

JavaScript Programming Guide (CR1500)



Copyright © 2014 - 2018 Code Corporation.

All Rights Reserved.

The software described in this manual may only be used in accordance with the terms of its license agreement.

No part of this publication may be reproduced in any form, or by any means, without written permission from Code Corporation. This includes electronic or mechanical means, such as photocopying or recording in information storage and retrieval systems.

NO WARRANTY. This technical documentation is provided AS-IS. Further, the documentation does not represent a commitment on the part of Code Corporation. Code Corporation does not warrant that the information is accurate, complete, or error-free. Any use of the technical documentation is at the risk of the user. Code Corporation reserves the right to make changes in specifications and other information contained in this document without prior notice. The reader should consult Code Corporation to determine whether any changes have been made. Code Corporation shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material. Code Corporation does not assume any product liability arising out of, or in connection with, the application or use of any product or application described herein.

NO LICENSE. No license is granted, either by implication, estoppel, or otherwise under any intellectual property rights of Code Corporation. Any use of hardware, software, and/or technology of Code Corporation is governed by its own agreement.

The following are trademarks or registered trademarks of Code Corporation:

CodeXML®, Maker, QuickMaker, CodeXML® Maker, CodeXML® Maker Pro, CodeXML® Router, CodeXML® Client SDK, CodeXML® Filter, HyperPage, CodeTrack, GoCard, GoWeb, ShortCode, GoCode®, Code Router, QuickConnect Code, Rule Runner®, Cortex®, CortexRM, CortexMobile, Code, Code Reader, CortexAG, CortexStudio, CortexTools, Affinity®, CortexDecoder, CortexJPOS, CR8200 Utility, CR8200 Utility2, and CortexOPOS.

All other product names mentioned in this manual may be trademarks of their respective companies and are hereby acknowledged.

For patent information, please refer to <http://codecorp.com/legal/patents.php>.

The Code Reader software uses the Mozilla SpiderMonkey JavaScript engine, which is distributed under the terms of the Mozilla Public License.

Version 1.0. The Code Reader software is based in part on the work of the Independent JPEG Group.

Code Corporation, 12393 South Gateway Park Place, Suite 600, Draper, UT 84020. www.codecorp.com



Table of Contents

1	Introduction	6
2	Document Purpose	6
3	Document Audience	6
4	Document and Coding Conventions	6
5	Related Documents	7
6	Related Utilities	7
7	JavaScript Resources	7
8	Regular Expression Resources	7
9	Installing and Running a .codeRules.js script on the Code Reader	8
10	Security	8
11	Debugging (See reader.debug)	8
12	Programming Concepts	8
13	Code JavaScript Objects	9
13.1	Reader Object	9
13.1.1	Callback Properties	9
13.1.1.1	reader.onDecodes(Decode Object Array decodes)	9
13.1.1.2	reader.onEvent(Event Object event)	9
13.1.1.3	reader.onRawData(string data)	10
13.1.1.4	reader.onConfigure(string config)	10
13.1.1.5	reader.onPowerChange(int mode)	10
13.1.1.6	reader.onBatteryChange(void)	11
13.1.1.7	reader.periodic(void)	11
13.1.1.8	reader.tick1Hz(void)	11
13.1.2	Informational Properties	11
13.1.2.1	reader.charging.....	11
13.1.2.2	reader.green	11
13.1.2.3	reader.amber	11
13.1.2.4	reader.red	11
13.1.2.5	reader.black	12
13.1.2.6	reader.softwareVersion	12
13.1.2.7	reader.baseModel	12
13.1.2.8	reader.model	12
13.1.2.9	reader.readerId.....	12
13.1.2.10	reader.hardwareVersion	12
13.1.2.11	reader.cabled	12
13.1.2.12	reader.locked.....	13
13.1.2.13	reader.debug.....	13
13.1.2.14	reader.decodes.....	13
13.1.2.15	reader.bdAddr	13
13.1.3	Reader Methods	13
13.1.3.1	reader.runScript.....	13
13.1.3.2	reader.configure	14



13.1.3.3	reader.beep	14
13.1.3.4	reader.vibrate	14
13.1.3.5	reader.indicateGoodRead	14
13.1.3.6	reader.indicateError	14
13.1.3.7	reader.shiftJisToUnicode	15
13.1.3.8	reader.unicodeToShiftJis	15
13.1.3.9	reader.indicateData	15
13.1.3.10	reader.indicateWireless	15
13.1.3.11	reader.indicateBattery	15
13.1.3.12	reader.getKeyboardStatus	15
13.1.4	Reader Decodes Object	15
13.1.4.1	decodes[i].decoderType	15
13.1.4.2	decodes[i].valid	15
13.1.4.3	decodes[i].x	15
13.1.4.4	decodes[i].y	16
13.1.4.5	decodes[i].bounds	16
13.1.4.6	decodes[i].time	16
13.1.4.7	decodes[i].symbology	16
13.1.4.8	decodes[i].quality_percent	17
13.1.4.9	decodes[i].aimSymbology	17
13.1.4.10	decodes[i].symbologyModifier	17
13.1.4.11	decodes[i].aimModifier	17
13.1.4.12	decodes[i].linkage	17
13.1.4.13	decodes[i].saPosition	17
13.1.4.14	decodes[i].saTotal	18
13.1.4.15	decodes[i].saParity	18
13.1.4.16	decodes[i].isConfig	18
13.1.4.17	decodes[i].decodeOutputFormat	18
13.1.4.18	decodes[i].data (type string)	18
13.2	Storage Object	19
13.2.1	Informational Properties	19
13.2.1.1	storage.isFull	19
13.2.1.2	storage.fullness_percent	19
13.2.2	Storage Methods	19
13.2.2.1	storage.read	19
13.2.2.2	storage.write	19
13.2.2.3	storage.append	20
13.2.2.4	storage.rename	20
13.2.2.5	storage.erase	20
13.2.2.6	storage.uploadFile	20
13.2.2.7	storage.defragment	20
13.2.2.8	storage.findFirstInternal	20
13.2.2.9	storage.findNextInternal	20
13.2.2.10	storage.findFirst	21
13.2.2.11	storage.findNext	21
13.2.2.12	storage.size	21
13.2.2.13	storage.getHeader	21
13.2.2.14	storage.listFiles	21
13.2.3	Data in Code Reader Local Storage	21
13.3	Comm Object	22
13.3.1	Informational Properties	22
13.3.1.1	comm.isConnected	22



13.3.2	Comm methods.....	22
13.3.2.1	comm.connect	22
13.3.2.2	comm.disconnect.....	22
13.3.2.3	comm.sendPacket.....	23
13.3.2.4	comm.sendText	23
13.3.2.5	comm.sendMessage	23
13.3.2.6	comm.getSendStatus.....	23
13.3.2.7	comm.sendUsbScanCode	23
13.3.2.8	comm.sendUsbScanCodes.....	23
13.3.2.9	comm.sendHidReport.....	24
13.4	Shell Functions.....	24
13.4.1	help([name]).....	24
13.4.2	include(name)	24
13.4.3	print (string) – see reader.print(string)	24
13.4.4	gc().....	24
13.4.5	sleep_ms(time_ms)	24
13.4.6	format(string format, ...)	25
13.4.7	wdt_pet().....	25
13.4.8	logMessage(string)	25
13.5	Pre-defined Scripts.....	25
13.5.1	.jseLib.js library functions.....	26
13.5.1.1	storage.findFirst(regex)	26
13.5.1.2	storage.findNext().....	26
13.5.1.3	reader.setInterval (func, interval).....	26
13.5.1.4	reader.clearInterval(intervalId)	26
13.5.1.5	reader.setTimeout (func, delay)	26
13.5.1.6	reader.clearTimeout(timeoutId).....	26
13.5.1.7	reader.tick1Hz() – default implementation	26
13.5.2	cra.js	27
13.5.2.1	reader.onEvent(event).....	27
13.5.2.2	reader.onDecodeAttempt(count).....	27
13.5.2.3	reader.onDecode(decode).....	27
13.5.2.4	reader.onDecodes(decodes).....	27
13.5.3	.codeRules.js.....	27
13.5.3.1	rules_onDecodes	27
13.5.3.2	rules_onDecodeAttempt	28
13.5.3.3	rules_onDecode	28
13.5.3.4	rules_onEvent	28
13.5.3.5	Optional Global variables.....	29
14	Appendix.....	29
14.1	Sending Keystrokes (CodeXml)	29
14.2	Glossary and Acronyms	30
14.3	Supported JavaScript Core.....	30
14.4	Symbology Identifier Values.....	32



1 Introduction

Code Corporation (Code) designs, develops, and manufactures image-based readers and software tools for data collection applications. With expertise in software development, optics, imaging, and Bluetooth™ wireless technology, Code is an innovative leader in the Auto ID and Data Collection Industries.

The CR1500 is one of the many Code readers that combines bar code reading with an easy-to-use JavaScript-based application development.

2 Document Purpose

This manual describes the JavaScript application programming interface (API) for the Code Reader. It is assumed the user has programming knowledge and familiarity with the JavaScript language.

- Code Readers read barcodes and can be programmed to transmit code data over a selected communications link, or to store data in reader memory (batch mode).
- The programming environment provides interfaces to:
 - Read and manipulate data in reader memory
 - Access data sent by host
 - Transmit data to a host computer, via communications link
 - Select type of communications link
 - Set, change, and retrieve reader configuration settings

3 Document Audience

This document is designed to guide Code Reader users and Developers that wish to customize the behavior of the Reader. The commands outlined are targeted at manipulating data, such as removing characters, translating one set of characters to another, and providing different types of output, based on the type of barcode read, appending characters, tabs or new lines, etc.

4 Document and Coding Conventions

This document employs the following conventions to aid in readability:

- Words that are part of the application development description use the `Courier New` font
- Code examples use the **`bold Courier New font in blue`**
- Variable names that must be supplied by the programmer are: `Courier New` font and are enclosed in relational signs; for example: `<variable_name>`.

The Code Reader JavaScript library uses the following naming conventions:

- Identifiers: Mixed-case with a capital letter, where words join (soCalledCamelCase), e.g., `nasaSpaceShuttle`, `httpServer`, `codeXml`
- Acronyms and other initialisms are capitalized like words, e.g., `CIMCE`, `CPU`
- Variables and Properties: Initial lower case, e.g., `thisIsAVariable`, `thatIsAProperty`
- Classes (i.e., constructors): Initial capital, e.g., `AClassIsCapitalizedCamelCase`
- Functions: Initial lower case, similar to variables and properties
- Unit of measure: Suffix to name, separated from name by underscore, using correct case when it's significant, e.g., `offset_pixels`, `width_mm`, `power_mW`, `powerRatio_dB`



5 Related Documents

Code readers are controlled by a large set of configuration settings that define the behavior of the reader. These settings are described in Code documents D026166 and D027153. Please familiarize yourself with the use of these configuration settings.

Note: Please visit Code's website at: <http://www.codecorp.com> to obtain documents D026166 and D027153.

6 Related Utilities

USB Virtual Com Driver (C002712): A software driver that creates a virtual COM port for a USB-cabled reader. This driver enables the reader to be used by a computer program that requires input from a serial device, while being connected to a USB port.

CR8200 Utility V 2.xx (C002640): Provides host access to all CR82x functionality, including file transfer between host and reader. Valid communication modes are USB Downloader, USB Virtual Com, and RS232.

File Uploader (C002880): Utility to transfer files from the reader to the host PC. Valid communication modes are RS232 (115 Baud), USB Downloader mode, and USB Virtual Com.

These utilities are available at: <http://www.codecorp.com/downloads.html>

7 JavaScript Resources

While you will not need to be a JavaScript expert, some JavaScript knowledge is required to use Rules, based on this document. This document is not a JavaScript manual. While there are a number of books on JavaScript programming, the following sources provide JavaScript reference books and online documents you may find useful:

- [*JavaScript: The Definitive Guide*](#)
by David Flanagan
- [*JavaScript, A Beginner's Guide, Third Edition \(Beginner's Guide\)*](#)
by John Pollock
- [*JavaScript Demystified \(Demystified\)*](#)
by James Keogh.
- [*JavaScript \(TM\) in 10 Simple Steps or Less*](#)
by Arman Danesh
- <http://www.w3schools.com/jsref/default.asp>
- <http://javascript.internet.com/>
- <http://www.javascript.com/>

8 Regular Expression Resources

You will not need to be a Regular Expression expert, but some Regular Expression knowledge is required to use Rules, based on this document. This document is not a Regular Expression manual. While there are a number of books on Regular Expression use, the following sources provide Regular Expression reference books and online documents you may find useful:



- [Mastering Regular Expressions](#)
by Jeffrey E F Friedl
- [Beginning Regular Expressions \(Programmer to Programmer\)](#)
by Andrew Watt
- http://en.wikipedia.org/wiki/Regular_expression#Basic_concepts
- <http://www.regular-expressions.info/>

9 Installing and Running a .codeRules.js script on the Code Reader

The Code-supplied JavaScript file .cra.js is written to check for and include a Code Rules file located on the reader. The file name must conform to the following regular expression: /[^]\.codeRules.*\.js/

The purpose for Code Rules files is primarily to provide override functions for decode attempt, decode, and onEvent.

10 Security

Code supplies an encryption utility for license protection:

- Each Code Reader contains a unique reader ID
- Select features of the reader are protected by license
- Code provides a license file that activates protected features
- A license file is required for each reader licensed to use protected features
- Third-party software licenses may also be protected, using the encryption utility

11 Debugging (See reader.debug)

12 Programming Concepts

To help the Developer create unique applications for the reader, Code provides an easy-to-use, object-oriented JavaScript Application Programming Interface (API). The Developer can create complex business applications with prompts and data-entry, through the JavaScript API.

The features of the programming interface include:

- Event handlers
- Symbol decoding
- Host communications
- Local data storage
- Code Reader configuration

In support of this interface, Code has provided three objects which support these features:

- Reader
- Storage
- Comm

There are also shell functions to provide utility functionality, not directly related to these objects. Using these features, the user can create robust, interactive, and sophisticated user applications.



A user-defined script may be configured as the default script, using the CR8200 configuration utility, or it may be run from the utility, without being configured as the default script. Please be aware that any JavaScript file that is run remains active (variables & functions) until the next reset, or until a different script is run.

13 Code JavaScript Objects

JavaScript objects have properties and methods. Code uses the properties to allow user customization of the JavaScript implementation on the reader. Methods, on the other hand, provide communication between the JavaScript engine and the reader firmware. All of the custom reader object properties and methods are described in the following sections:

13.1 Reader Object

The reader object provides application software access to various Code Reader functionality and information.

13.1.1 Callback Properties

Callback properties are functions that are provided by JavaScript assignments. Most callback properties have default behavior defined in `.cra.js`.

13.1.1.1 `reader.onDecodes(Decode Object Array decodes)`

The `reader.onDecodes` function provides processing control to the application program at the completion of a decode action. The `reader.onDecodes(decodes)` argument “decodes” is an array of Decode Objects.

Example:

```
reader.onDecodes = function(decodes)
{
    for(var i = 0; i < decodes.length ; ++i)
        comm.sendText(decodes[i].data); // send decoded data to the
host
    return true;
}
```

13.1.1.2 `reader.onEvent(Event Object event)`

The `onEvent` property of the reader object provides a way for the user to issue and handle simulated key press, release, and hold events, which will be processed by JavaScript. `onEvent` has an event object passed to it, which has two properties: `event.type`, and `event.input`. Values for `type` are `PRES(1)`, `HOLD(2)`, and `RELS(3)`, and the values for `input` are `TRIG(0)`, `WAKE(1)`, `STND(2)`, `INP0(3)`, and `INP1(4)`.

Example:

```
const TRIG = 0;
const PRES = 1;
reader.onEvent = function(event)
{
    if(event.type == PRES)
        reader.beep();
    return true;
}
```



13.1.1.3 reader.onRawData(string data)

The `reader.onRawData` property is called when the reader receives data in a non-packetized stream, like a configuration command issued from the host console.

Example:

```
reader.onRawData = function(data, len)
{
    if(len > 0)
        comm.sendText(data);
    else
        return false;
    return true;
}
```

13.1.1.4 reader.onConfigure(string config)

The `onConfigure` property of the Code Reader calls the specified function when the reader:

- Receives a configuration command from a communication port
- Decodes a configuration command from a code read by the Code Reader

The application uses this property as an event handler to:

- Receive notification of command processing

Prevent execution of a command

The function will not be called in response to a `reader.configure` call.

Return 0 to instruct the reader to process the command, or return 1 to suppress the command. When a command is suppressed, the firmware will not send any response to the host, but the JavaScript application may provide its own response to the host.

The `onConfigure` property is a blocking call, which means other tasks, such as the `reader.tick1Hz` or `reader.periodic`, aren't guaranteed to work, if called within the `onConfigure` property.

Example:

```
reader.onConfigure = function(confString)
{
    if(confString.match("JSJSG") === null)
        return false;
    return true;
}
```

13.1.1.5 reader.onPowerChange(int mode)

The `reader.onPowerChange` property is called when the reader requests permission to change between active (0), standby (1), sleep (2), and off (3) modes. The user can prevent the mode from changing by returning "false", or can accept the change by returning "true".



Example:

```
reader.onPowerChange = function(mode)
{
  if(mode == 2)
  {
    comm.sendText("No Sleep For You");
    return false;
  }
  return true;
}
```

13.1.1.6 reader.onBatteryChange(void)ⁱ

The `reader.onBatteryChange` property is not yet implemented.

13.1.1.7 reader.periodic(void)

The `reader.periodic` function property is called on a set periodic rate. This rate is modifiable via the `JSPMIT` parameter. The default value is 1 Hz.

Example:

```
reader.periodic = function()
{
  comm.sendText("Some Time Has Passed");
  return;
}
```

13.1.1.8 reader.tick1Hz(void)

The `reader.tick1Hz` function property is called once every second. This is a set rate and is not modifiable.

Example:

```
reader.tick1Hz = function()
{
  comm.sendText("Some Time Has Passed");
  return;
}
```

13.1.2 Informational Properties

Informational properties are properties that provide information about the reader when called.

13.1.2.1 reader.charging

The `reader.charging` property is not yet implemented.

13.1.2.2 reader.green

The `reader.green` property is not yet implemented.

13.1.2.3 reader.amber

The `reader.amber` property is not yet implemented.

13.1.2.4 reader.red

The `reader.red` property is not yet implemented.



13.1.2.5 reader.black

The `reader.black` property is not yet implemented.

13.1.2.6 reader.softwareVersion

The `softwareVersion` property of the reader object contains a read-only string, comprising the version number of the firmware currently running in the Code Reader.

Example:

```
swVersion = reader.softwareVersion;
comm.sendText("Version X.X: " + swVersion);
```

13.1.2.7 reader.baseModel

The `baseModel` property of the reader object contains a read-only string, containing the Code Reader base model from the locked flash memory, ("CR1500"), etc.

Example:

```
rbmod = reader.baseModel
```

13.1.2.8 reader.model

The `model` property of the reader object contains a read-only string, containing the Code Reader model from the locked flash memory, ("2MD0"), etc.

Example:

```
rmod = reader.model
```

13.1.2.9 reader.readerId

The `reader ID` property of the reader object contains a read-only string, containing the Code Reader unique ID from the locked flash memory, ("FFFFFFFFFFFFFF"), etc.

Example:

```
rid = reader.readerId;
```

13.1.2.10 reader.hardwareVersion

The `hardware version` property of the reader object contains a read-only string, containing the Code Reader hardware revision number of the reader's hardware, ("00"), etc.

Example:

```
hwVersion = reader.hardwareVersion;
comm.sendText("Version XX: " + hwVersion);
```

13.1.2.11 reader.cabled

The `cabled` property of the reader object contains a read-only Boolean value, which indicates whether the reader is cabled or wireless.

Example:

```
if(reader.cabled == true)
    reader.indicateError();
```



13.1.2.12 reader.locked

The `locked` property of the reader object contains a Boolean value, which indicates whether the reader is locked or not. The `locked` property also supports unlocking by assigning the `reader.locked` property the valid unlock string.

Example:

```
reader.locked = "magic unlock string";
if(reader.locked == true)
    reader.indicateError();
```

13.1.2.13 reader.debug

The `debug` property of the reader object contains an integer value, which indicates the reader's current debug level.

Example:

```
dbglvl = reader.debug;
reader.debug = 1;
```

13.1.2.14 reader.decodes

The `decodes` property of the reader object contains a `decodes` object, which corresponds to the previous `decodes` object passed to the `onDecodes` callback.

Example:

```
dcds = reader.decodes;
```

13.1.2.15 reader.bdAddr

The `reader.bdAddr` property is not yet implemented.

13.1.3 Reader Methods

13.1.3.1 reader.runScript

The `runScript` method instructs the Code Reader to schedule the load, compile, and execution of the specified JavaScript. The Code Reader schedules execution of the script immediately after the currently executing event handler or main script completes. The `runScript` method does not include a mechanism to return to the calling script. Running a script causes the currently executing JavaScript context and runtime to close, then opens a new, fresh context and runtime. Any previously running script settings are lost when this happens. If more than one script need to work together, use an `include ("script");` statement within the main script.

Example:

```
var execute = reader.runScript("codeXml.js");
```



13.1.3.2 reader.configure

The configure method instructs the Code Reader to execute a configuration command. It supersedes readSetting, writeSetting, defaultSettings, and saveSettings. Please review configuration settings in the CR8200 ICD/CCD, (D026160) for specific details.

Example:

```
reader.configure("\CDOPPR2") // Sets maximum 2 barcode decodes per read
```

13.1.3.3 reader.beep

The beep method causes the Code Reader to beep. The user can call the command with one (required) to four arguments, which are as follows:

`reader.beep(numberOfBeeps, beepOnTime, beepOffTime, delayBeforeBeep).`

Example:

```
// beep 6 times, 250ms on, 1S off each beep, no delay before beep
reader.beep(6, 250, 1000, 0);
```

```
// beep once with the default values
reader.beep(1);
```

13.1.3.4 reader.vibrate

`reader.vibrate` can be called without arguments, which gives the default vibrate, or can be customized by entering values for number of vibration pulses, time on, time off, and the delay before beginning (in milliseconds).

Example:

```
// vibrate 3 times, 500ms on, 500ms off, 100ms delay
reader.vibrate(3, 500, 500, 100);
```

```
// vibrate once with default values
Reader.vibrate();
```

13.1.3.5 reader.indicateGoodRead

This method performs the same indication that a valid decode would produce. The default behavior is to turn on the green LED for 500mS and beeps once.

Example:

```
reader.indicateGoodRead();
```

13.1.3.6 reader.indicateError

This method turns on the red error LED and performs the same indication that an error would produce. The default behavior is to beep three times.

Example:

```
reader.indicateError();
```



13.1.3.7 reader.shiftJisToUnicode

This method converts a valid shiftJIS string to unicode, and returns the resulting string.

Example:

```
myUnicodeString = reader.shiftJisToUnicode(shiftjisStr);
```

13.1.3.8 reader.unicodeToShiftJis

This method converts a valid Unicode string to a shiftJIS encoding, and returns the resulting string.

Example:

```
uniString = reader.unicodeToShiftJis(unicodeStr);
```

13.1.3.9 reader.indicateData

13.1.3.10 reader.indicateWireless

13.1.3.11 reader.indicateBattery

13.1.3.12 reader.getKeyboardStatus

This method returns an integer representation of a byte value which corresponds to the current keyboard status. The lowest order bit represents a keyboard “caps lock”. The next bit represents a keyboard “num lock”.

Example:

```
keyStat = reader.getKeyboardStatus();
```

13.1.4 Reader Decodes Object

The decodes object is provided by the onDecodes function. The argument is an array of decode objects (see onDecodes). The following properties are provided:

13.1.4.1 decodes[i].decoderType

13.1.4.2 decodes[i].valid

The decoder marks decoded data as valid or not valid. This flag can be used to signal to the system that data has already been handled or ignore it.

Example:

```
var validDecode = decodes[i].valid;
```

13.1.4.3 decodes[i].x

The `decode.x` property is a read-only property that defines the horizontal position, in pixels, of the barcode that was just decoded, in relation to the entire image captured by the reader and analyzed by the decode engine. The value for `decode.x` can be positive or negative, based on the fact that 0 is the center of the image.

Example:

```
var hPos = decodes[i].x;
```



13.1.4.4 decodes[i].y

The `decode.y` property is a read-only property that defines the vertical position, in pixels, of the barcode that was just decoded, in relation to the entire image captured by the reader and analyzed by the decode engine. The value for `decode.y` can be positive or negative, based on the fact that 0 is the center of the image.

Example:

```
var vPos = decodes[i].y;
```

13.1.4.5 decodes[i].bounds

The `bounds` property is an array of four sets of x- and y-coordinates that give information about the position of the last decoded barcode. The four points (x, y) indicate the position in pixels of the four corners of the barcode from the upper left (0, 0) position in the captured image as integers. Be aware, in some readers the image is rotated 90°, when displayed in CortexTools.

//The x and y coordinates of the top right corner of the barcode in the image:

```
pos.tR = (decode.bounds[0].x, decode.bounds[0].y)
```

//The x and y coordinates of the top left corner of the barcode in the image:

```
pos.tL = (decode.bounds[1].x, decode.bounds[1].y)
```

//The x and y coordinates of the bottom left corner of the barcode in the image:

```
pos.bL = (decode.bounds[2].x, decode.bounds[2].y)
```

//The x and y coordinates of the bottom right corner of the barcode in the image:

```
pos.bR = (decode.bounds[3].x, decode.bounds[3].y)
```

13.1.4.6 decodes[i].time

The `decode.time` property is a read-only property that defines the amount of time, in milliseconds, it took the decode engine to decode the barcode that was just analyzed.

Example:

```
var decTime = decodes[i].time;
```

13.1.4.7 decodes[i].symbology

The `decode.symbology` property is a read-only property that contains the symbology index assigned by Code Corporation. Valid values for `symbology` are defined in Appendix 5.3.

Example:

```
var symbology = decodes[i].symbology;
```



13.1.4.8 `decodes[i].quality_percent`

The `decode` quality percent property is a read-only property that defines an internally-defined image quality value determined by the decode engine, while analyzing the captured image. This is not the quality of the printed or displayed barcode, but rather the quality of the captured image for use in decoding.

Example:

```
var quality_percent = decodes[i].quality_percent;
```

13.1.4.9 `decodes[i].aimSymbology`

Property is a read-only property that gives the first character of the AIM (Automatic Identification and Mobility) symbology, determined by the decode engine of the barcode that was just decoded. More information on AIM Standards can be found at the Association for Automatic Identification & Mobility website.

13.1.4.10 `decodes[i].symbologyModifier`

The `decode.symbologyModifier` property is a read-only property that contains symbology modifier information for the barcode decoded by the decode engine. Valid values for `symbologyModifier` are based on an old proprietary decoder and are defined in the Appendix. The values are the same as `aimModifier` for all symbologies, except UPC.

Example:

```
var symbologyModifier = decodes[i].symbologyModifier;
```

13.1.4.11 `decodes[i].aimModifier`

The `decode.aimModifier` property is a read-only property that gives the AIM (Automatic Identification and Mobility) Modifier, determined by the decode engine of the barcode that was just decoded. More information on AIM Standards can be found at the Association for Automatic Identification & Mobility website.

Valid values for `aimModifier` are defined in section 5.3.

13.1.4.12 `decodes[i].linkage`

The `decode.linkage` property is a read-only property that gives information about the linking code between the segments that make up a composite barcode, if a composite barcode was just decoded by the decode engine. This property is null if the decoded barcode was not a composite barcode.

Example:

```
var link = decodes[i].linkage;  
if( link === null ) print("not composite\n");
```

13.1.4.13 `decodes[i].saPosition`

In order to handle larger messages, that are practical, in a single symbol, a data message can be distributed across several QR Code symbols. Up to 16 QR Code symbols may be appended in a structured format to convey more data. If a symbol is part of a structured append, this will be indicated by a structured append block in barcode text.



The position index of the symbol within the set of QR Code symbols in the structured append format and is an integer between 1 and 16 (including the boundaries) in string format.

Example:

```
var numSplits = decodes[i].saTotal;
var idx      = decodes[i].saPosition;
print("this is " + idx + " of " + numSplits + "\n");
```

13.1.4.14 decodes[i].saTotal

The total amount of the symbol within the set of QR Code symbols in the structured append format. It's an integer between 2 and 16 (including the boundaries) in string format.

Example:

```
var numSplits = decodes[i].saTotal;
```

13.1.4.15 decodes[i].saParity

Parity Date: It shall be an 8-bit byte value obtained by using the GetParity method, with the original input data as its parameter. It is identical in all symbols in the structured append, enabling verification that all symbols read form part of the same structured append message.

Example:

```
var parity = decodes[i].saParity;
```

13.1.4.16 decodes[i].isConfig

The decoder marks decoded data that matches configuration code formatting with isConfig true. JavaScript must then decide the appropriate way to handle or not handle the code.

Example:

```
var isConfigCode = decodes[i].isConfig;
```

13.1.4.17 decodes[i].decodeOutputFormat

The decode.decodeOutputFormat property is a read-only property that gives the type of output the JavaScript should expect.

Value	Definition
0	<u>Raw output</u> : No formatting applied by the decoder
1	<u>Customer formatted data</u> : This is formatting applied from a customer-supplied .parse file. These files can be generated using the DL/ID tab of CortexTools
2	<u>JSON formatted data</u> : This is data formatted in JavaScript Object Notation (JSON). Currently, only DL/ID data can be formatted in JSON by the decoder. For more information about JSON, see JSON.org

13.1.4.18 decodes[i].data (type string)

The decode.data property is a read/write value representing the payload of the barcode that has just been decoded. This data may also include data processing, checksum information, or data formatting, based on settings made by the user for the decode engine. If the decode



engine has no special handling settings applied, this information should match the string that was used to encode the barcode before the barcode was printed or displayed.

Example:

```
var decodeText = decodes[i].data;
```

13.2 Storage Object

The storage object provides application software access to Code Reader file storage. Files are written to storage by the `storage.write` method and by downloading from the host.

Note: Names of files can be up to 200 printable ASCII characters. For compatibility with host file systems, Code recommends you do not use characters that are reserved by host operating systems: `/`, `\`, `:`, `?`, `*`, `[`, `]`, `'`, `"`, etc. Files are stored in UTF8 format, which encodes Unicode characters in one or more bytes each. Any file that uses multi-byte characters coming from the host must be encoded in UTF-8, in order for the stored file to be correctly handled in UTF-8. JavaScript converts these files to UTF-16 when reading them from the file system.

13.2.1 Informational Properties

13.2.1.1 `storage.isFull`

The `storage.isFull` property is a read-only Boolean value; true if storage is full and cannot be added to; otherwise, false.

13.2.1.2 `storage.fullness_percent`

The `storage.fullness_percent` property is a read-only integer containing the percent of storage in use.

13.2.2 Storage Methods

The following section documents the methods defined for the Code Reader storage object.

In this section, the examples use elements of a time card application that assumes time card records are maintained as files, organized by employee number. The naming convention for the time card records is `TimeCard<employee_number>`.

13.2.2.1 `storage.read`

The `storage.read` method reads a file. This file is converted to UTF-16 for use in JavaScript.

Example:

```
var timeInfo = storage.read("TimeCard1");
```

13.2.2.2 `storage.write`

The `storage.write` method writes a file to storage. If the file does not exist, the Code Reader creates it. If there was an existing file of the same name, it is replaced. If the file contains multi-byte characters, it must be encoded in UTF-8.



Example:

```
var employee57Info;  
storage.write("TimeCard57", employee57Info);
```

13.2.2.3 storage.append

The `storage.append` method adds data to the end of a file.

Example:

```
var extraInfo = "extra info";  
storage.append("timeCard57", extraInfo);
```

13.2.2.4 storage.rename

The `storage.rename` method renames a file.

Example:

```
storage.rename("newEmployeeTime", "timeCard58");
```

13.2.2.5 storage.erase

The `storage.erase` method erases a file.

Example:

```
storage.erase("newEmployeeTime");
```

13.2.2.6 storage.uploadFile

The `storage.uploadFile` method uploads a file's name that match the RegEx, to the host, over the current active host comm port.

Example:

```
storage.uploadFile("timeCard58");
```

13.2.2.7 storage.defragment

`storage.defragment` gathers split or scattered file blocks into contiguous blocks, as much as possible.

Example:

```
storage.defragment();
```

13.2.2.8 storage.findFirstInternal

The `storage.findFirstInternal` method finds the first file in the file system.

Example:

```
var name = storage.findFirstInternal();
```

13.2.2.9 storage.findNextInternal

The `storage.findNextInternal` method finds the next file in the file system after a previous call to `storage.findFirstInternal`, `storage.findFirst`, `storage.findNextInternal`, or `storage.findNext`.

Example:

```
var name = storage.findNextInternal();
```



13.2.2.10 storage.findFirst

The `storage.findFirst` method locates the first file, where the name matches a regular expression specified in the call parameter.

Example:

```
var name = storage.findFirst(regex);
```

13.2.2.11 storage.findNext

The `storage.findNext` method locates the next file, where the name matches the regular expression, specified in the expression parameter of a previous `storage.findFirst` call. The matching names are not ordered, but they will not be repeated; a `findFirstNative - findNextNative` sequence will return all matching files, provided that there are no other intervening storage method calls. One can put the files into an array and use JavaScript's `sort` method when one needs them ordered.

Example:

```
var name = storage.findNext();
```

13.2.2.12 storage.size

The `storage.size` method returns the size of a file in bytes.

Example:

```
var fileSize = storage.size("timeCard57");
```

13.2.2.13 storage.getHeader

The `storage.getHeader` method returns the first multi-line comment block from a JavaScript file. This includes encrypted files, if the proper developer key is installed.

Example:

```
var headerInfo = storage.getHeader("timeCard57");
```

13.2.2.14 storage.listFiles

The `storage.listFiles` method returns a text string containing the names of the files on the file system if it is given no input. If the method is given a text string as an input, it will list all of the files that have a substring which matches the given string.

Example:

```
var allFilesOnFS = storage.listFiles();  
var filesThatContainFile = storage.listFiles("File");
```

13.2.3 Data in Code Reader Local Storage

The application development environment provides program access to Code Reader local storage through the storage object. Data is maintained in storage as named objects called files. The CR8200 Utility2 can transfer host data into a Code Reader file. The Code Reader application can also store data in files.

The name of a Code Reader file may contain any printable ASCII characters.



Use the erase and write methods of the storage object to manage files. Use the findFirst and findNext methods to locate files. Use the read method to access a file or the upload method to send it to the host.

NOTE: `storage.findFirst` and `storage.findNext` are implemented by Code in the library file `.jselib.js`. The scanner supplies the `storage.findFirstInternal` and `storage.findNextInternal` functions.

13.3 Comm Object

The Code Reader application development environment defines a host communication comm object to support communications with a host resident application. For example, the CR8200 Utility (section 6) is a host resident utility that communicates with the Code Reader for downloading files to the Code Reader.

From the host computer's view, the Code Reader is a serial device accessible through a serial or USB port. Code Reader configuration settings define the active host communications port. The JavaScript program transfers data to the host by writing to the Code Reader host communications port, using methods defined by the Code Reader comm object. The Code Reader host communications implementation supports two basic styles of communication: raw text and packets. The Code Reader host communications implementation also supports a set of native protocols.

13.3.1 Informational Properties

13.3.1.1 `comm.isConnected`

The `isConnected` property of the comm object contains a read-only Boolean, specifying the host connection status. Possible connection values are:

True: Reader is connected to the host.

False: Reader is not connected to the host.

Example:

```
if( comm.isConnected == false )
    comm.connect;
```

13.3.2 Comm methods

13.3.2.1 `comm.connect`

The connect method instructs the Code Reader communication driver to attempt to establish a connection.

Note: Not applicable to wired products

13.3.2.2 `comm.disconnect`

The disconnect method instructs the Code Reader communication driver to disconnect from the host.

Note: Not applicable to wired products



13.3.2.3 comm.sendPacket

The sendPacket method instructs the Code Reader to send a data packet to the host, via the communications port currently specified by the active Code Reader communication settings. The Code Reader creates a packet formatted according to the active Code Reader packet protocol configuration setting.

For a discussion of data packets, see the Code Interface Configuration Document, which can be downloaded from <http://www.codecorp.com>.

Example:

```
var myPacket = "my data packet";
comm.sendPacket(myPacket);
```

13.3.2.4 comm.sendText

The sendText method instructs the Code Reader to send arbitrary text (which may include NULL characters) to be sent via the active communication port; the text will be sent "raw" regardless of the reader comm mode settings. This method buffers the data until <include more details>

Example:

```
var myPacket = "my data packet";
comm.sendText(myPacket);
```

13.3.2.5 comm.sendMessage

Sends a message to the host using protocol_sendMessage.

Example:

```
var myMsg = "my data message";
comm.sendMessage(myMsg);
```

13.3.2.6 comm.getSendStatus

The comm.getSendStatus property is not yet implemented.

13.3.2.7 comm.sendUsbScanCode

The comm.sendUsbScanCode method is only available when the reader is in keyboard mode and is designed to allow the user to send keyboard scancodes to the host, as if they had come from a keyboard. Each scancode consists of 8 bytes of data, which map to the language set in the keyboard, or selected by the user, by means of configuration settings.

Ex. comm.sendUsbScanCode(0, 0, 0, 0, 0, 0, 0, 0); which represents a key release.

13.3.2.8 comm.sendUsbScanCodes

The comm.sendUsbScanCodes method allows the user to send multiple scancodes (see 13.3.2.7) to the host at once, when the reader is in keyboard mode. This might represent an upper-case letter, or control character sequence to be sent together. These scancodes will use the scancode delays set for scan codes sent by the reader in keyboard mode. Each group of 8 bytes is sent as a HID report to the host, the same way a keyboard does. If a user wanted to send a capital 'a' he would send the first 8 bytes with only the shift key pressed, then sent



the next 8 bytes with the shift key and the 'a' key pressed, then send a group of 8 bytes with all the values as '0' representing a key release.

```
Ex. comm.sendUsbScanCodes(2,0,0,0,0,0,0,0,2,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0);
```

13.3.2.9 comm.sendHidReport

The comm.sendHidReport is not yet implemented.

13.4 Shell Functions

13.4.1 help([name])

The help function displays available shell commands. Optionally, [name] can be any shell command for more detailed information on [name].

Example:

```
help(include);
```

13.4.2 include(name)

The include function executes the included script inline.

Example:

```
// Adds the definitions in myScript.js to the application.
// The definitions become part of the "including" script.
include("myScript.js");
```

13.4.3 print(string) – see reader.print(string)

The print function is an alias for the reader.print(string) method.

Example:

```
print("some random text");
```

13.4.4 gc()

The gc function cleans up memory that has been allocated, but is no longer needed by the runtime environment. This function is processor-intensive, so its use can degrade performance.

Example:

```
gc();
```

13.4.5 sleep_ms(time_ms)

The sleep function performs an inline blocking delay for time_ms number of milliseconds.

Example:

```
sleep(1000);
```



13.4.6 format(string format, ...)

format takes a formatting string as an argument, and an argument list, to apply to the format and then sends the formatted data back to the JavaScript, as the return value.

Ex.

```
{
    var myString = "";
    format("This is an int %d and this is a string %s", 274, "Two hundred
    seventy four", &myString);
    print(myString);
}
```

13.4.7 wdt_pet()

Long processes may require the firmware watchdog to be petted during the operation. If the watchdog times out during a processor-intensive operation, the reader will reboot and an error will be logged in the error log. To prevent the reader from rebooting during a processor-intensive section, the firmware watchdog timer needs to be petted. `wdt_pet()` takes a single argument indicating the number of seconds before the watchdog timer times out.

Example:

```
wdt_pet(1);
```

13.4.8 logMessage(string)

The `logMessage` function writes the given string into the log.

Example:

```
logMessage("Writing to log");
```

13.5 Pre-defined Scripts

The Code Reader core JavaScript application provides a simple method for handling the most common tasks. This method will allow people with minimal programming knowledge to customize the product to their needs, using a subset of JavaScript and Regular Expressions. Regular Expressions, combined with JavaScript String methods, such as `.match` and `.replace` become powerful data manipulators.

When JavaScript is enabled, the reader first loads the default library file `“.jseLib.js”`. JavaScript then loads `“.startup.js”`, which contains the include statement `“include(.default.js);”`. The default `“.default.js”` file will `“include(.cra.js)”`. These files provide the basic JavaScript functionality for the reader.



13.5.1 .jseLib.js library functions

13.5.1.1 storage.findFirst(regex)

The implementation of `storage.findFirst(regex)` is to scan all files looking for a regex match and stop on the first match.

13.5.1.2 storage.findNext()

The implementation of `storage.findNext()` is to scan all remaining files from the previous `storage.findFirst()` / `storage.findNext()` looking for a regex match established in the `storage.findFirst(regex)` and stop on the next match.

13.5.1.3 reader.setInterval(func, interval)

The `setInterval` method calls a function or evaluates an expression at specified intervals, in seconds. `setInterval` method will continue calling the function until `clearInterval` is called.

The ID value returned by `setInterval` is used as the parameter for the `clearInterval` method.

Example:

```
intervalId = reader.setInterval(function, interval_sec);
```

13.5.1.4 reader.clearInterval(intervalId)

The `clearInterval` method removes the instance of `setInterval` that has the handle `intervalId`.

Example:

```
reader.clearInterval(intervalId);
```

13.5.1.5 reader.setTimeout(func, delay)

The `reader.setTimeout()` method calls a function after a delay number of seconds. The function cannot be a code-predefined object method.

The ID value returned by `reader.setTimeout()` is used as the parameter for the `reader.clearTimeout()` method.

Example:

```
timeoutId = reader.setTimeout(function, timeout_sec);
```

13.5.1.6 reader.clearTimeout(timeoutId)

The `clearTimeout` method removes the instance of `setTimeout` that has the handle `timeoutId`.

Example:

```
reader.clearTimeout(timeoutId);
```

13.5.1.7 reader.tick1Hz() – default implementation

This implementation is used to manage the internal “interval” timers.



13.5.2 cra.js

Code provides default reader behavior by implementing special features in JavaScript. These special features are in “.cra.js”.

13.5.2.1 reader.onEvent(event)

This function checks for a function variable “rules_onEvent” and if it is not “null”, it will call that function. If “rules_onEvent” returns false, the function will handle the call.

13.5.2.2 reader.onDecodeAttempt(count)

This function checks for a function variable “rules_onDecodeAttempt” and if it is not “null”, it will call that function. Otherwise, it will always return true, allowing the decodes array to be processed.

13.5.2.3 reader.onDecode(decode)

This function checks for a function variable “rules_onDecode” and if it is not “null”, it will call that function. If “rules_onDecode” does not return false, the function will handle the decode.

13.5.2.4 reader.onDecodes(decodes)

This function checks for a function variable “rules_onDecodes” and if it is not “null”, it will call that function. If “rules_onDecodes” does not return false, and “rules_onDecodeAttempt” does not return false, this function will process the decodes by calling `reader.onDecode()` for each decode object in the decodes array.

13.5.3 .codeRules.js

Code provides the ability to define a “.codeRules.js” file, which can specify functionality for the listed methods.

13.5.3.1 rules_onDecodes

The decodes array object (Section 13.1.4) is passed into the function and may be returned by the function. If a Boolean false is returned by the function, the reader will prevent further processing of the decode. The Code Reader will initially indicate a good read as soon as a bar code is found.

The `rules_onDecodes()` function must return either the decodes array object or the value ‘false’. Returning ‘false’ will stop further action by the `reader.onDecodes()` function.

This example illustrates using regular expression to replace each uppercase letter ‘A’ in the decode string with lowercase letter ‘a’. The properties of the Decode object are described in Section 4.

Example:

```
rules_onDecodes = function(decodes)
{
    decodes[0].data.replace(/A/g, "a");
    return decodes;
};
```



This example shows the use of a regular expression that matches a specific pattern of decode data, then the use of a JavaScript method to remove the last digit from the matched decode data.

Example:

```
rules_onDecodes = function(decodes)
{
    if(decodes[0].data.match(/^([0-9]{9})$/g) != null)
        decodes[0].data = decodes[0].data.substring(
            0, decodes[0].data.length - 1);
    return decode;
};
```

13.5.3.2 rules_onDecodeAttempt

The `rules_onDecodeAttempt()` function is run when a call to the default `reader.onDecodes()` is made. `rules_onDecodeAttempt()` is passed the length of the `decodes` array (Section 13.1.4). The function can then return “true” if this `decodes` array is to be processed, or “false” to skip the decoding of the array.

13.5.3.3 rules_onDecode

The `rules_onDecode()` function is run when a call to the default `reader.onDecodes()` is made and the `rules_onDecodeAttempt()` returns “true”. `rules_onDecode()` is passed an individual `decode` object, and is called for each `decode` object contained in the `decodes` array.

The `rules_onDecode()` function must return either the `decode` object or the value “false”. Returning “false” will stop further action by the `reader.onDecode()` function.

13.5.3.4 rules_onEvent

`rules_onEvent()` will be called whenever a `userEvent` is utilized. The user event numbers are restricted to `event_user0` through `event_user9`. The most common use of this will be to implement some logic, based on the press of a button or to send a read-failure notification. Users can pre-handle any of the following events. Returning a “false” lets the firmware process the event, while returning a “true” tells the firmware that the event has been handled and needs no further processing.

In the example below, the reader is first set to intercept a button press. The reader will then send a `codeXml` enter keypress over the comm port. See the CR200 CCD, Code Document Number D026160 for more information.



Example:

```
var enter = "\x01X\x1ean//n\x04";
rules_onEvent = function(event)
{
    if (event.type == buttonPress)
    {
        comm.sendPacket(enter);
        return true; // tell the firmware that we handled the
press
    }
};
```

13.5.3.5 Optional Global variables

The writer can include global variables in JavaScript Rules file, outside any of the pre-defined functions. These variables will be instantiated when the reader starts, as the JavaScript engine starts.

14 Appendix

14.1 Sending Keystrokes (CodeXml)

The Code Reader products are often connected to a PC using keyboard input. The data contained in the bar code is simply “typed” into the PC application. It is also often required to send a certain key to the application, such as an “enter” key. Please note that an “enter” key is not the same as an ASCII carriage return (0x13).

To add an enter suffix, one can use the following format, where the /n represents the enter key. A full list of available keys are listed below, which can be substituted for the ‘/n’.

```
enter = “\x01Y\x1ean/2F/2Fn\x04”;
```

```
decode.data = decode.data + enter;
```

Characters	Key
/a	Toggle Alt
/c	Toggle Ctrl
/s	Toggle Shift
/w	Toggle Windows Logo
/u	Up arrow
/l	Left arrow
/r	Right arrow
/d	Down arrow
/t	Tab



Characters	Key
/z	Delete
/e	Esc
/n	Enter
/v	End
/b	Backspace
/i	Insert
/p	Page up
/x	Page down
/h	Home
/,	500 ms delay
/0 - /9	Number pad
/f1 - /f12	Function keys
/	

14.2 Glossary and Acronyms

Term **Definition**

Code Data: Data resulting from the decode process after data capture or bar code read.

Smart Quote: Previously formatted quotation marks, usually found in a word processing program.

Consume: Used with no return value by the user-defined application or firmware.

14.3 Supported JavaScript Core

This is a list of JavaScript that is supported in the 'rules.js' functions:

Objects, Methods, and Properties

Array

Boolean

Date

Function

Math

Number

Object

Packages

RegExp

String

sun

Top-Level Properties and Functions

decodeURI

decodeURIComponent

encodeURIComponent

encodeURIComponent

eval

Infinity

isFinite



isNaN

NaN

Number

parseFloat

parseInt

String

undefined

Statements

break

const

continue

do...while

export

for

for...in

function

if...else

import

label

return

switch

throw

try...catch

var

while

with

Operators

Assignment Operators

Comparison Operators

Arithmetic Operators

% (Modulus)

++ (Increment)

-- (Decrement)

- (Unary Negation)

Bitwise Operators

Bitwise Logical Operators

Bitwise Shift Operators

Logical Operators

String Operators

Special Operators

?: (Conditional operator)

, (Comma operator)

delete

function

in

instanceof

new

this

typeof

void



14.4 Symbology Identifier Values

Symbology Name	CSI	CSI Mod	ID Char	ID Mod
Interleaved 2 of 5 (checksum not checked)	17	48	73	48
Interleaved 2 of 5 (Checksum checked and sent)	17	49	73	49
Interleaved 2 of 5 (Checksum checked and stripped)	17	51	73	51
Code 39 (checksum not checked)	18	48	65	48
Code 39 (Checksum checked and sent)	18	49	65	49
Code 39 (Checksum checked and stripped)	18	51	65	51
Code 39 Full ASCII (checksum not checked)	18	52	65	52
Code 39 Full ASCII (Checksum checked and sent)	18	53	65	53
Code 39 Full ASCII (Checksum checked and stripped)	18	55	65	55
Code 128 (standard)	19	48	67	48
Code 128 (FNC1 in first character position)	19	49	67	49
Code 128 (FNC1 in second character position) (aka UCC/EAN 128)	19	50	67	50
UCC/EAN 128 (aka Code 128 (FNC1 in second character position))	19	49	67	49
Codabar (checksum not checked)	20	48	70	48
Codabar (Checksum checked and sent)	20	50	70	50
Codabar (Checksum checked and stripped)	20	54	70	54
Code93	21	48	71	48
Australia Post	29	97	88	97
Aztec	30	48	122	48
Aztec (FNC1 in first position)	30	49	122	49
Aztec (FNC1 after initial letter or pair of digits)	30	50	122	50
Aztec (ECI protocol implemented)	30	51	122	51
Aztec (ECI protocol implemented, FNC1 in first position)	30	52	122	52
Aztec (ECI protocol implemented, FNC1 after initial letter or pair of digits)	30	53	122	53
Aztec (Structured Append header included)	30	54	122	54
Aztec (Structured Append, FNC1 in first position)	30	55	122	55
Aztec (Structured Append, FNC1 after initial letter or pair of digits)	30	56	122	56
Aztec (Structured Append, ECI protocol implemented)	30	57	122	57
Aztec (Structured Append, FNC1 in first position, ECI protocol implemented)	30	65	122	65
Aztec (Structured Append, FNC1 after initial letter or pair of digits, ECI protocol implemented)	30	66	122	66
Data Matrix (ECC 000 – 140, not supported by CortexDecoder)	31	48	100	48
Data Matrix (ECC 200)	31	49	100	49
Data Matrix (ECC 200, FNC1 in 1st or 5th position)	31	50	100	50
Data Matrix (ECC 200, FNC1 in 2nd or 6th position)	31	51	100	51
Data Matrix (ECC 200 supporting ECI protocol)	31	52	100	52
Data Matrix (ECC 200, FNC1 in 1st or 5th position + supporting ECI protocol)	31	53	100	53
Data Matrix (ECC 200, FNC1 in 2nd or 6th position + supporting ECI protocol)	31	54	100	54
Straight 2 of 5 with 2-Bar Start/Stop	32	48	82	48



Straight 2 of 5 with 3-Bar Start/Stop	33	48	83	48
Japan Post	34	106	88	106
Dutch KIX Post	35	100	88	100
MSI Plessey	36	48	77	48
Maxi	37	48	85	48
PDF417 (Standard)	38	48	76	48
PDF417 (Support ECI. All characters 92 are doubled)	38	49	76	49
PDF417 (Basic Channel operation. Char 92 not doubled)	38	50	76	50
USPS PLANET	39	101	88	101
USPS POSTNET	40	116	88	116
QR (Model 1 symbol)	41	48	81	48
QR (ECI protocol not implemented)	41	49	81	49
QR (ECI protocol implemented)	41	50	81	50
QR (ECI protocol not implemented, FNC1 implied in 1st position)	41	51	81	51
QR (ECI protocol implemented, FNC1 implied in 1st position)	41	52	81	52
QR (ECI protocol not implemented, FNC1 implied in 2nd position)	41	53	81	53
QR (ECI protocol implemented, FNC1 implied in 2nd position)	41	54	81	54
Royal Mail 4 State Customer	42	114	88	114
GS1 DataBar Expanded	43	48	101	48
GS1 DataBar Expanded Stacked	44	48	101	48
GS1 DataBar Limited	45	48	101	48
GS1 DataBar Omnidirectional	46	48	101	48
GS1 DataBar Stacked / GS1 DataBar Stacked Omnidirectional	47	48	101	48
GoCode	48	71	88	71
UPC-A	49	65	69	48
UPC-E0	49	66	69	48
UPC-E1	49	67	69	48
EAN/JAN-8	49	68	69	52
EAN/JAN-13	49	69	69	48
EAN Bookland (EAN13 with 5 digits supplemental)	49	52	69	48
EAN Bookland (EAN13 without 5 digits supplemental)	49	69	69	51
UPC-A with 2-digit Supplemental	49	97	69	51
UPC-A with 5-digit Supplemental	49	48	69	51
UPC-E0 with 2-digit Supplemental	49	98	69	51
UPC-E0 with 5-digit Supplemental	49	49	69	51
UPC-E1 with 2-digit Supplemental	49	99	69	51
UPC-E1 with 5-digit Supplemental	49	50	69	51
EAN/JAN-8 with 2-digit Supplemental	49	100	69	51
EAN/JAN-8 with 5-digit Supplemental	49	51	69	51
EAN/JAN-13 with 2-digit Supplemental	49	101	69	51
EAN/JAN-13 with 5-digit Supplemental	49	52	69	51
Codablock 256: FNC1 not used (not supported by CD)	50	48	79	48
Codablock 256: FNC1 in first char position (unsupported by CD)	50	49	79	49



Codablock F: FNC1 not used	50	52	79	52
Codablock F: FNC1 in first character position	50	53	79	53
Codablock A (unsupported by CD)	50	54	79	54
Code11 (1-digit or 2-digit check characters checked and sent)	51	48	72	48
Code11 (Check character(s) checked and stripped)	51	50	72	50
Pharmacode	52	80	88	80
Matrix 2 of 5 (checksum not checked)	53	77	88	77
Matrix 2 of 5 (Checksum checked and sent)	53	48	88	48
Matrix 2 of 5 (Checksum checked and stripped)	53	49	88	49
NEC 2 of 5 (checksum not checked)	54	78	88	78
NEC 2 of 5 (Checksum checked and sent)	54	50	88	50
NEC 2 of 5 (Checksum checked and stripped)	54	51	88	51
Telepen	56	48	66	48
Trioptic Code 39	57	84	88	84
USPS 4CB (Intelligent Mail)	58	105	88	105
BC412	59	66	88	66
Micro PDF417	60	48	76	48
Han Xin	61	72	88	72
Composite CA	62	48	101	48
Composite CB	63	48	101	48
Composite CC	64	48	101	48
Code 32 (Italian Pharmacode)	65	108	88	108
Plessey	66	48	80	48
Hong Kong 2 of 5	67	104	88	104
Korea Post	68	107	88	107
UPU ID Tag	69	117	88	117
Micro QR	70	49	81	49
Canada Post	71	99	88	99
Code 49 (Standard)	72	48	84	48
Code 49 (FNC1 in 1st character position)	72	49	84	49
Code 49 (FNC1 in 2nd character position)	72	50	84	50
Code 49 (FNC2 in 1st character position)	72	52	84	52
Grid Matrix	73	103	88	103

ⁱ Applies to battery powered readers only.

