JavaScript Programming Guide

CR1500, CR1100, CR2700, CR8x7x, and CR5200

January 2021

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

Code Corporation, 434 West Ascension Way, Suite 300, Salt Lake City, UT 84123. www.codecorp.com

Table of Contents

JavaScript Programming Guide 1

CR1500, CR1100, and CR2700 1

1 Introduction 7

2 Document Purpose 7

3 Document Audience 7

4 Document and Coding Conventions 7

5 Related Documents 8

6 Related Utilities 8

7 JavaScript Resources 8

8 Regular Expression Resources 8

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

10 Security 9

11 Debugging (See reader.debug) 9

12 Programming Concepts 9

13 Code JavaScript Objects 10

13.1 Reader Object **10**

13.1.1 Callback Properties 10

13.1.1.1 reader.onDecodes(Decode Object Array decodes) 10

13.1.1.2 reader.onEvent(Event Object event) (CR1100, CR1500) 11

13.1.1.3 reader.onRawData(data, len) 11

13.1.1.4 reader.onConfigure(string config) 11

13.1.1.5 reader.onPowerChange(int mode) 12

13.1.1.6 reader.onBatteryChange(void) 12

13.1.1.7 reader.onTrigger(triggerButton, buttonAction)(CR2700) 12

13.1.1.8 reader.onDecodeAttempt(count) 13

13.1.1.9 reader.periodic(void) 13

13.1.1.10 reader.tick1Hz(void) 13

13.1.2 Informational Properties 13

13.1.2.1 reader.charging 14

13.1.2.2 reader.green 14

13.1.2.3 reader.amber 14

13.1.2.4 reader.red 14

13.1.2.5 reader.black 14

13.1.2.6 reader.softwareVersion 14

13.1.2.7 reader.baseModel 14

13.1.2.8 reader.model 14

13.1.2.9 reader.readerId 15

13.1.2.10 reader.hardwareVersion 15

13.1.2.11 reader.cabled 15

13.1.2.12 reader.locked 15

13.1.2.13 reader.debug 15

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

13.1.2.14 reader.decodes 16
13.1.2.15 reader.bdAddr 16
13.1.3 Reader Methods 16
13.1.3.1 reader.runScript 16
13.1.3.2 reader.configure 16
13.1.3.3 reader.beep 16
13.1.3.4 reader.vibrate 17
13.1.3.5 reader.indicateGoodRead 17
13.1.3.6 reader.indicateError 17
13.1.3.7 reader.shiftJisToUnicode 17
13.1.3.8 reader.unicodeToShiftJis 17
13.1.3.9 reader.indicateData 17
13.1.3.10 reader.indicateWireless 18
13.1.3.11 reader.indicateBattery 18
13.1.3.12 reader.getKeyboardStatus 18
13.1.3.13 reader.getLastImage 18
13.1.4 Reader Decodes Object 18
13.1.4.1 decodes[i].decoderType 18
13.1.4.2 decodes[i].valid 18
13.1.4.3 decodes[i].x 18
13.1.4.4 decodes[i].y 19
13.1.4.5 decodes[i].bounds 19
13.1.4.6 decodes[i].time 19
13.1.4.7 decodes[i].symbology 19
13.1.4.8 decodes[i].quality_percent 20
13.1.4.9 decodes[i].aimSymbology 20
13.1.4.10 decodes[i].symbologyModifier 20
13.1.4.11 decodes[i].aimModifier 20
13.1.4.12 decodes[i].linkage 20
13.1.4.13 decodes[i].saPosition 21
13.1.4.14 decodes[i].saTotal 21
13.1.4.15 decodes[i].saParity 21
13.1.4.16 decodes[i].isConfig 21
13.1.4.17 decodes[i].decodeOutputFormat 21
13.1.4.18 decodes[i].data (type string) 22

13.2 Storage Object 22
13.2.1 Informational Properties 22
13.2.1.1 storage.isFull 22
13.2.1.2 storage.fullness_percent 22
13.2.2 Storage Methods 22
13.2.2.1 storage.read 23
13.2.2.2 storage.write 23
13.2.2.3 storage.append 23
13.2.2.4 storage.rename 23
13.2.2.5 storage.erase 23
13.2.2.6 storage.uploadFile 23
13.2.2.7 storage.defragment 23
13.2.2.8 storage.findFirstInternal 24
13.2.2.9 storage.findNextInternal 24
13.2.2.10 storage.findFirst 24
13.2.2.11 storage.findNext 24
13.2.2.12 storage.size 24
13.2.2.13 storage.getHeader 24

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

13.2.2.14 storage.listFiles 25
13.2.3 Data in Code Reader Local Storage 25
13.3 Comm Object 25
13.3.1 Informational Properties 25
13.3.1.1 comm.isConnected 25
13.3.2 Comm methods 26
13.3.2.1 comm.connect 26
13.3.2.2 comm.disconnect 26
13.3.2.3 comm.sendPacket 26
13.3.2.4 comm.sendText 26
13.3.2.5 comm.sendData 26
13.3.2.6 comm.sendMessage 27
13.3.2.7 comm.getSendStatus 27
13.3.2.8 comm.sendUsbScanCode 27
13.3.2.9 comm.sendUsbScanCodes 27
13.3.2.10 comm.sendHidReport 28
13.4 Shell Functions 28
13.4.1 help(*[name]*) 28
13.4.2 include(name) 28
13.4.3 print (string) – see reader.print(string) 28
13.4.4 gc() 28
13.4.5 sleep_ms(time_ms) 28
13.4.6 format(string format, ...) 28
13.4.7 wdt_pet() 29
13.4.8 logMessage(string) 29
13.5 Pre-defined Scripts 29
13.5.1 .jseLib.js library functions 29
13.5.1.1 storage.findFirst(regex) 29
13.5.1.2 storage.findNext() 29
13.5.1.3 reader.setInterval (func, interval) 30
13.5.1.4 reader.clearInterval(intervalId) 30
13.5.1.5 reader.setTimeout (func, delay) 30
13.5.1.6 reader.clearTimeout(timeoutId) 30
13.5.1.7 reader.tick1Hz() – default implementation 30
13.5.1.8 reader.now() (CR5200) 30
13.5.1.9 checkSAV(decode) (CR5200) 30
13.5.2 This implementation is used to manage the internal “interval” timers.cra.js 31
13.5.2.1 reader.onEvent(event) 31
13.5.2.2 reader.onDecodeAttempt(count) 31
13.5.2.3 reader.onDecode(decode) 31
13.5.2.4 reader.onDecodes(decodes) 31
13.5.2.5 reader.onConfigure(data) 31
13.5.3 .codeRules.js 31
13.5.3.1 rules_onDecodes 31
13.5.3.2 rules_onDecodeAttempt 32
13.5.3.3 rules_onDecode 32
13.5.3.4 rules_onEvent 32
13.5.3.5 rules_onConfgure(data) 33
13.5.3.6 Optional Global variables 33
13.5.4 .setttings.js 33
13.5.4.1 jsSettings.def(param, val) 34
13.5.4.2 jsSettings.put(param, val) 34

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

13.5.4.3 jsSettings.set(param, val) 34
13.5.4.4 jsSettings.reset(param) 34
13.5.4.5 jsSettings.get(param) 34

14 Appendix 36

14.1 Sending Keystrokes (CodeXml) 36

14.2 Glossary and Acronyms 38

14.3 Encrypting a JavaScript file 39

14.4 Supported JavaScript Core 39

14.5 Symbology Identifier Values 40

14.6 onDecode(s) Return Value Matrix 43

14.7 onDecode(s) Compatibility Matrix 43

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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.

Many Code readers combine bar code reading with an easy-to-use JavaScript- based application development. This user manual is applicable for the CR1100, CR1500 and CR2700 series readers.

2 Document Purpose

This manual describes the JavaScript application programming interface (API) for the Code readers (CR1100, CR1500 and CR2700). 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 when available).
- 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 communication link
  - Select type of communication 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

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

- 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 document D027153 (CR8200 CR950 CR1500 CR1100 CR2700 CRA-A271 Configuration Control Document). Please familiarize yourself with the use of these configuration settings.

Note: Please visit Code's website at: https://www.codecorp.com/ to obtain CR8200 CR950 CR1500 CR1100 CR2700 CRA-A271 Configuration Control Document from the Documentation section of the product.

6 Related Utilities

USB Virtual Com Driver: 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.

CortexTools2: Provides host access to all Code reader functionality, including file transfer between host and reader. Valid communication modes are USB Downloader, USB Virtual Com, and RS232.

These utilities are available at: https://www.codecorp.com/

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://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 should conform to the following pattern: .codeRules<.><identifier>.js
Where:

<.> must be included if <identifier> is included, or omitted if <identifier> is omitted.
<identifier> is any valid string using alphanumeric characters and '_'.

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 may also be protected using an 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, or started via a command from a host utility or a command from a barcode. Please be aware that any JavaScript file that is run remains active (variables &

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

functions) until the next reset, or until a different script is run. Each time a script is run (executed), the reader closes, resets, and restarts the JavaScript engine.

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 selected 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;
}
```

If you implement this function on the **CR5200** and you want standalone age verification to work, you will need to call the `checkSAV` function for each, individual decode object as shown below.

```
Example:

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


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

13.1.1.2 reader.onEvent(Event Object event) (CR1100, CR1500)

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(data, len)

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. The stream data comes as individual characters and a count, and not necessarily as a complete string. Use the count to assemble the string. You, as the user, must watch for the carriage return as it will be passed as a normal character, as will all other control characters. Please note that the serial comm channel is the command channel in serial mode. If you return true from your handler you must not expect the system to handle normal serial commands, such as CFG. You may watch for and replace commands (for example, replace 8x commands with the new 82x commands) by sending the new command with a reader.configure(cmd) when you detect an old command.

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

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

Prevent execution of a command

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

Return false to instruct the reader to process the command, or return true 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.onTrigger(triggerButton, buttonAction)(CR2700)

The `reader.onTrigger` property allows the user to intercept trigger presses, take actions to handle the trigger events, and then either absorb the trigger event (return true), or allow the reader to process the event after JavaScript does its handling of the event (return false). This allows a wide variety of actions to be programmed for each time a trigger button is pressed.

```
Example:
// constants for reader.onTrigger
const mainTrigger  = 0;
const frontTrigger = 1;
const rearTrigger  = 2;
```


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

```
const triggerPress = 0;
const triggerHold  = 1;     // hold is pressed for 700 mS
const triggerRelease = 2;
reader.onTrigger = function(Trigger, Type)
{
    if( Trigger == frontTrigger )
        reader.configure("CDOPG");
    if(Trigger == rearTrigger && Type == triggerHold)
        reader.configure("CFR");
    return true;
}
```

13.1.1.8 reader.onDecodeAttempt(count)

The `reader.onDecodeAttempt` function is called anytime there is a decode attempt made, including times that there hasn't been a valid decode. It takes in an integer that represents the number of successful decodes that happened during the decode attempt.

Example:

```
reader.onDecodeAttempt = function(mode)
{
   if(count)
       comm.sendText("Successful Decode");
   else
       comm.sendText("No Decode");
}
```

13.1.1.9 reader.periodic(void)

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

Example:

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

13.1.1.10 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 base model 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
```


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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, (“FFFFFFFFFFFFF”), 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. CR1100 and CR1500 will return true, and CR2700 will return false.

Example:

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

13.1.2.12 reader.locked

Note: The `reader.locked` property is not yet implemented.

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;
```


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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).

The volume of the reader.beep is controlled by the goodReadBeep volume, so setting the good read beep volume controls the reader.beep volume.

```
Example:
// beep 6 times, 250mS on, 1S off each beep, no delay before beep
reader.beep(6, 250, 1000, 0); //
reader.configure("FBGRSVO0"); // Set the volume of the beep to 0
```


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

```
// 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

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

13.1.3.10 reader.indicateWireless

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

13.1.3.11 reader.indicateBattery

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

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();
```

13.1.3.13 reader.getLastImage

getLastImage gets the last captured image from the capture engine, and returns the address high and low words and the size of the image to javascript. It allows javascript to create custom packetizing around the image data. If an encoding is selected, the image address will be the address of the encoded image.

```
Example:
Info = reader.getLastImage();
addrH = info.imgHAddress;
addrL = info.imgLAddress;
sizeL = info.imgLSize;
sizeH = info.imgHSize;
```

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

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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:

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

```
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");
```


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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	Raw output: No formatting applied by the decoder
1	Customer formatted data: This is formatting applied from a customer-supplied .parse file. These files can be generated using the DL/ID tab of CortexTools

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

Value	Definition
2	JSON formatted data: 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 decodes[i].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. CortexTools2 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 CortexTools2 (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 Rreader 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.

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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 text to the host (which may include NULL characters) via the active communication port; the text will be sent "raw" regardless of the reader comm mode settings.

```
Example:

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

13.3.2.5 comm.sendData

The sendData method sends a specified number of bytes from the address pointer to the commHost. It is treated as raw unfiltered data, so may include non-character data.

Example:

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

```
var addressHigh = 0x5038;
var addressLow  = 0x0200;
var sizeH = 0x0002;
var sizeL = 0x0192;
comm.sendData(addressHigh, addressLow, sizeH, sizeL);
```

13.3.2.6 comm.sendMessage

Sends a message to the host using protocol_sendMessage.

Example:

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

13.3.2.7 comm.getSendStatus

The `comm.getSendStatus` property is not yet implemented.

13.3.2.8 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.

Example:

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

13.3.2.9 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.

Example:

```
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);
```


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

13.3.2.10 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_ms(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.

```
Example:

{
    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.

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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

13.5.1.8 reader.now() (CR5200)

If supported by the hardware, returns the current time in seconds since midnight January 1, 1970, Otherwise, returns 0.

13.5.1.9 checkSAV(decode) (CR5200)

Standalone age verification for the CR5200 is performed in the `reader.onDecodes` function. If this function is implemented (overridden) by the user, `checkSAV` can be called to perform standalone age verification manually. You must pass in the individual decode object. The `checkSAV` function returns a value of 0 (zero) if the decode was not a driver's license or the hardware does not support the functionality. A value of 1 (one) is returned if the decode was a valid driver's license and the hardware supports the functionality. A value of -1 (minus one) is returned if there was an error.

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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

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.

13.5.2.5 reader.onConfigure(data)

This function checks for a function variable “rules_onConfigure” and if it is not “null”, it will call that function. If “rules_onConfigure” returns true, this function returns immediately. If “rules_onConfigure” returns false, this function will attempt to process the configuration data and return true if processed, otherwise false.

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 Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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.onDecodeAttempt()` is made. `rules_onDecodeAttempt()` is passed the length of the decodes array (Section 13.1.1).

13.5.3.3 rules_onDecode

The `rules_onDecode()` function is run when a call to the default `reader.onDecodes()` is made. `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.

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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 CR8200 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 rules_onConfgure(data)

When a configuration code is read, `rules_onConfigure()` is called with the contents of the configuration code as *data*. If the configuration data is handled in `rules_onConfigure()`, the value "true" is returned which will stop further action by `reader.onConfigure()`.

In the example below, the `rules_onConfigure()` calls `jsSettings.parseUserParams()` to scan the configuration data for a JavaScript user parameter command and returns "true" if the configuration data was handled successfully, otherwise "false".

```
Example:

include(".settings.js");

rules_onConfigure = function(data)
{
    if( jsSettings.parseUserParams )
        return jsSettings.parseUserParams(data);

    return false;
};
```

13.5.3.6 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.

13.5.4 .setttings.js

Code provides the ability to store persistent user settings by including ".settings.js" to an application or rules file. Configuration barcodes may be created and must match the following format:

<JS Identifier> JSUP **<P | S | R | G> <PP> <Val>**, where

<JS Identifier>: Identifier indicating that this command is to be handled exclusively by JavaScript: \x1

JS: Primary Category

UP: Sub Category

<P | S| R | G>: Action: Put, Set, Reset, or Get

<PP>: Two character parameter ID

<Val>: Quoted (optional) string

```
Example 1:
\x1bJSUPSP01 ≈ JSUPSP0”1”
Example 2:
\x1bJSUPSP1”exampleString” ≈ \x1bJSUPSP1exampleString
```

13.5.4.1 jsSettings.def(param, val)

Set the parameter *param* to the default value *val*.

This must be done before a parameter is referenced for the first time (i.e. `jsSettings.def()` must be called before any call to `jsSettings.set()`/`jsSettings.put()`/`jsSettings.reset()`/`jsSettings.get()`). This is important to ensure the run-time list of settings contains the appropriate default value when the settings file is empty.

13.5.4.2 jsSettings.put(param, val)

Set the parameter *param* to the value *val* and store in RAM. This is a temporary set and will not persist through a reboot.

13.5.4.3 jsSettings.set(param, val)

Set the parameter *param* to the value *val*, update settings in RAM and store in ‘.settings.txt’ file on the reader. This is a permanent set and will persist through a reboot.

13.5.4.4 jsSettings.reset(param)

Reset *param* to the default value set in `jsSettings.def()`.

13.5.4.5 jsSettings.get(param)

Get the current value of a parameter *param*. This will get the current value stored in RAM.

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:
include(".settings.js");

function triggerDecodeNormal()
{
```


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

```
    P0_intervalDelay_sec = jsSettings.get("P0");
    if( P0_last != P0_intervalDelay_sec )
    {
        P0_last = P0_intervalDelay_sec;
        reader.clearInterval(intervalId);
        intervalId = reader.setInterval(triggerDecodeNormal,
          P0_intervalDelay_sec);
    }
};

rules_onConfigure = function(data)
{
    if( jsSettings.parseUserParams )
        return jsSettings.parseUserParams(data);

    return false;
};

function initSettings()
{
    jsSettings.def("P0", 3);
}

// Initialize any user parameters that are used for this application
//  Note: This must be called before the first call to jsSettings.get()
initSettings();

// Get the current value of applicable user parameters
var P0_intervalDelay_sec = jsSettings.get("P0");

// Keep a copy of the last parameter value for detecting a parameter
change
var P0_last = P0_intervalDelay_sec;

// Trigger a decode on the interval defined by user parameter P0
var intervalId = reader.setInterval(triggerDecodeNormal,
P0_intervalDelay_sec);
```


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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
/g	Toggle AltGr (Alt Grave; right Alt)
/c	Toggle Ctrl
/m	Toggle Menu
/s	Toggle Shift
/w	Toggle Windows Logo
/u	Up arrow
/l	Left arrow
/r	Right arrow
/d	Down arrow
/t	Tab
/z	Delete
/e	Esc
/n	Enter
/v	End
/b	Backspace
/i	Insert
/p	Page up
/x	Page down
/h	Home
/,	500 ms delay

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

Characters	Key
/0 - /9	Number pad
/f1 - /f12	Function keys
//	/
/k	Keyboard scan codes

In addition to sending keyboard keystrokes using the keystroke representations above, CodeXML also has the ability to send USB scan codes (/k) to identify an exact key on a keyboard for a non-ASCII character.

One such use case involves some language keyboards (e.g., Italian) labeling the left Alt key as "Alt" and the right "Alt" key as "AltGr" and entering different language characters for a keystroke based on just a key, Shift+key, AltGr+key, and even AltGr+Shift+key. Using CodeXML to identify the scan code for AltGr (right Alt), a reader can send a language character available only when AltGr (Alt Grave) is pressed by sending the scan codes for AltGr and the key.

USB scan codes provide for "modifiers"; that is, an indication of whether or not the Ctrl, Shift, Alt, AltGr and/or Meta/GUI (e.g., "Windows") keys are pressed at the same time a normal key is pressed, thus "modifying" the key's keystroke. For example, to send just the "a" character using scan codes requires sending the scan code for the "a" key (0x04) with no modifier (0x00); however, to send the "A" character requires sending the "a" key's scan code with a "Shift" modifier (0x02 (left Shift) or 0x20 (right Shift)).

The table below identifies the 2-digit hexadecimal representation for the "modifier" keys.

Key	Modifier
Left Ctrl	0x01
Left Shift	0x02
Left Alt	0x04
Left Meta/GUI	0x08
Right Ctrl	0x10
Right Shift	0x20
Right Alt (AltGr)	0x40
Right Meta/GUI	0x80

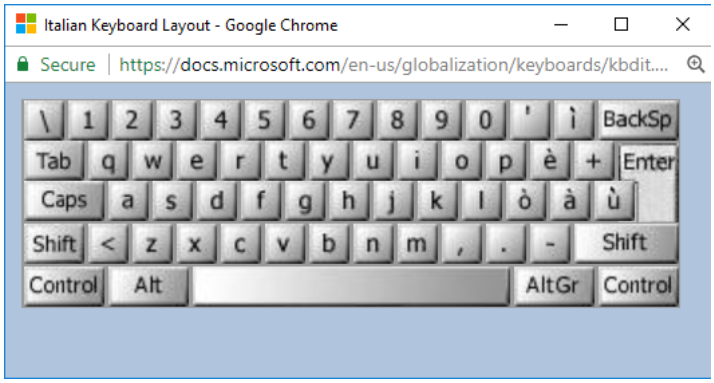
Note modifier keys may be combined by or'ing their values together; e.g., Left Shift + Right Alt = 0x42.

The CodeXML syntax for sending scan codes is the CodeXML header, followed by "/k", followed by two 2-digit hexadecimal values indicating the modifier(s) and key scan codes, respectively.

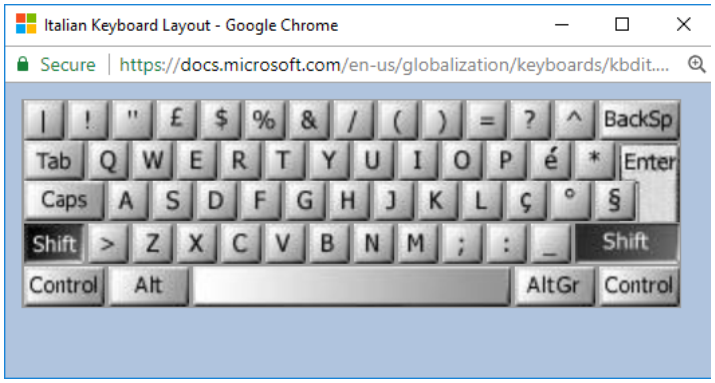
To illustrate, assume an Italian-based application requires the Euro symbol "€", which is a non-ASCII character, for use with an Italian keyboard.

Here are the Italian keyboard character layouts based on the modifier keys pressed. Note the Euro sign is available as AltGr+5 or AltGr+e.

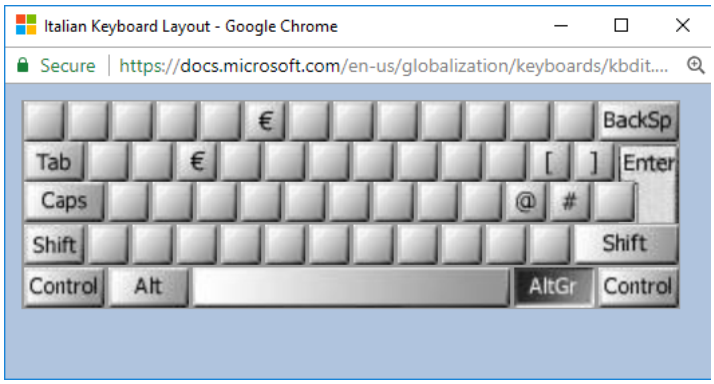
No Modifier Keys



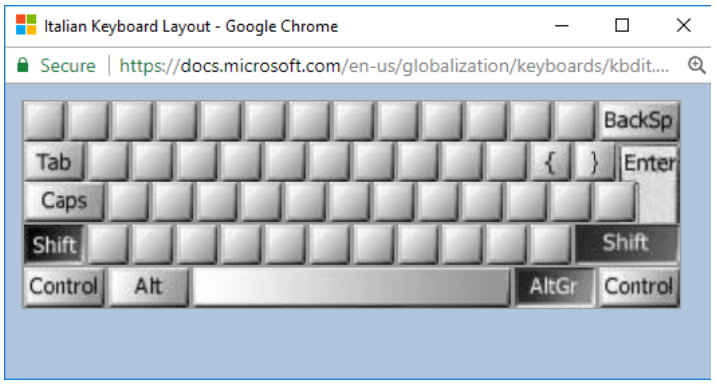
Shift Modifier Key



AltGr Modifier Key



AltGr+Shift Modifier Keys



The USB scan codes for the "5" and "e" keys, which are in the same keyboard key position on both the English and Italian keyboards, are 0x22 and 0x08, respectively. The USB modifier scan code for the Italian AltGr key position, which is also the right Alt key position on the English keyboard, is 0x40.

Below is the CodeXML for the Alt+e scan codes to enter the Euro sign on an Italian keyboard.

CodeXML:

```
\x01Y\x1Ean/2F/2Fk4008\x04
```

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 Encrypting a JavaScript file

JavaScript files can be protected through encryption. Contact support for an encryption key.
http://www.codecorp.com/code-support/

14.4 Supported JavaScript Core

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

Objects, Methods, and Properties

```
Array
Boolean
Date
Function
Math
Number
Object
Packages
RegExp
String
sun
```

Top-Level Properties and Functions

```
decodeURI
decodeURIComponent
encodeURI
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
```


The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

14.5 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

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

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

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280

14.6 onDecode(s) Return Value Matrix

The following matrix describes how the return values from the various onDecode(s) functions effect further processing of decode data by the reader.

Return Value	reader.onDecodes	rules_onDecodes	reader.onDecode	rules_onDecode
decode(s) object	Stops processing data	Continues processing data	Stops processing data	Stops processing data, but will send data to the host and beep appropriately
0	Continues processing data	Stops processing data	Stops processing data	Stops processing data

Note: do not use “true” or “false” as return values for these functions.

14.7 onDecode(s) Compatibility Matrix

The following matrix describes the compatibility when calling two instances of an onDecode(s) function.

	reader.onDecodes	rules_onDecodes	reader.onDecode	rules_onDecode
reader.onDecodes	Incompatible	Incompatible	Incompatible	Incompatible
rules_onDecodes	Incompatible	Incompatible	Compatible if rules_onDecodes returns decodes object	Compatible
reader.onDecode	Incompatible	Compatible if rules_onDecodes returns decodes object	Incompatible	Incompatible if rules_onDecode returns 0
rules_onDecode	Incompatible	Compatible	Incompatible if rules_onDecode returns 0	Incompatible

[i] Applies to battery powered readers only.

The Code Corporation • 434 West Ascension Way Suite 300, Salt Lake City, UT 84123 • +1 (801) 495-2200 • FAX +1 (801) 495-0280