905	
MSI	0	Check digits are sent.
	1	No check digit is sent.
	Example:An MSI bar code 4123, with a single check digit checked, is transmitted as **]M1**4123	
D 2 of 5	0	No options specified at this time. Always transmit 0.
	Example:A D 2 of 5 bar code 4123, is transmitted as **]S0**4123	
UPC/EAN	0	Standard packet in full EAN country code format, which is 13 digits for UPC-A and UPC-E (not including supplemental data).
	1	Two-digit supplement data only.
	2	Five-digit supplement data only.
	4	EAN-8 data packet.
	Example:A UPC-A bar code 012345678905 is transmitted as **]E0**0012345678905	
Bookland EAN	0	No options specified at this time. Always transmit 0.
	Example:A Bookland EAN bar code 123456789X is transmitted as **]X0**123456789X	



Appendix B
Sample Program

The sample program in this Appendix allows you to enable and disable to barcode scanner and program the scanner to decode Code 39 and Code 39 Trioptic barcodes.

```
/** This software is provided to you by Symbol Technologies, Inc. under a
* license from Motorola, Inc. You may copy, modify and use this software
* solely for purposes of designing, manufacturing, and selling the barcode
* reader accessory or other devices that interact with or control this
* accessory. No other license, express or implied, relating to the software
* is granted to you.
*/

/*
*
*  © Copyright 2000 Motorola, Inc.  All Rights Reserved.
*  This notice does not imply publication.
*/

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import java.io.*;
import javax.microedition.io.*;

import  com.symbol.j2me.midlets.ascanner.BarCodeReader;

public class c39Test extends MIDlet implements CommandListener {
  public static final boolean COLOR = false;     // color support
  public static final boolean DEBUG = false;     // debug statements

  // basic colors
  public static final int WHITE = 0xFFFFFF;
  public static final int BLACK = 0x000000;
```

```
public static final int BLUE = 0x0000FF;
public static final int LIGHT_GRAY = 0xAAAAAA;
public static final int DARK_GRAY = 0x555555;

// reference to current display
private Display myDisplay = null;
private List mainScreen = null;
private List c39Screen = null;
private List c39TriopticScreen = null;
private Form resultScreen = null;
private Command okCommand = new Command("OK", Command.OK, 1);
private Command backCommand = new Command("Back", Command.BACK, 2);
private Command exitCommand = new Command("Exit", Command.EXIT, 2);
private BarCodeReader barCodeReader = null;
private Alert alert = null;
private DecodeCanvas decodeCanvas = null;

private String sdkver = "";

private boolean response = true;

public c39Test() {

  myDisplay = Display.getDisplay(this);
  barCodeReader = new BarCodeReader();

  // Get version of SDK
  sdkver = barCodeReader.getSDKVersion();

  decodeCanvas = new DecodeCanvas(this, barCodeReader);
  decodeCanvas.addCommand(backCommand);
  decodeCanvas.setCommandListener(this);

  // create main screen

  mainScreen = new List("CODE 39 Test", List.IMPLICIT);
  mainScreen.append("Enable Scanner", null);
  mainScreen.append("Scan Barcodes", null);
  mainScreen.append("Disable Scanner", null);
  mainScreen.append("Code 39", null);
  mainScreen.append("Code 30 Trioptic", null);

  mainScreen.addCommand(okCommand);
  mainScreen.addCommand(exitCommand);
  mainScreen.setCommandListener(this);

  // create result screen
  resultScreen = new Form("Result");
```

```
  resultScreen.addCommand(okCommand);
  resultScreen.setCommandListener(this);

  // create Code 39 screen
  c39Screen = new List("Code 39", List.IMPLICIT);
  c39Screen.append("Enable Code 39",null);
  c39Screen.append("Disable Code 39",null);
  c39Screen.addCommand(okCommand);
  c39Screen.addCommand(backCommand);

  // create Code 39 Trioptic screen
  c39TriopticScreen = new List("Code 39 Trioptic", List.IMPLICIT);
  c39TriopticScreen.append("Enable C39 Trioptic",null);
  c39TriopticScreen.append("Disable C39 Trioptic",null);
  c39TriopticScreen.addCommand(okCommand);
  c39TriopticScreen.addCommand(backCommand);

}

/**
* Start the MIDlet
*/
public void startApp() throws MIDletStateChangeException {
  myDisplay.setCurrent(mainScreen);
}

/**
* Pause the MIDlet
*/
public void pauseApp() {

}

/**
* Called by the framework before the application is unloaded
*/
public void destroyApp(boolean unconditional)
  throws MIDletStateChangeException {
}

/**
* Indicates that a command event has occurred on Screen d
* @param c a Command object identifying the command
* @param d the Screen on which this event has occurred
*/
public void commandAction(Command c, Displayable d) {
```

```
// make a selection from the main screen
if ((c == okCommand) && (d == mainScreen)) {
  int selected = mainScreen.getSelectedIndex();
  try {
    switch (selected) {
      case 0:                  // Enable Scanner
        if (barCodeReader.isEnabled()) {
          alert = new Alert("Error", "Reader is already enabled", null,
                  AlertType.ERROR);
          alert.setTimeout(Alert.FOREVER);
          myDisplay.setCurrent(alert);
        }
        else {
          resultScreen = new Form("Result");
                         resultScreen.append(sdkver); // 07/29/2002 TM
          resultScreen.append("Enabling barcode reader. Please wait..");
          myDisplay.setCurrent(resultScreen);
          EnableThread t = new EnableThread(this);
          t.start();
        }
        break;
      case 1:                  // Scan Barcodes
        if (!barCodeReader.isEnabled()) {
          alert = new Alert("Error", "Reader is not enabled!", null,
                  AlertType.ERROR);
          alert.setTimeout(Alert.FOREVER);
          myDisplay.setCurrent(alert);
        }
        else {

          decodeCanvas.setDecodeText("");
          myDisplay.setCurrent(decodeCanvas);
        }
        break;
      case 2:                  // Disable Scanner
        if (!barCodeReader.isEnabled()) {
          alert = new Alert("Error", "Reader is not enabled!", null,
                  AlertType.ERROR);
          alert.setTimeout(Alert.FOREVER);
          myDisplay.setCurrent(alert);
        }
        else {
          resultScreen = new Form("Result");
          resultScreen.append("Disable - ");
```

```
        barCodeReader.disable();
        resultScreen.append("OK");
        resultScreen.addCommand(okCommand);
        resultScreen.setCommandListener(this);
        myDisplay.setCurrent(resultScreen);
      }
      break;

    case 3:                 // Code 39
      if (!barCodeReader.isEnabled()) {
        alert = new Alert("Error", "Reader is not enabled", null,
                AlertType.ERROR);
        alert.setTimeout(Alert.FOREVER);
        myDisplay.setCurrent(alert);
      }
      else {
        c39Screen.setCommandListener(this);
        myDisplay.setCurrent(c39Screen);
      }
      break;

    case 4:                 // Code 39 Trioptic
      if (!barCodeReader.isEnabled()) {
        alert = new Alert("Error", "Reader is not enabled", null,
                AlertType.ERROR);
        alert.setTimeout(Alert.FOREVER);
        myDisplay.setCurrent(alert);
      }
      else {
        c39TriopticScreen.setCommandListener(this);
        myDisplay.setCurrent(c39TriopticScreen);
      }
      break;

  }
 }
 catch (Exception e) {
  System.err.println(e);
  resultScreen = new Form("Error!");
  resultScreen.append("\n" + e.toString());
  resultScreen.addCommand(okCommand);
  resultScreen.setCommandListener(this);
  myDisplay.setCurrent(resultScreen);
 }
}
```

```
// make a selection from the Code 39 screen
if ((c == okCommand) && (d == c39Screen)) {
  int selected = c39Screen.getSelectedIndex();
  try {
    switch (selected) {
      case 0:                  // Enable CODE 39
        response = barCodeReader.code39(true,
          false,(byte)0x00,(byte)0x0A,(byte)0x06);
        if(!response){
          alert = new Alert("Error", "C39 Enable FAILED", null,
                  AlertType.ERROR);
          alert.setTimeout(Alert.FOREVER);
          myDisplay.setCurrent(alert);
        } else {
          resultScreen = new Form("Result");
          resultScreen.append("Code 39 Enabled!");
          resultScreen.addCommand(okCommand);
          resultScreen.setCommandListener(this);
          myDisplay.setCurrent(resultScreen);
        }
        break;

      case 1:                  // Disable CODE 39
        response = barCodeReader.code39(false,
          false,(byte)0x00,(byte)0x00,(byte)0x00);
        if(!response){
          alert = new Alert("Error", "C39 Disable FAILED", null,
                  AlertType.ERROR);
          alert.setTimeout(Alert.FOREVER);
          myDisplay.setCurrent(alert);
        } else {
          resultScreen = new Form("Result");
          resultScreen.append("Code 39 Disabled!");
          resultScreen.addCommand(okCommand);
          resultScreen.setCommandListener(this);
          myDisplay.setCurrent(resultScreen);
        }
        break;
    }
  }
  catch (Exception e) {
    System.err.println(e);
    resultScreen = new Form("Error!");
    resultScreen.append("\n" + e.toString());
```

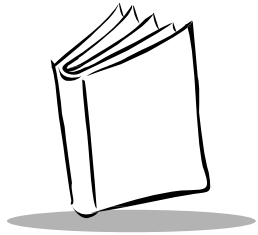
```
    resultScreen.addCommand(okCommand);
    resultScreen.setCommandListener(this);
    myDisplay.setCurrent(resultScreen);
  }
}

// make a selection from the Code 39 Trioptic screen
if ((c == okCommand) && (d == c39TriopticScreen)) {
  int selected = c39TriopticScreen.getSelectedIndex();
  try {
    switch (selected) {
      case 0:                  // Enable CODE 39 Trioptic
        response = barCodeReader.code39Trioptic(true);
        if(!response){
          alert = new Alert("Error", "C39 Trioptic Enable FAILED", null,
                  AlertType.ERROR);
          alert.setTimeout(Alert.FOREVER);
          myDisplay.setCurrent(alert);
        } else {
          resultScreen = new Form("Result");
          resultScreen.append("C39 Trioptic Enabled!");
          resultScreen.addCommand(okCommand);
          resultScreen.setCommandListener(this);
          myDisplay.setCurrent(resultScreen);
        }
        break;
      case 1:                  // Disable CODE 39 Trioptic
        response = barCodeReader.code39Trioptic(false);
        if(!response){
          alert = new Alert("Error", "C39 Trioptic Disable FAILED", null,
                  AlertType.ERROR);
          alert.setTimeout(Alert.FOREVER);
          myDisplay.setCurrent(alert);
        } else {
          resultScreen = new Form("Result");
          resultScreen.append("C39 Trioptic Disabled!");
          resultScreen.addCommand(okCommand);
          resultScreen.setCommandListener(this);
          myDisplay.setCurrent(resultScreen);
        }
        break;
    }
  }
  catch (Exception e) {
```

```
      System.err.println(e);
      resultScreen = new Form("Error!");
      resultScreen.append("\n" + e.toString());
      resultScreen.addCommand(okCommand);
      resultScreen.setCommandListener(this);
      myDisplay.setCurrent(resultScreen);
    }
  }

  // go back to main screen from c39 screen
  if ((c == backCommand) && ( (d == c39Screen) ||
      (d == c39TriopticScreen))){
    myDisplay.setCurrent(mainScreen);
  }

  // go back to main screen from scan screen
  if ((c == backCommand) && (d == decodeCanvas)) {
    myDisplay.setCurrent(mainScreen);
  }

  // go back to main screen from the result screen
  if ((c == okCommand) && (d == resultScreen)) {
    myDisplay.setCurrent(mainScreen);
  }

  // exit the MIDlet
  if ((c == exitCommand) && (d == mainScreen)) {
    // if (barCodeReader.isEnabled())
    //    barCodeReader.disable();
    notifyDestroyed();
  }

}

class EnableThread extends Thread {
  c39Test parent = null;

  public EnableThread(c39Test parent) {
    this.parent = parent;
  }

  public void run() {
    boolean result = false;
    try {
      System.out.println("Enabling..");
      result = barCodeReader.enable();
      System.out.println("Configuring..");
```

```
      resultScreen.delete(0);
      System.out.println("Done..");
    }
    catch (Exception e) {
      System.err.println(e);
    }
    if (result) {
      resultScreen.append("Ready to Scan");
    }
    else  {
      resultScreen.append("Error");
    }
    resultScreen.addCommand(okCommand);
    resultScreen.setCommandListener(parent);
    myDisplay.setCurrent(resultScreen);
  }
}

class DecodeCanvas extends Canvas {
  private c39Test parent = null;
  private BarCodeReader barCodeReader = null;
  private String decodeString = "";
  private int width = getWidth();
  private int height = getHeight();

  public DecodeCanvas(c39Test parent, BarCodeReader barCodeReader) {
    this.parent = parent;
    this.barCodeReader = barCodeReader;
  }

  public void paint(Graphics g) {
    // clear screen
    g.setColor(WHITE);       // white
    g.fillRect(0, 0, width, height);

    // draw the title
    Font f = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,
                      Font.SIZE_MEDIUM);
    int yPos = 0;
    if (COLOR)
      g.setColor(BLUE);       // blue
    else
      g.setColor(LIGHT_GRAY);
    g.fillRect(0, yPos, width, f.getHeight());
```

```
if (COLOR)
  g.setColor(WHITE);       // white
else
  g.setColor(BLACK);
g.setFont(f);
g.drawString("Barcode Scanner", 0, yPos, Graphics.LEFT |
             Graphics.TOP);
g.setColor(BLACK);       // black
g.drawLine(0, f.getHeight() + yPos - 1, width, f.getHeight() +
           yPos - 1);

// draw instructions
f = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_PLAIN,
                  Font.SIZE_SMALL);
g.setFont(f);
g.drawString("Press PTT to'", 0, 16, Graphics.TOP | Graphics.LEFT);
g.drawString("begin scanning", 0, 16 + f.getHeight() + 1,
             Graphics.TOP | Graphics.LEFT);

// draw the decoded text
if (decodeString == null)
  decodeString = "Unable to read";
System.out.println("String width: " + f.stringWidth(decodeString));
g.setFont(Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,
                   Font.SIZE_SMALL));
if (f.stringWidth(decodeString) < width) {
  g.drawString(decodeString, 0, height/2, Graphics.LEFT |
               Graphics.TOP);
}
else {
  int y = height/2;
  int start = 0;
  int increment = (width/f.charWidth('W'));
  int end = increment;
  while (end < decodeString.length()) {
    g.drawString(decodeString.substring(start, end), 0, y,
                 Graphics.TOP | Graphics.LEFT);
    y = y + f.getHeight() + 1;
    start = end;
    end = end + increment;
  }
  // draw the last portion of the string
  g.drawString(decodeString.substring(start), 0, y,
               Graphics.TOP | Graphics.LEFT);
```

```
      }
    }

    public void keyPressed(int keyCode) {
      switch (keyCode) {
        case -50:               // PTT button
          decodeString = barCodeReader.startDecode();
          if (decodeString == null) {
            AlertType.CONFIRMATION.playSound(myDisplay); // error
          }
          else {
            if (decodeString.equals("NR"))
              AlertType.ERROR.playSound(myDisplay);       // no decode
            else
              AlertType.INFO.playSound(myDisplay);        // correct decode
          }
          repaint();
          break;
        default:
          break;
      }
    }

    public void setDecodeText(String decodeString) {
      this.decodeString = decodeString;
      repaint();
    }
  }

}  // end of class c39Test
```



Index

B

Basic Function Commands 2-5
- Disable 2-5
- getCodeType 2-7
- isEnabled 2-9
- resetScanner 2-10
- setPrefix 2-15
- setSuffix1 2-16
- setSuffix2 2-17
- startDecode 2-18
- transmitCodeID 2-19

bullets vii

C

code identifiers
- AIM code IDs A-2
- Symbol code IDs A-1

D

Decode Configuration Commands 2-20
- booklandEAN 2-20
- codabar 2-21
- code128 2-23
- code39 2-24
- code39Trioptic 2-26
- code93 2-27
- discrete2of5 2-28
- ean128 2-30
- ean13 2-31
- interleaved2of5 2-33
- isbt128 2-35
- msiPlessey 2-36
- upcA 2-38
- upcE1 2-40
- upcEANCouponCode 2-41
- upcEANSupplRedundancy 2-44

Default Setting 2-4

F

Function Definitions 2-1

I

Installing the SDK 1-3

M

Miscellaneous Commands 2-45
- getSDKVersion 2-45

N

Notational conventions vii

S

Sample Program B-1
Service Information vii
Supported Barcode Type 2-4
symbol support center viii

PSM20i Bar Code Scanner Attachment
Programmer's Guide

72E-59169-01
Revision A — December 2002

Symbol Technologies, Inc. One Symbol Plaza, Holtsville N.Y. 11742-1300
http://www.symbol.com/manuals