# ABR Scripting Manual

5/17/19
1.0

# Table of Contents

# 1. Script Formatter Introduction

Script formatting in Barcode Manager allows for almost unlimited flexibility in the configuration of ABR barcode reader outputs and messages. It replaces the standard Output Message tools; after you enable the Script Formatter, all output messages must be generated by the script. The scripts are written in JavaScript language, conforming to ECMAScript 5.0/5.1 Language Specification. This manual will not cover all the commands possible in JavaScript, so many more things will be possible beyond what is listed here. Scripts are allowed to read the following data:

- Decoding data
- Code Quality Metrics (grading)
- Device statistics
- Device Diagnostics
- Data sent to the script by Host Mode Programming commands

The Script Formatter can be enabled from the Data Formatting > General Settings > Enable Script Formatter checkbox.

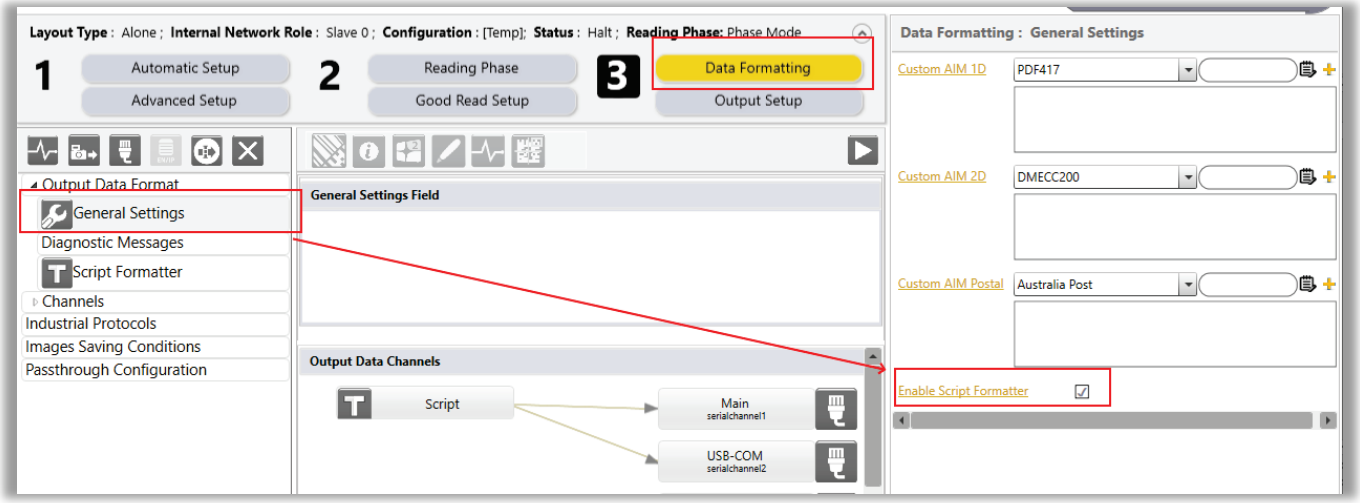


Figure 1: Enable Script Formatter

Access the Script Formatter by clicking on Script Formatter. Any of the scripts in this document should be able to be copied and pasted into the script formatter directly from Microsoft Word. Trying to use other programs may not copy correctly.

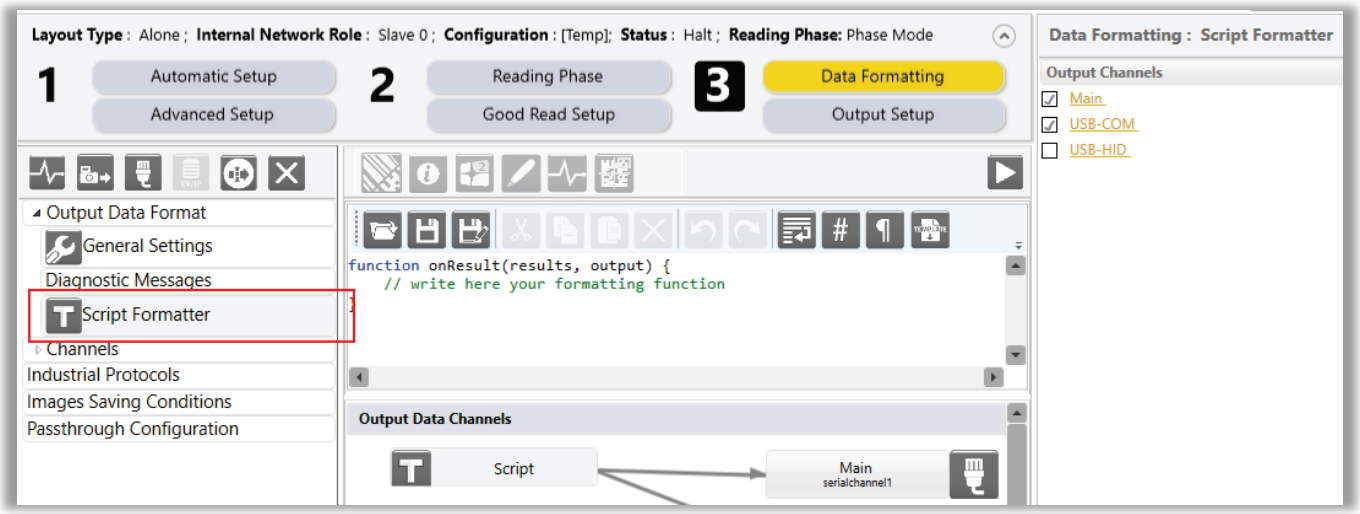


Figure 2: The Script Formatter window

# 2. Scripting Basics

## 2.1 Simple Script Examples

The default script looks like this:

```
function onResult(result, output) {
    // write your formatting function here
}
```

The code between the {} brackets of the onResult function runs every time an ABR is ready to send data. The results object is all the input data from the reading attempt. The output object is the data sent back to the output channels when the function is finished executing. Use "//" slashes to denote a comment (no code is executed).

This next script shows how to send out data from the script:

```
function onResult(result, output) {
    // write your formatting function here
    output.setMessage('Hello World!\n');
}
```

In this case the data "Hello World!" followed by the ASCII control character Line Feed (LF). ASCII characters can be written as their Hexadecimal value or their escape sequence. This means that replacing the "/n" with a "/x0A" would make no difference because 0A is the Hex value for Line Feed, and /n is the escape sequence programming shorthand for Line Feed. Either way, the data would appear in the console as:


## 2.2 Creating Variables

This is an example of how a variable is created. In this example, a message is sent only once. The counter variable increases past 0 and nothing more is sent until the ABR reboots or the script changes. This is because every time the onResults() function runs, the output object is emptied.

```
var counter = 0;
```

```
function onResult(result, output) {
    // this works only once
    if (counter == 0)
    {
        output.setMessage('Hello World!\n');
    }

    counter = counter + 1;
}
```

## 2.3 Sending Separate Data to Different Channels

An option with the setMessage command specifies the message gets sent not to all channels, but just one specific channel, using the channel names listed in Barcode Manager on the Data Formatting page.

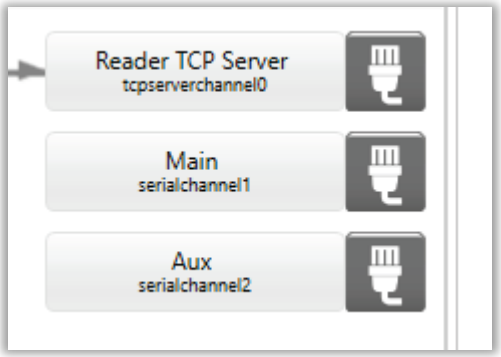


Figure 3: Channel Names

This example shows how to send a default message to all active ports but then change the message to the TCP Server port.

```
var EndLine = '\n';

function onResult(result, output) {
    // this is the default message
    output.setMessage('Hello everybody!' + EndLine);
    // this is a specific channel message
    output.setMessage('Hello TcpServer!' + EndLine,
                                'tcpserverchannel0');
}
```

## 2.4 Controlling Discrete Outputs

There are three available script events that can be triggered from a script and then connected on the Output Setup page to enable discrete outputs.

```
var counter = 0;

function onResult(result, output) {
    // this works only once
    if (counter == 0)
    {
            output.setEvent1();
    }

    counter = counter + 1;
```

```
}
```

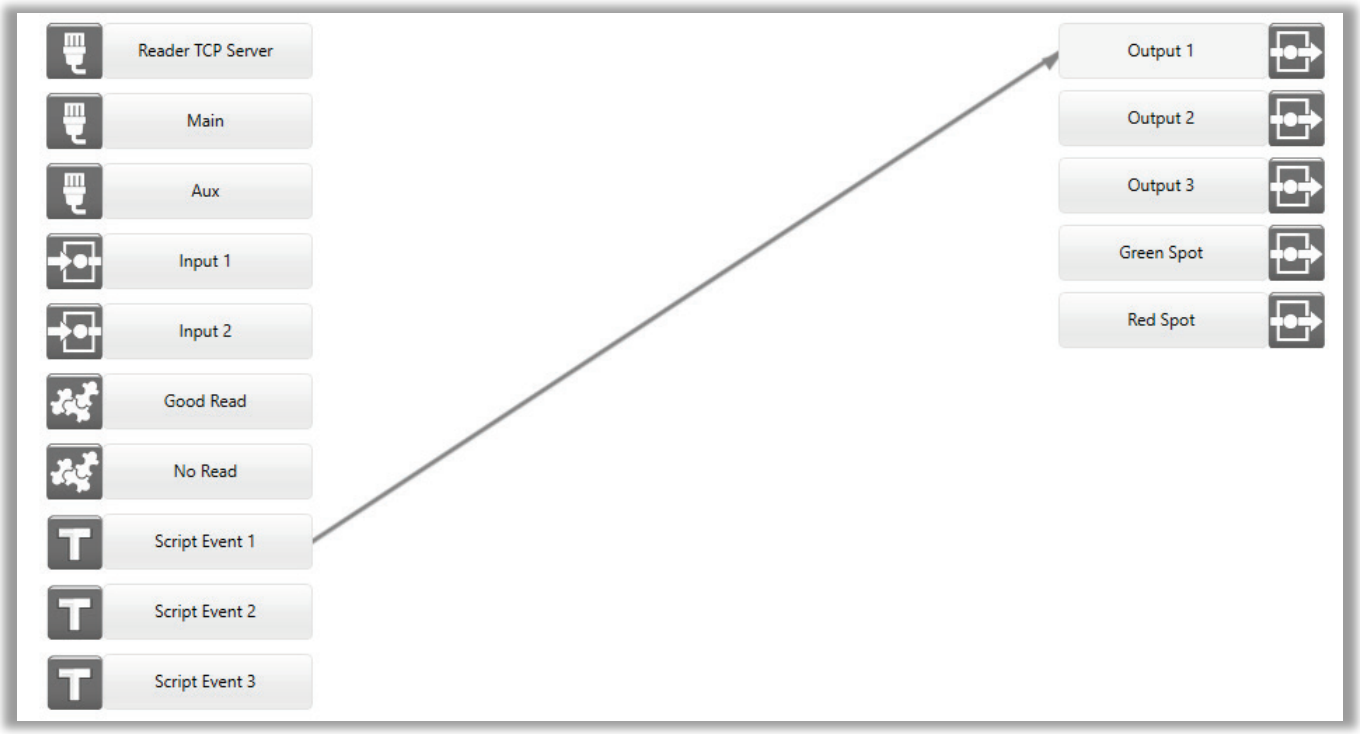


Figure 4: Tying Script Events to Physical Discrete Outputs

This example script fires Script Event 1 (and by connection, Output 1 if configured as in Figure 4) on the first Reading Phase after power up.

## 2.5 Using Array Variables

Arrays are variables that contain multiple values or elements and have an index value in brackets [] to specify the address of the element to be used.

```
var myfriends = ['Alice', "Bob", 'Carl'];

function onResult(result, output) {
    var msg;    // a local variable
        // writes the number of friends
    msg = myfriends.length + ' friends: ';
        // adds their names
    msg = msg + myfriends[0];
    msg = msg + ', ' + myfriends[1];
    msg += ', ' + myfriends[2];
    output.setMessage(msg + '\n');
}
```

This script results in the full array contents and its length being sent out, appearing in the Barcode Manager Console like this:

```
05/23/2019 08:32:42:618 AM > 3 friends: Alice, Bob, Carl<LF>
05/23/2019 08:32:43:085 AM > 3 friends: Alice, Bob, Carl<LF>
05/23/2019 08:32:43:570 AM > 3 friends: Alice, Bob, Carl<LF>
```

## 2.6 Useful String Methods

When dealing with text strings and you need to search or edit the string, these methods may be useful.

- indexOf(*searchValue[*, *fromIndex]*)

  returns the first position of specified substring

- lastIndexOf(*searchValue[*, *fromIndex]*)

  returns the last position of specified substring

  *(Ecmascript 6 but they work)*

- startsWith(*searchString[*, *position]*)

  returns '*true*' if starts with the specified substring

- endsWith(*searchString[*, *length]*)

  returns '*true*' if ends with the specified substring

- includes(*searchString[*, *position]*)

  returns '*true*' if contains the specified substring

- repeat(*count*)

  returns a string consisting of *count* string repeats

- substr(*start[*, *length]*)

  returns the specified subset of the string, by specifying the *start* and a *length*.

- substring(*start[*, *end]*)

  returns the specified subset of the string, by specifying the *start* and *end* indexes.

- slice(*start[*, *end]*)

  like subtring, extracts a section of a string and returns a new string.

- toLowerCase()

  returns the string in all lowercase

- toUpperCase()

  returns the string in all uppercase

- trim()

  trims whitespace from the beginning and end of the string.

## 2.7 Using Regular Expressions for Complex String Matching

Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects. Regular expressions can be written as, for example:

- /ab+c/i a constant *regular expression* that matches any

  string beginning with 'a' or 'A', followed by at least a 'b' or 'B' and a 'c' or 'C'; or

- RegExp('ab+c', 'i')

same as above but, in this case, the arguments can also be variables. The first part specifies the pattern to be matched. The second part can specify zero, one or more flags, for example:

- i ignores case in match
- g matches the pattern globally in the string

Here is a list of regular expression special characters:

- . (the dot) matches any single character except line terminators
- \d matches any digit (arabic numeral). Equiv. to [0-9].
- \w matches any alphanumeric character from the basic Latin alphabet. Equiv. [A-Za-z0-9_].
- \s matches a single white space character, including space, tab, form feed, line feed
- [xyz]
  [a-c] matches any one of the enclosed characters. The '-' hyphen specifies a range of characters. A '^' in the first character specifies the complementary set.
- *x*|*y* matches either *x* or *y patterns*.
- ^ matches the beginning of the string
- $ matches the end of the string
- (*x*) matches *x* pattern and remembers the match.
- *n* references back the *n*-th pattern remembered
- (?:*x*) matches *x* pattern and does not remembers the match.
- *x** matches the preceding item *x* 0 or more times.
- *x*+ matches the preceding item *x* 1 or more times.
- *x*? matches the preceding item *x* 0 or 1 times.
- *x*{*n*} matches exactly *n* occurences of the item *x*.
- *x*{*n*,} matches at least *n* occurences of the item *x*.
- *x*{*n*,*m*} matches *n* to *m* occurences of the item *x*.

Regular expressions can be very complicated. For more info, see: *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions*

These are additional string methods that use regular expressions:

- match(*regexp*)
  returns an array with the matching parts of a string
- replace(*regexp*|*substr*, *newSubstr*|*function*)

returns a new string with some or all matches of

a pattern replaced

- search(*regexp*)

  return the position of the first match between a

  regular expression and this string

- split(*[separator[, limit]]*)

  splits a String object into an array of substrings. *separator* specifies the string at which each split should occur and can be a string or a regular expression. *limit* specifies max number of splits

# 3. Data Objects

The data that can be referenced in scripts often need to be referenced as a value within a property, that is nested within another property. For example, the X coordinate of the center of the code that fulfilled Expected Code slot 1 in Code Combination mode would typically be referenced in a script as:

*result.slots[0].codes[0].center.x*

## 3.1 Properties of the **Result** object

| Property | Type | Description |
|---|---|---|
| codes | *Array* | The array of all the decoded **codes** objects |
| slots | *Array* | The array of all the configured s**lots** objects |
| images | *Array* | The array of all the decoded images **images** objects |
| success | *Boolean* | **true** if the current result is a *Good Read*, *Success* or *Match*. |
| addresses | *Array* | The array of device addresses that decoded at least one code. |
| readingCount | *Number* | The total number of decoded codes in the reading phase. |
| rejectedCodes | *Number* | The number of decoded codes removed from the output. |
| imageAcquisitionTime | *Number* | Average time used to acquire an image [ms]. |
| imageWaitTime | *Number* | Average time between acquisition and decoding [ms]. |
| imageProcessingTime | *Number* | Average time used to decode an image [ms]. |
| imageId | *Number* | The current image identifier |
| phaseId | *Number* | The current phase identifier. |
| phaseDuration | *Number* | The duration of the last phase. |
| phaseOffDuration | *Number* | The duration of the last phase off period. |

## 3.2 Properties of the **codes** object

| Property | Type | Description |
|---|---|---|
| content | *String* | The content of the code as *String*. |
| center | Point | The center position of the code inside the image. |
| corners | *Array* | The bounding box of the code as an array of **Point** objects. |
| addresses | *Array* | The array of device addresses that decoded this code. |
| binaryContent | *Uint8Array* | The content of the codes as an *Uint8Array*. |
| symbology | *String* | The code symbology name. |
| aimId | *String* | The aim identifier of the code symbology. |
| angle | *Number* | The code angle [degrees]. |
| ppe | *Number* | The code pixel per element. |
| moduleSize | *Number* | The code modules size [mils]. |
| decodingTime | *Number* | The decoding time in microseconds. |
| overrunTime | *Number* | The time in microseconds elapsed after tool's timeout. |
| symbolsSize | *String* | The code symbol size (if applicable). |
| quality | QualityMetrics | All the quality metrics info for the code as **QualityMetrics**. |
| imageIndex | *Number* | The index of image contains the code. |

## 3.3 **Parameters** Object – For External Access to Script Variables with Host Mode Programming (HMP)

The Parameters global object allows adding custom variables that can be set and requested over Ethernet or Serial ports using Host Mode Programming. To set up a variable for external access, simply define a variable like this one called matchcode, for example:

```
Parameters.matchcode = 'Alice';
```

Then a host controller can read the value of this variable with this HMP command:

SCRIPT_GET_PARAM matchcode<CR><LF>

Or write a new value to this variable with this HMP command:

SCRIPT_SET_PARAM matchcode "Bob"<CR><LF>

This technique is especially useful in complex match code applications. See ABR Host Mode Programming manual for info on HMP.

## 3.4 QualityMetrics Object Type

Provides access to the quality metrics of a single decoded code. This data is contained within the **quality** property of the **codes** object. Each metric returns an object with two properties:

- **grade** – a string representing the quality grade (A, B, C, D, or F)
- **value** – a numeric representation of the quality

For example, the reference:

*result.codes[0].quality.overall.grade*

Would return an "A" for a good barcode with an overall quality grade of A.

| Property | Type | Description |
|---|---|---|
| overall | *Object* | Overall quality grade. |
| decode | *Object* | Decode grade and value. |
| contrast | *Object* | Contrast grade and value. |
| modulation | *Object* | Modulation grade and value. |
| decodability | *Object* | Decodability grade and value. |
| meanEdgeContrast | *Object* | Mean Edge Contrast grade and value. |
| axialNonUniformity | *Object* | Axial Non Uniformity grade and value. |
| uec | *Object* | Unused Error Correction grade and value. |
| printGrowth | *Object* | Print Growth grade and value. |
| minReflectance | *Object* | Minimum Reflectance grade and value. |
| defects | *Object* | Defects grade and value. |
| fixedPatternDamage | *Object* | Fixed Pattern Damage grade and value. |
| gridNonUniformity | *Object* | Grid Non Uniformity grade and value. |

## 3.5 Point Object Type

This is an object with the X and Y pixel coordinates of a point on an image. Values are relative to the upper left corner of the image (0,0). Point objects are contained within the **center** and **corners** properties within the **code** object.

| Property | Type | Description |
|---|---|---|
| x | *Number* | The x coordinate (increases from left to right). |
| y | *Number* | The y coordinate (increases from top to bottom). |

## 3.6 **slots** Object

A slot corresponds to a single Expected Code in Code Combination. This object provides functions to access the code objects associated with the Expected Code. The slots object is within the **result** object.

| Property | Type | Description |
|---|---|---|
| codes | *array* | The array of all the code objects assigned to this slot |

## 3.7 **images** Object

Provides access to the data of a single decoded image, which are within the **result** object.

| Property | Type | Description |
|---|---|---|
| processingTime | *Number* | The decoding time in microseconds. |
| imageId | *Number* | The current image identifier |

## 3.8 **Statistics** Object

Provides access to the global device statistics. These values can be reset by a power cycle or with Barcode Manager software. Since these are global values, they are not contained within any other object, and are referenced in a script like this:

*Statistics.fps* references the current frame rate of the ABR

| Property | Type | Description |
|---|---|---|
| phases | *Number* | Number of Reading Phases. |
| goodReads | *Number* | Number of Good Reads. |
| noReads | *Number* | Number of No Reads. |
| multipleReads | *Number* | Number of Multiple Reads. |
| partialReads | *Number* | Number of Partial Reads. |
| phaseOverrun | *Number* | Number of Phase Overruns. |
| triggerOverrun | *Number* | Number of Trigger Overruns. |
| unexpectedPhaseOn | *Number* | Number of Unexpected Phase Ons. |
| images | *Number* | Number of Image Acquisitions. |
| fps | *Number* | Current frames per second. |

## 3.9 **Diagnostics** Object

Provides access to the global device diagnostics. Each diagnostic returns an object with the following properties:

**alarm** – a flag set to true if the alarm is currently active, otherwise it is set to false

**addresses** – an array of numbers where every number is the ID-NET address of the device that has activated the alarm. If the alarm equals false, the array is empty. Address 0 corresponds to the Master address. So the reference:

*Diagnostics.slaveNoReply.alarm*

Would be true if a Slave No Reply alarm is active.

| Property | Type | Description |
|---|---|---|
| slaveNoReply | *Object* | Slave No Reply diagnostic alarm. |
| slaveAddressDuplication | *Object* | Slave Address Duplication diagnostic alarm. |
| fieldbusCommunicationFailure | *Object* | Fieldbus Communication Failure diagnostic alarm. |
| fieldbusDHCPFailure | *Object* | Fieldbus DHCP Failure diagnostic alarm. |
| fieldbusConfigurationFailure | *Object* | Fieldbus Configuration Failure diagnostic alarm. |
| fieldbusTypeMismatch | *Object* | Fieldbus Type Mismatch diagnostic alarm. |
| wrongRotarySwitch | *Object* | Wrong Rotary Switch diagnostic alarm. |
| backupMemoryCommunication Failure | *Object* | Backup Memory Communication Failure diagnostic alarm. |
| protocolIndexFailure | *Object* | Protocol Index Failure diagnostic alarm. |
| noCameraHeadFailure | *Object* | No Camera Head Failure diagnostic alarm. |

# 4 Example Scripts

## 4.1 Send the first code read, or a no read string

```
function onResult(result, output) {
    var msg;
        // checks if there are any codes read ...
    if (result.codes.length > 0)
    {
        // ... yes, takes the first one ...
        var code = result.codes[0];
        // ... and gets its content
        msg = code.content;
    }
    else
        msg = "No code read";

    output.setMessage(msg + '\n');
}
```

## 4.2 Send all codes, or a no read string

```
function onResult(result, output) {
    var msg;
        // checks if there are any codes read ...
    if (result.codes.length > 0)
    {
        msg = '';    // for each code read
        for (var i = 0; i < result.codes.length; i++)
        {
            var code = result.codes[i];
            if (i != 0)        // if it isn't the first one
                msg += '-';        // add a separator
            msg += code.content;    // add the content
        }
    }
    else
        msg = "No code read";

    output.setMessage(msg + '\n');
}
```

## 4.3 Send all codes, or a no read string, using the forEach() method

The forEach() method runs its code once on each element in an array.

```
function onResult(result, output) {
    var msg;
        // checks if there are any codes read ...
    if (result.codes.length > 0)
    {
        msg = '';    // for each code read
        result.codes.forEach(function(code, index){
            if (index != 0) // if it isn't the first one
                msg += '-';     // add a separator
            msg += code.content;   // add the content
        });
    }
    else
        msg = "No code read";

    output.setMessage(msg + '\n');
}
```

## 4.4 Code Matching

```
var tobematched = 'Alice';    // wanted content

function onResult(result, output) {
    var msg;
        // checks if there are any codes read ...
    if (result.codes.length > 0)
    {
        // ... yes, takes the first one content ...
        var content = result.codes[0].content;
    // ... and checks if it matches
        if (content == tobematched)
        {
            msg = 'Yes, "' + content + "\" matches";
            output.setEvent1();
        }
        else
```

```
        {
            msg = 'No, "' + content + '" doesn\'t match';
            output.setEvent2();
        }
    }
    else
    {
        msg = "No code read";
        output.setEvent3();
    }
    output.setMessage(msg + '\n');

}
```

## 4.5 Check a code to see if it contains numbers

```
// for each read code it says if it contains numbers
function onResult(result, output) {
    var msg;
        // checks if there are any codes read ...
    if (result.codes.length > 0)
    {
        msg = '';    // for each code read
        result.codes.forEach(function(code, index){
            if (index != 0)    // if it isn't the first one
                msg += '\r\n';        // add a separator
            var tocheck = code.content;
            msg += tocheck + ": ";    // add the content
            if (tocheck.match(/^[0-9]+$/))
                msg += "contains only numbers";
            else if (tocheck.match(/^[0-9]+/))
                msg += "starts with numbers";
            else if (tocheck.match(/[0-9]+$/))
                msg += "ends with numbers";
            else if (tocheck.match(/[0-9]+/))
                msg += "contains numbers";
            else
                msg += "does not contain any number";
        });
    }
    else
        msg = "No code read";

    output.setMessage(msg + '\n');
}
```

## 4.6 String Split + FTP file naming

```
// In this example the script expects that the code
// is a record of string fields separated by '|'.

function onResult(result, output)
{
    if (result.codes.length == 0)
    {
        output.setMessage("no read\n", "tcpserverchannel0");
        return;
    }
    var code = result.codes[0].content;
    var fields = code.split("|");
    if (fields.length < 3)
    {
```

```
        output.setMessage("wrong format\n", "tcpserverchannel0");
        return;
    }
    // set the relative dest path in FTP user home
    var fullpath = "data/ftp/in/parcel_" + fields[2] + ".txt";
    output.setMessage(fullpath, "tcpserverchannel0");
    output.setFtpFileName(fullpath);
    output.setMessage(code + "\n");
}
```

## 4.7 Character Finding and Deleting, using regexp

```
// Character finding and deleting example, using regexp
function onResult(result, output) {
    var msg;
        // regular expression to match (, ) or <GS>
    var regexp = /[()\x1d]/g;
        // checks if there are any codes read ...
    if (result.codes.length > 0)
    {
        // ... yes, takes the first one ...
        var code = result.codes[0];
        // ... and gets its content
        msg = code.content;
        // delete the chars matching the regexp
        msg = msg.replace(regexp, '');
    }
    else
        msg = "No code read";

    output.setMessage(msg + '\n');
}
```

## 4.8 Replace GS ASCII Character with text, using regexp

```
// JavaScript source code to replace GS ASCII Character with text, using
regexp
var charToReplace = "\x1d"; // ascii char <GS>
var replaceWith = "<GS>";

// istantiate and initialize a RegExp object to be used for search &
replace
var regexp = new RegExp(charToReplace, "g");  // 'g' stands for global match
(see ECMAScript reference)

function onResult(result, output) {
    if (!result.success) {
        output.setMessage("\x18");  // ascii char <CAN>
        return;
    }
    var content = result.codes[0].content;
    content = content.replace(regexp, replaceWith);
    output.setMessage(content + "\n");
}
```

## 4.9 Max Accuracy Angle Measurement

```
// JavaScript source code to generate the most accurate angle possible for 1
barcode, by averaging out the angle of the 4 sides created
//by using the X,Y coordinates of the 4 corners. This was slightly more
accurate than the built-in angle value accessible from the scripts.
//The corner coordinates are always generated in integer pixel coordinates.
//The accuracy improves with higher PPE.
//If your symbology has an "Advanced Box Improvement" setting, enable it for
better accuracy.
//This script transmits the barcode contents, X, Y and angle with the
Separator character in between.
var Separator = ";";
var Header = "\x02";  //<STX>
var Terminator = "\x0D\x0A";  //<CR><LF>
var CustomMessage = "\x18";  //<CAN>

function onResult(result, output) {

    var outputMessage = Header;

    if (result.codes.length)
        {

        angle1 = Math.atan2(result.codes[0].corners[3].y -
result.codes[0].corners[2].y,result.codes[0].corners[3].x -
result.codes[0].corners[2].x) * 180 / Math.PI + 180;
        angle2 = Math.atan2(result.codes[0].corners[0].y -
result.codes[0].corners[1].y,result.codes[0].corners[0].x -
result.codes[0].corners[1].x) * 180 / Math.PI + 180;
        angle3 = Math.atan2(result.codes[0].corners[2].y -
result.codes[0].corners[1].y,result.codes[0].corners[2].x -
result.codes[0].corners[1].x) * 180 / Math.PI + 270;
        angle4 = Math.atan2(result.codes[0].corners[3].y -
result.codes[0].corners[0].y,result.codes[0].corners[3].x -
result.codes[0].corners[0].x) * 180 / Math.PI + 270;
        if (angle3 > 360)
            angle3 = angle3 - 360;

        if (angle4 > 360)
            angle4 = angle4 - 360;

        foursideaverageangle = (angle1 + angle2 + angle3 + angle4) / 4;

        outputMessage += result.codes[0].content + Separator +
result.codes[0].center.x + Separator + result.codes[0].center.y + Separator
+ foursideaverageangle;

        }
    else
        {
        outputMessage += CustomMessage;
        }
    outputMessage += Terminator;

    output.setMessage(outputMessage);
}
```

## 4.10 Angle Check Added on to Match Code Mode

```
//Script to give pass output only when a match read has an angle that meets
set tolerances.
```

```
//There are also 2 predefined barcodes that will set the angle setpoint to 2
predefined setpoint angles.

//Constants for the installer to adjust if necessary
var anglesetpoint = 0;    //Starting angle setpoint on power up
var angletolerance = 20;    //The angle 0-360 tolerance value, must be this
close to the setpoint to pass
var horizontalteachcode = "111111111117"; //This is the barcode content that
sets the angle to horizontal
var verticalteachcode = "222222222224";      //This is the barcode content
that sets the angle to vertical
var horizontalangle = 0; // defines on 0-360 scale what angle will be called
horizontal
var verticalangle = 90;    //defines on a 0-360 scale what angle will be
called vertical

//variables set on each read below
var angleadjusted = 0;
var anglesetpointadjusted = 0;
var angledistancefromsetpoint = 0;

//Code that runs after each read attempt
function onResult(result, output) {

    //On a barcode Match:
    if (result.success)
        {

                //adjust the angle numbers to wrap around so 360 is next to
0

                if (anglesetpoint > 180)
                    anglesetpointadjusted = -1 * (360 - anglesetpoint);
                else
                    anglesetpointadjusted = anglesetpoint;


                if (result.codes[0].angle > 180)
                    angleadjusted = -1 * (360 - result.codes[0].angle);
                else
                    angleadjusted = result.codes[0].angle;


                //Calculate our error distance from the setpoint
                angledistancefromsetpoint = Math.abs(anglesetpointadjusted -
angleadjusted);


                if (angledistancefromsetpoint < angletolerance)
//when the code passes the angle check
                    {

                                output.setMessage("success " +
result.codes[0].angle + '\n');
                                output.setEvent1();

                    }
                    else    // when the code fails due to angle
                        {
                         output.setMessage("fail: angle error " +
angledistancefromsetpoint + '\n');
                         output.setEvent2();
                    }
```

```
        }

    //On a barcode No Match or No Read:
    else
        {

            //On a barcode MisMatch:
              if (result.codes.length)
                  {
                //Check to see if the code is a special angle teach code
                if (result.codes[0].content == verticalteachcode)
                        {
                                    anglesetpoint = verticalangle;
                                    output.setMessage("Vertical" +
anglesetpoint + '\n');
                                    output.setEvent3();
                        }

                //Check to see if the code is a special angle teach code
                else if (result.codes[0].content == horizontalteachcode)
                        {
                                    anglesetpoint = horizontalangle;
                                    output.setMessage("Horizontal" +
anglesetpoint + '\n');
                                    output.setEvent3();
                        }

                else
                        {
                         output.setMessage("fail: Mismatch " +
result.codes[0].content + '\n');
                         output.setEvent2();
                        }
                }


            //On a barcode No Read:
            else
                {
                    output.setMessage("fail: " + " NoRead" + '\n');
                    output.setEvent2();

                }
        }

}
```

## 4.11 Output Toggling between images for External Light Control

```
// Output Toggling between images for External Light Control
// This script is designed to toggle an output off after the first image of
a phase, and back on after the 2nd image, and so on until a code is read.
//This was designed to work in Phase mode, continuous or periodic
acquisition mode, with a small "Delay Value on external trigger" time set
//on the Reading phase page to give time for the external light to turn on
before the first image.
//This was also designed to be used with 2 Image Settings configured, with
Image Setting 1 having internal lighting disabled, and Image Setting 2
having
//internal lighting enabled.
//The Good Read Setup page setting "Analysis" should be set to:"Within an
Image"

//Define the custom variables used
```

```
var counter = 0;            // 0 = the external light should have already
been on for this image that just processed, 1 = the external light should
have been off.
var lastphasecounter = 0;    //The last saved value copied from the ABR
internal phase counter, used here to figure out whether we just finished
processing the first image of a phase or not.

//This is the code that runs after each image is processed.
function onResult(results, output) {

        //Check to see if it is a new phase, and reset the counter if it is
        if (lastphasecounter != results.phaseId)
        {
        lastphasecounter = results.phaseId;
        counter = 0;
        }


        //if result of the last image was no read, check which image we are
on, and toggle the output on or off for the next image
        if (results.codes.length == 0)
        {

            if (counter == 0)
            {
                output.setMessage('No Read');
                counter = 1;
                output.setEvent2();            //turn OFF output 1, when
configuring your ABR: set Output 1's Deactivation to "Script Event 2"
            }
            else
            {
                counter = 0;
                   output.setMessage('No Read');
                   output.setEvent1();            //turn ON Output 1 to
illuminate the next image, when configuring your ABR: set Output 1's
Activation to Script Event 1 and Phase On
            }
        }
        else                //If result was a good read, send the data.
        {
        output.setMessage(results.codes[0].content);
        }
}
```

## 4.12 Sort an array of numbers

```
// Sort an array of numbers

const arr = [1, 2, 30, 4];

function compare(a, b){
  if (a > b) return 1;
  if (b > a) return -1;

  return 0;
}


function onResult(results, output) {


// set output string with the sorted array in it
    output.setMessage('Hello World!\n' + arr.sort(compare));
```

```
}
```

The output string will look like this: Hello World!<LF>1,2,4,30

## 4.13 Sort a 3x3 Array

```
//Sort a 3x3 Array
const bands = [
  { genre: 'Rap', band: 'Migos', albums: 2},
  { genre: 'Pop', band: 'Coldplay', albums: 4, awards: 13},
  { genre: 'Rock', band: 'Breaking Benjamins', albums: 1}
];

//'compare' is a function we'll create and define ourselves, in order to
return values the sort function will understand later
//to sort the bands array by the number of albums
function compare(a, b){

  const albumsA = a.albums;
  const albumsB = b.albums;

  if (albumsA > albumsB) return 1;
  if (albumsB > albumsA) return -1;

  return 0;
}

function onResult(results, output) {

//'sort' is a special javascript function for sorting arrays
bands.sort(compare);

output.setMessage('Hello World!\n' + bands[0].band + bands[1].band +
bands[2].band);
}
```

The output string will look like: Hello World!<LF>Breaking BenjaminsMigosColdplay

## 4.14 Sort all codes by X Position

```
// Sort all codes by X position
var Separator = ";";
var Header = "\x02";  //<STX>
var Terminator = "\x0D\x0A";  //<CR><LF>
var CustomMessage = "\x18";  //<CAN>


//'compare' is a function we'll create and define ourselves, in order to
return values the sort function will understand later
function compare(a, b){
  //Choose what element of the array it will be sorted by
  const albumsA = a.center.x;
  const albumsB = b.center.x;
  //Sorting action
  if (albumsA > albumsB) return 1;
  if (albumsB > albumsA) return -1;

  return 0;
}

function onResult(result, output) {
```

```
//Sort the code results array by the manner specified in the "compare"
function above
result.codes.sort(compare);


//Take all code contents from the array and transmit them
    var barcodes = "";

    result.codes.forEach(function (code, index) {
        barcodes += code.content;
        if (index < result.codes.length - 1)
            barcodes += Separator;
    });
    var outputMessage = Header;

    if (result.codes.length)
        outputMessage += barcodes;
    else
        outputMessage += CustomMessage;
    outputMessage += Terminator;

    output.setMessage(outputMessage);
}
```

## 4.15 Sort all codes by length, shortest first

```
// Sort all codes by length, shortest first
var Separator = ";";
var Header = "\x02";  //<STX>
var Terminator = "\x0D\x0A";  //<CR><LF>
var CustomMessage = "NoRead";  // “NoRead” is the message sent on a failed
read
var MinNumberOfCodes = 2          // This Requires a minimum number of codes
to transmit, specified here

//'compare' is a function we'll create and define ourselves, in order to
return values the sort function will understand later
function compare(a, b){
  //Choose what element of the array it will be sorted by
  const CandidateA = a.content.length;
  const CandidateB = b.content.length;
  //Sorting action
  if (CandidateA > CandidateB) return 1;
  if (CandidateB > CandidateA) return -1;

  return 0;
}



function onResult(result, output) {


//Sort the code results array by the manner specified in the "compare"
function above
result.codes.sort(compare);

//Take all code contents from the array and transmit them
    var barcodes = "";


    result.codes.forEach(function (code, index) {
```

```
        barcodes += code.content;
        if (index < result.codes.length - 1)
            barcodes += Separator;
    });
    var outputMessage = Header;

    if (result.codes.length>=MinNumberOfCodes)
        outputMessage += barcodes;
    else
        outputMessage += CustomMessage;
    outputMessage += Terminator;

    output.setMessage(outputMessage);
}
```

## 4.16 Code Quality Trending

```
//Code quality trending
var Separator = ";";
var Header = "\x02";  //<STX>
var Terminator = "\x0D\x0A";  //<CR><LF>
var CustomMessage = "\x18";  //<CAN>
var grade1Min = 'C';
var grade2Min = 'D';
var slidingWinIdx = 0;
var slidingWin = [ true, true, true ];

function checkMinGrade(metric,minGrade) {
    if (!metric)
      return false;
    return (metric.grade <= minGrade);
}

function onResult(result, output) {
    var outputMessage = Header;
    var barcodes = "";
    var gradeOk = false;
    var trendOk = true;
    var count = 0;
    result.codes.forEach(function(code, index){
        barcodes += code.content;
        if (index < result.codes.length - 1)
            barcodes += Separator;
        if (checkMinGrade(code.quality.overall, grade1Min) &&
            checkMinGrade(code.quality.axialNonUniformity, grade2Min))
            count++;
    });
    gradeOk = ((result.codes.length > 0) && (count == result.codes.length));
    slidingWin[slidingWinIdx] = gradeOk;
    slidingWinIdx = (slidingWinIdx + 1) % slidingWin.length;

    slidingWin.forEach(function(q){
        if(!q) trendOk = false;
    });
    if (trendOk)
        output.setEvent1();
    else
        output.setEvent2();

    if (result.codes.length)
        outputMessage += barcodes;
    else
        outputMessage += CustomMessage;
```

```
    outputMessage += Terminator;
    output.setMessage(outputMessage);
}function onResult(results, output) {
    // write here formatting function here
}
```