



AirScan

RiSL Programming Guide

July 2022

Copyright © 2022 Infinite Peripherals

All Rights Reserved.

Warranty

The information contained in this document is subject to change without notice. Infinite Peripherals makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties or merchantability and fitness for a particular purpose. Infinite Peripherals shall not be liable for errors contained herein, nor for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Table of Contents

Introduction.....	3
Pre-Requisites	3
About the Programming Guide	4
^StartCard	5
^Font	6
^Text	7
^TextC.....	8
^TextL	9
^TextR.....	10
^Button	11
^CardBackColor	12
^ShowCard.....	13
^SaveCard	14
^LoadCard.....	15
^CardExists.....	16
^Set	17
^Vibrate.....	18
^PlaySound	19
^Clear	20
RiSL Examples.....	21

Introduction

This guide describes how to use the **Ring Scanner Language (RiSL)**. The language is comprised of a set of commands that allows a system to customize what is displayed on the screen of a Halo Ring running our off-the-shelf AirScan application.

The commands can be used to customize the size, text and color of each message while additional functionality like buttons, vibration and sound can provide the user with advanced feedback and gather data input from the user as well.

Pre-Requisites

- Halo Ring with AirScan application installed
- MQTT Server - or -
- Bluetooth SPP server

About the Programming Guide

How the Commands are Documented

Name: Name of the command

Description: Each command will come with a paragraph describing how the command is used. This section will explain the command's capabilities and how it can be used in conjunction with other commands. Examples are sometimes also included to further elaborate on how the command can be used.

Format: The format section details how the command is called, what parameters it takes and the order of the parameters.

Parameters: If a command has input parameters, they will appear in a table like the one below. Each parameter will have a details section that describes the values that can be passed and will often come with an accompanying note.

Parameters	Details
y = y coordinate	Values: 1 to 200 This number is in pixels with 200 being the maximum height of a RiSL card.
text = text to be written	Values: Any string

RiSL Examples

After the RiSL Commands section is the RiSL Examples section. This section will include sample cards with screenshots that will demonstrate the use of every command found in this document.

If you are unclear about how to use of any of the RiSL commands, be sure to check out how they are being used in this section.

^StartCard

Starts the current card

The **^StartCard** command starts a new card using the size parameters supplied. Any commands sent after a card is “started” will apply to the current card. Only until **^ShowCard** or **^SaveCard** are called will the current card be “ended”. When a card has been “ended” any commands that are received will be ignored until a new card is “started”.

Format: ^StartCard|width|height

Parameters	Details
width = card width (in pixels)	Values: 1 to 290 This number is in pixels with 290 being the maximum width of a RiSL card.
height = card height (in pixels)	Values: 1 to 200 This number is in pixels with 200 being the maximum height of a RiSL card.

^Font

Sets the current font

The **^Font** command sets the current font that any Text (**^Text**, **^TextC**, **^TextL**, **^TextR**) or Button (**^Button**) command will adhere to. The command can take a list of attributes that will be applied to the current font. The current font can be set multiple times in a single RiSL card allowing you to have text and buttons using different fonts at once.

For example, the command **^Font|24|Bold|Underline|#FFFFFF** would set the current font to be size 24, Bold and Underlined with the white hex color (#FFFFFF). Any **^Text** command that follows would use this font when writing to the current card. In addition, any **^Button** that had text would use this font as well.

Format: **^Font|size|attribute|attribute|...**

Parameters	Details
size = font size	Values: Any integer
attribute = font attribute	Values: Attributes can be any of the following: <ul style="list-style-type: none"> • Bold • Underline • Italic • A hex color with the format: #FFFFFF Note: There can potentially be 0 – 4 attributes listed at any one time.

^Text

Writes text using the current font

The **^Text** command writes text at the given x,y coordinate. The text will be written using whatever the current font is set to. If no font was set before the **^Text** command, then the default font will be used. The default font is size 24 in white.

Format: ^Text|x|y|text

Parameters	Details
x = x coordinate (in pixels)	Values: 1 to 290 This number is in pixels with 290 being the maximum width of a RiSL card.
y = y coordinate (in pixels)	Values: 1 to 200 This number is in pixels with 200 being the maximum height of a RiSL card.
text = text to be written	Values: Any string

^TextC

Writes centered text using the current font

The **^TextC** command writes centered text at the given y pixel row. The text will be written using whatever the current font is set to. If no font was set before the **^TextC** command, then the default font will be used. The default font is size 24 in white.

Format: ^TextC|y|text

Parameters	Details
y = y coordinate	Values: 1 to 200 This number is in pixels with 200 being the maximum height of a RiSL card.
text = text to be written	Values: Any string

^TextL

Writes left justified text using the current font

The **^TextL** command writes left justified text at the given y pixel row. The text will be written using whatever the current font is set to. If no font was set before the **^TextL** command, then the default font will be used. The default font is size 24 in white.

Format: ^TextL|y|text

Parameters	Details
y = y coordinate	Values: 1 to 200 This number is in pixels with 200 being the maximum height of a RiSL card.
text = text to be written	Values: Any string

^TextR

Writes right justified text using the current font

The **^TextR** command writes centered text at the given y pixel row. The text will be written using whatever the current font is set to. If no font was set before the **^TextR** command, then the default font will be used. The default font is size 24 in white.

Format: ^TextR|y|text

Parameters	Details
y = y coordinate	Values: 1 to 200 This number is in pixels with 200 being the maximum height of a RiSL card.
text = text to be written	Values: Any string

^Button

Draws a button capable of notifying the server it is clicked

The **^Button** command creates a button at the given x,y coordinate. The command takes in parameters that define size, title and color. The button title will use the current font that is set.

The parameter **id** allows the Halo Ring to communicate back with the server that the button has been pressed. For example, look at the following button:

```
^Button|10|100|270|50|#FF0000|Package Damaged?|pkgDmgd
```

If this button was pressed, the server would receive the JSON {"buttonPressed": "pkgDmgd"}

Format: ^Button|x|y|width|height|color|text|id

Parameters	Details
x = x coordinate	Values: 1 to 290 This number is in pixels with 290 being the maximum width of a RiSL card.
y = y coordinate	Values: 1 to 200 This number is in pixels with 200 being the maximum height of a RiSL card.
width = width of button	Values: 1 to 290 This number is in pixels with 290 being the maximum width of a RiSL card.
height = height of button	Values: 1 to 200 This number is in pixels with 200 being the maximum height of a RiSL card.
color = background color of button	Values: Hex color using the format: #FFFFFF
text = title of button	Values: Any string
id = identifier returned on button press	Values: Any string

^CardBackColor

Sets the background color of the current card

The **^CardBackColor** command sets the background color of the current card using the given color. Make sure this color is in the hex format with the “#” included.

Format: ^CardBackColor|color

Parameters	Details
color = hex color	Values: Hex color using the format: #FFFFFF

^ShowCard

Shows the current card

The **^ShowCard** command will display the current card to the Halo Ring's screen. After this is done, the current card has effectively "ended". This means a new card must be "started" using the **^StartCard** or **^LoadCard** commands before you can start adding other RiSL commands. Any commands received before a new card is "started" are ignored.

Format: ^ShowCard

^SaveCard

Saves the current card to the Halo Ring's memory

The **^SaveCard** command is a convenience command that allows you to save a RiSL card into the Halo Ring's memory, so you don't need to rewrite it over and over. This command is used in conjunction with the **^LoadCard** command that loads a saved card to be the current card.

^SaveCard is similar to **^ShowCard** in that when performed, the current card has effectively "ended". This means a new card must be "started" using the **^StartCard** or **^LoadCard** commands before you can start adding other RiSL commands. Any commands received before a new card is "started" are ignored.

Here's an example use case: You have a RiSL card that is sent when a package is flagged and requires special attention. It is used many times a day, so to make easier you tell the Halo Ring to save it into memory. Now that it is saved, instead of building the card from scratch, you can simply call **^LoadCard** and it will be loaded as the current card as if all the commands that make up that card were just entered.

Cards saved in the Halo Ring's memory will last until they are replaced, or that app is deleted. To replace a previously saved card, just use the same name and call **^SaveCard** again.

Format: ^SaveCard|name

Parameters	Details
name = name of card	Values: Any string This serves as the card's unique identifier when being saved into a Halo Ring's memory.

^LoadCard

Loads a card from the Halo Ring's memory

The **^LoadCard** command is used to load a previously saved card from the Halo Ring's memory and set it as the new current card. If no card matching the name is found the Halo Ring will do nothing.

An easy way to think of this is to imagine all the commands that make up a card before you save it. When loading the card, it's as if every one of those commands were processed when you called **^LoadCard**.

Format: ^LoadCard|name

Parameters	Details
name = name of card	Values: Any string This is the card's unique identifier when it was saved into the Halo Ring's memory.

^CardExists

Checks to see if a specific card has been saved in the Halo Ring's memory

The **^CardExists** command allows the server to check if a card matching the provided name has been saved in the Halo Ring's memory.

This can be especially useful when new Halo Rings are introduced that might not have saved your premade cards yet.

Format: ^CardExists|name

Parameters	Details
name = name of card	Values: Any string This is the card's unique identifier when being stored into the Halo Ring's memory.

^Set

Sets any placeholders in the current card to the given text

The **^Set** command leverages **placeholders** which can be added to any text or button command. A **placeholder** is any string surrounded by square brackets. Example: [packageNum]

As the name suggests, a **placeholder** is meant to be replaced by a different string by using the **^Set** command. If no string is set, the text or button will simply print the **placeholder** as is. This ability when used in conjunction with the **^SaveCard** and **^LoadCard** commands means you have the ability change text in previously saved cards.

For an example of this command in action, please see our Sample RiSL Cards section at the end of this document.

Format: ^Set|[placeholder]|text

Parameters	Details
[placeholder] = name of the placeholder	Values: Any string Note: The placeholder must follow the format: [string]. Example: ^Set [packageNum] 12345
text = text to replace placeholder	Values: Any string

^Vibrate

Vibrates the Halo Ring

The **^Vibrate** command is used to vibrate the Halo Ring to give the user haptic feedback. This command is unique in that it does not need to be included with a card to work and can be used standalone. If it is included in a card, the device will vibrate when the current card is displayed with the **^ShowCard** command.

Format: ^Vibrate|type

Parameters	Details
type = type of vibration	<p>Values: 1 to 4</p> <p>There are currently 4 different vibration lengths ranging from 1 (shortest) to 4 (longest).</p>

^PlaySound

Plays a sound from the Halo Ring

The **^PlaySound** command is used to play sounds from the Halo Ring to give the user audio feedback. This command is unique in that it does not need to be included with a card to work and can be used standalone. If it is included in a card, the sound will play when the current card is displayed with the **^ShowCard** command.

Format: ^PlaySound|type

Parameters	Details
type = type of sound	Values: The current sounds supported are: <ul style="list-style-type: none"> • Good • Good2 • Bad • Alert

^Clear

Clears all cards on screen

As RiSL cards are sent to the Halo Ring, they are added to the end of a list of all cards received. This list allows the user to scroll up to see previous cards they received in case they need to view the information again. The **^Clear** command will erase all previous cards to give the user a new empty list.

Format: ^Clear

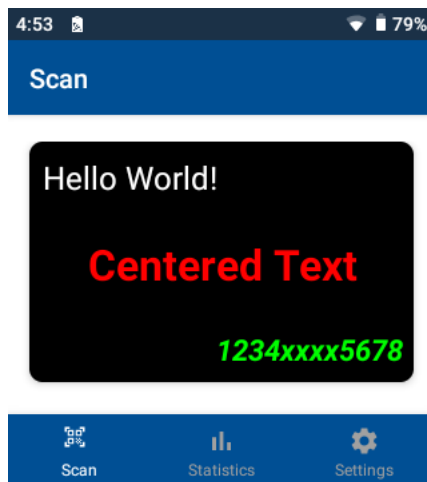
RiSL Examples

The following is a list of RiSL examples that will cover every RiSL command in this document. The examples will be accompanied by screenshots from a Halo Ring to give you a better understanding of each command and what the resulting card looks like.

Basic Font & Text

This example creates a card with 3 different text commands. The first one occurs before any font is defined so it uses the default font. After the first text command, a red and bold font is defined which is applied to the following text command. Then again, a new font and text. Notice that each of these text commands use a different alignment (^TextC, ^TextL, ^TextR).

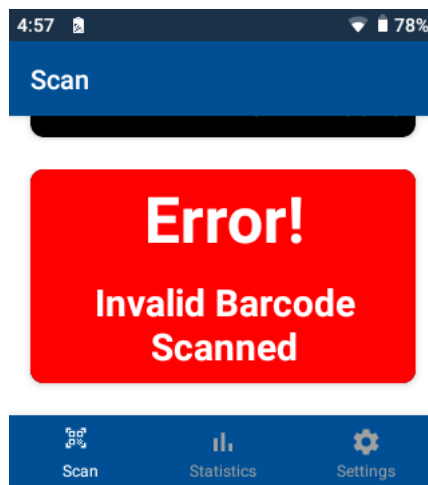
```
^StartCard|290|180^TextL|10|Hello World!^Font|32|Bold|#FF0000^TextC|70|Centered
Text^Font|22|Italic|Bold|#00FF00^TextR|140|1234xxxx5678^ShowCard
```



Color, Vibration and Sound

In this example, we are setting the background of the card to a bright red to get the user's attention while making sure our font is big and bold. Two other things we do to make sure we grab the user's attention are vibrate the device and play a sound.

```
^StartCard|290|160^CardBackColor|#FF0000^Font|48|Bold|#FFFFFF^TextC|4|Error!^Font|28|
Bold|#FFFFFF^TextC|80|Invalid Barcode Scanned^Vibrate|2^PlaySound|Bad^ShowCard
```



Adding Save & Load Card

To add to this example, let's assume invalid barcodes get scanned often and we want to save this card so in the future we can call it with a simple load command. By changing **^ShowCard** into **^SaveCard** we now save the card in memory for future use.

```
^StartCard|290|160^CardBackColor|#FF0000^Font|48|Bold|#FFFFFF^TextC|4|Error!^Font|28|
Bold|#FFFFFF^TextC|80|Invalid Barcode
Scanned^Vibrate|2^PlaySound|Bad^SaveCard|invalidBC
```

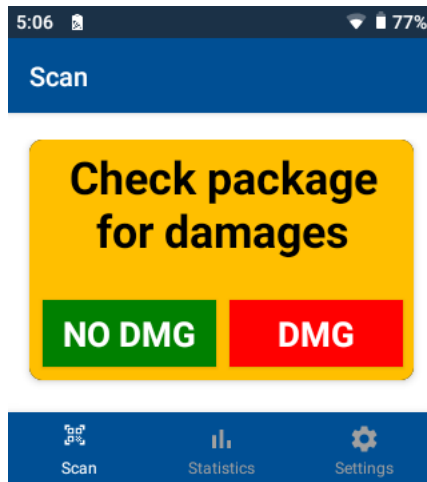
To load this saved card, we simply call load card with our identifier then show the loaded card. The output will be the exact same as the screenshot above.

```
^LoadCard|invalidBC^ShowCard
```

Buttons

This example utilizes buttons to ask the user for input. Remember that the “id” of the button that is pressed is sent back to the server so they can be notified if the package is damaged.

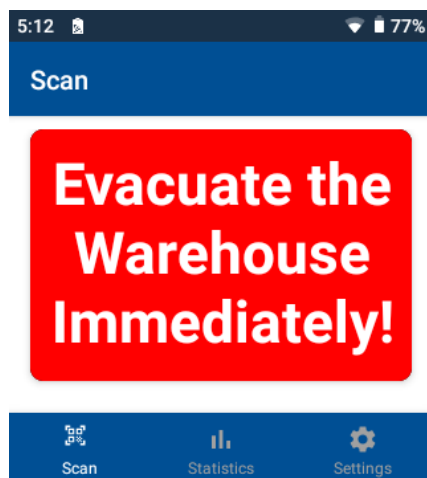
```
^StartCard|290|180^CardBackColor|#ffbf00^Font|34|Bold|#000000^TextC|5|Check package
for damages^Font|26|Bold|#FFFFFF^Button|10|120|130|50|#008000|NO
DMG|noDmg^Button|150|120|130|50|#FF0000|DMG|yesDmg^Vibrate|1^ShowCard
```



Clear

In this example, we are imagining a scenario where there is an emergency at the warehouse. We don't want the user to see anything on the screen besides the current message, so we call the clear command, then send out alert with vibrations and sound.

```
^Clear^StartCard|290|190^CardBackColor|#FF0000^Font|44|Bold|#FFFFFF^TextC|10|Evacuate
the Warehouse Immediately!^Vibrate|4^PlaySound|Alert^ShowCard
```

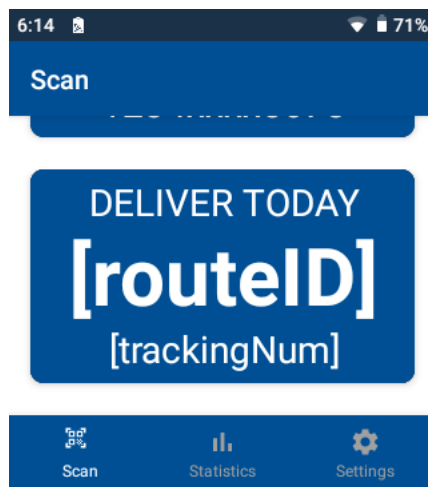


Placeholders & Set

If you look closely at this example, you will notice there are two placeholders within text commands: [routeID] and [trackingNum]. The below command is not to show but to save the card. Now if we were to send the command below it, then the place holders would show as is.

```
^StartCard|290|160^CardBackColor|#004F94^Font|28|#FFFFFF^TextC|5|DELIVER  
TODAY^Font|60|Bold|#FFFFFF^TextC|38|[routeID]^Font|28|#FFFFFF^TextC|115|1234xxxx5678  
^SaveCard|deliverToday
```

```
^LoadCard|deliverToday^ShowCard
```



Now instead of just showing the card, let's set our two placeholders to real values.

```
^LoadCard|deliverToday^Set|[routeID]|C003^Set|[trackingNum]|1234xxxx5678^ShowCard
```

