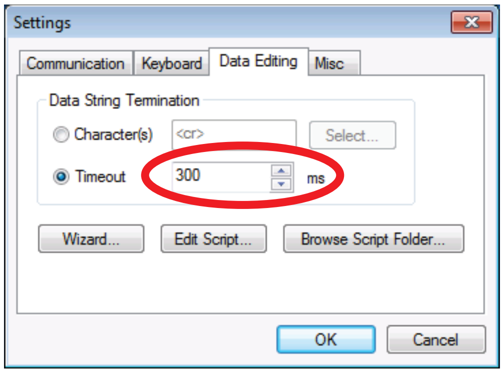
7. The millisecond delay box will activate.



Data String Termination

Character(s) <cr>

Timeout 300 ms

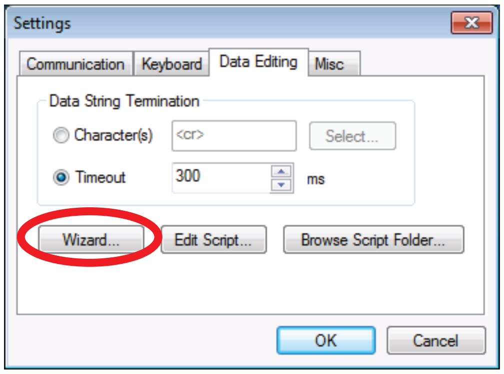
8. Set the desired millisecond delay by either using the up or down arrows, or by entering a number directly in the box.

2.9.4.3.1 Wizard

Enable the user to define a simple setup that matches and modifies a data string. The wizard consist of four different parts, Trig, Strip, Replace, and Add. It is only intended to be used for very simple tasks. For more advanced tasks, you need to use the scripting language.

To start the wizard:

1. Tap on the **Wizard** button.

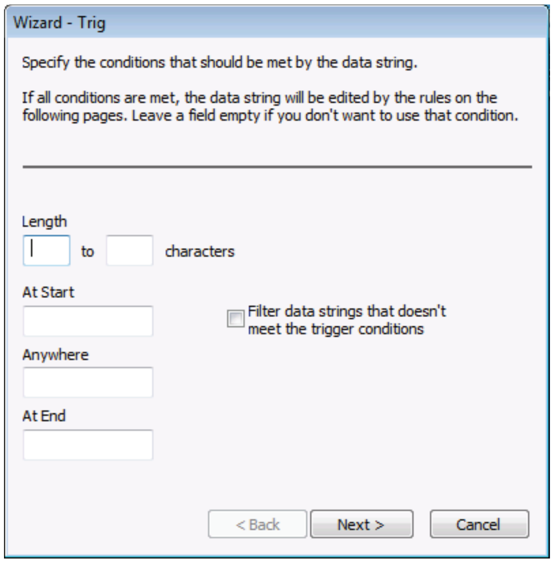


Data String Termination

Character(s) <cr>

Timeout 300 ms

2. The **Wizard – Trig** window will open.



Specify the conditions that should be met by the data string.

If all conditions are met, the data string will be edited by the rules on the following pages. Leave a field empty if you don't want to use that condition.

Length [ ] to [ ] characters

At Start

Anywhere

At End

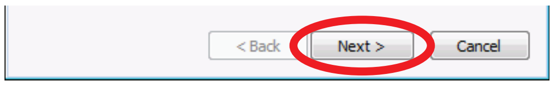
Filter data strings that doesn't meet the trigger conditions

In this dialog you can specify zero or more conditions that should be met for a data string before it is edited by the wizard rules. If a data string doesn't meet the conditions and the check box Filter Data Strings is checked, the data string is filtered. If the check box is cleared, the data string is sent to the receiving application unmodified.

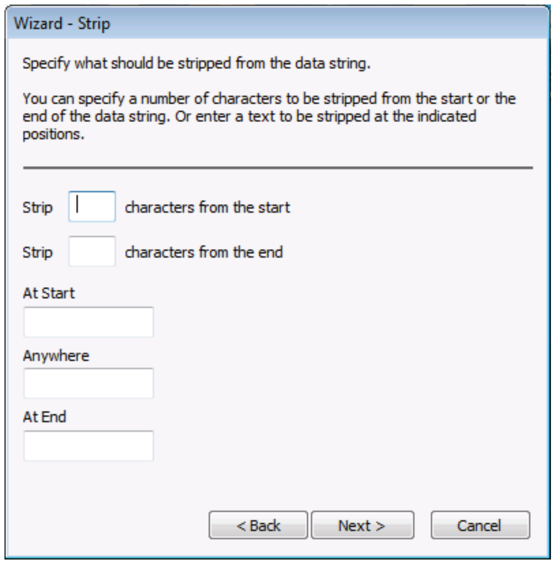
If you specify a length condition, any data strings shorter or longer will not be modified by the wizard rules.

You can enter texts that should be present in the data string. Entering 00 in the At Start field will check for 00 in the beginning of the data string. Then any string that starts with something else will not meet the condition.

3. Click the **Next** button when you are done.



4. The **Wizard – Strip** window will open.



Specify what should be stripped from the data string.

You can specify a number of characters to be stripped from the start or the end of the data string. Or enter a text to be stripped at the indicated positions.

Strip [ ] characters from the start

Strip [ ] characters from the end

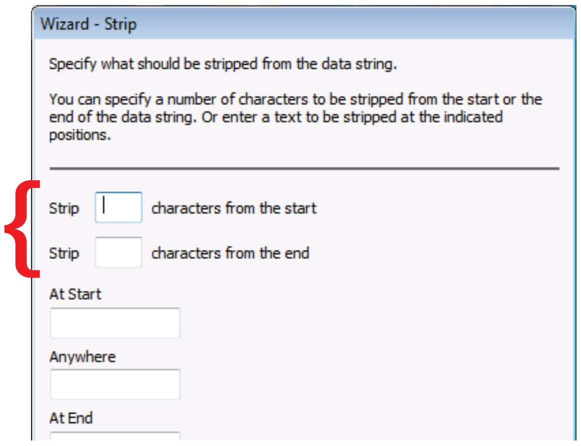
At Start

Anywhere

At End

5. Select one of two choices:

- Stripping a certain number of characters at the start and/or the end of the data string, or



Specify what should be stripped from the data string.

You can specify a number of characters to be stripped from the start or the end of the data string. Or enter a text to be stripped at the indicated positions.

Strip [ ] characters from the start

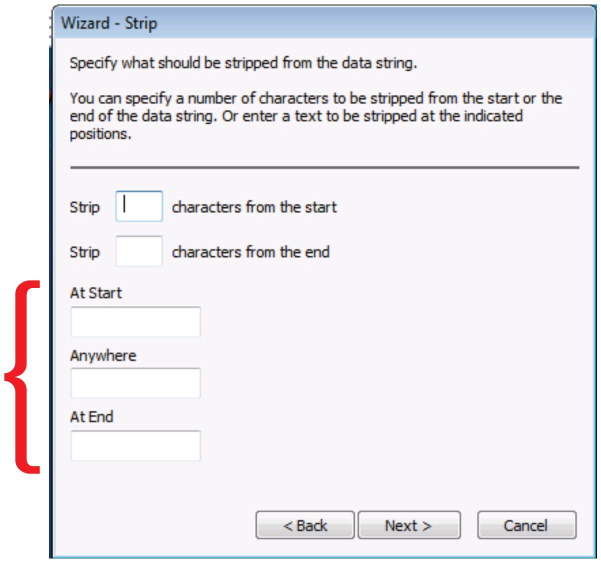
Strip [ ] characters from the end

At Start

Anywhere

At End

- Removing one or more texts at the indicated positions of the data string.


Wizard - Strip

Specify what should be stripped from the data string.

You can specify a number of characters to be stripped from the start or the end of the data string. Or enter a text to be stripped at the indicated positions.

Strip characters from the start

Strip characters from the end

At Start

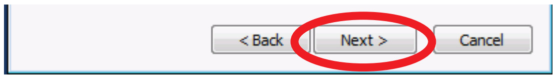
Anywhere

At End

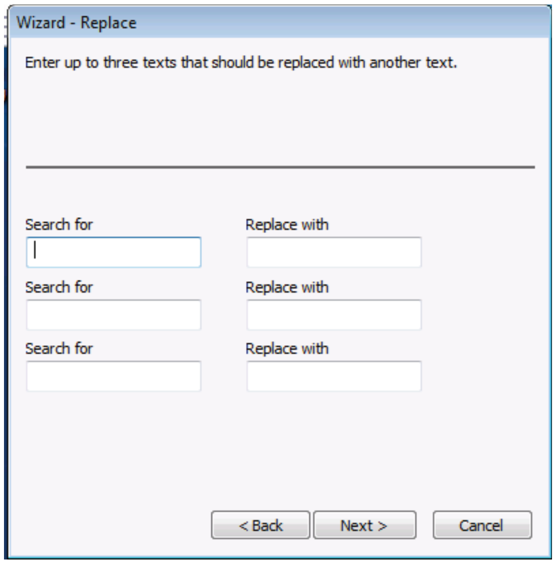
For example, if 00 is specified in the field **At Start** and the data string is 0012345, the resulting output will be 12345. However, if the data string is 9912345, nothing will be removed from the start of the data string.

If all fields are left empty, no text will be stripped from the data string.

6. Tap the **Next** button.



7. The **Wizard - Replace** window will open.

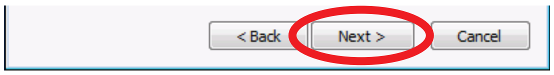

Wizard - Replace

Enter up to three texts that should be replaced with another text.

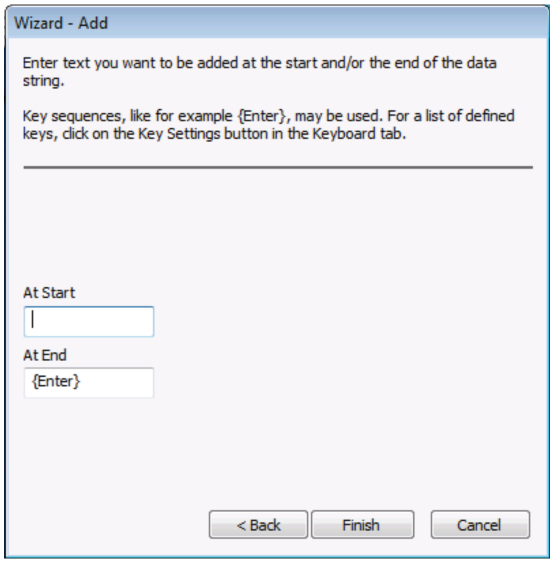
Search for	Replace with
Search for	Replace with
Search for	Replace with

NOTE: This window allows the user to enter up to three text replacements. Enter the text to be replaced in one of the **Search for** fields, and then enter the text to replace it with in the corresponding **Replace with** field.

8. Tap the **Next** button.



9. The **Wizard – Add** window will open.


Wizard - Add

Enter text you want to be added at the start and/or the end of the data string.

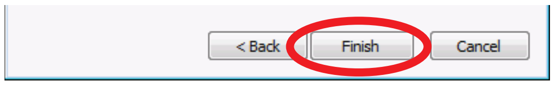
Key sequences, like for example {Enter}, may be used. For a list of defined keys, click on the Key Settings button in the Keyboard tab.

At Start

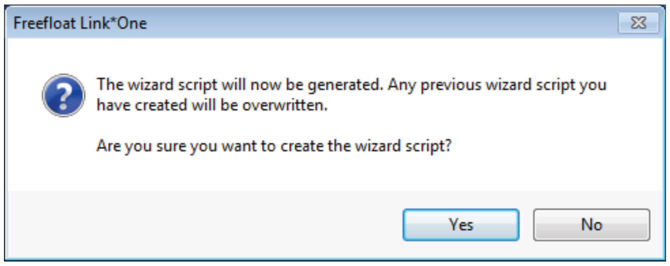
At End
{Enter}

This window allows the user to enter texts that need to be added to the data string at the start or the end.

10. When text entry is complete, click on the **Finish** button.



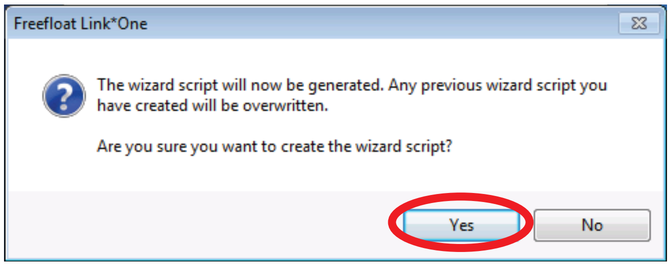
11. The **Freefloat Link*One Wizard Script** window will appear.


Freefloat Link*One

The wizard script will now be generated. Any previous wizard script you have created will be overwritten.

Are you sure you want to create the wizard script?

12. Tap the **Yes** button to confirm that a wizard script should be created.


Freefloat Link*One

The wizard script will now be generated. Any previous wizard script you have created will be overwritten.

Are you sure you want to create the wizard script?

The wizard script is a Lua script and can be modified manually afterwards if, for example, the need to add more advanced conditions or modifications arises.

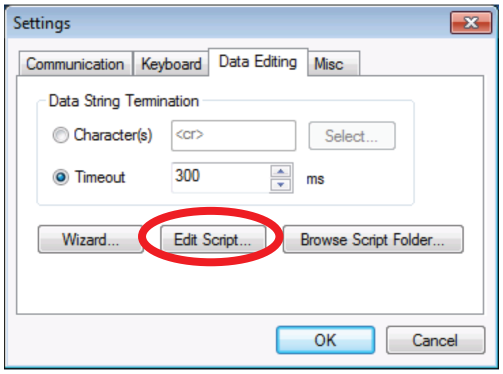
NOTE: The wizard script is overwritten each time the wizard is run.

2.9.4.3.2 Edit Script

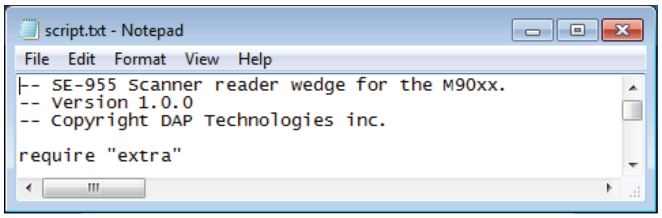
Opens the Link*One script. The name of the script file is **Script.txt** and it is opened in the associated program, normally Notepad.

To edit a script:

1. Tap the **Edit Script** button.

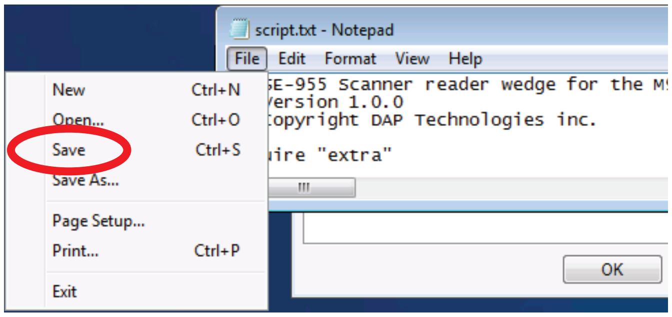


2. A **Notepad** window will open.



```
-- SE-955 Scanner reader wedge for the M90xx.
-- Version 1.0.0
-- Copyright DAP Technologies inc.

require "extra"
```

3. Edit the script as desired.
4. When finished, tap **File > Save** to save the script.



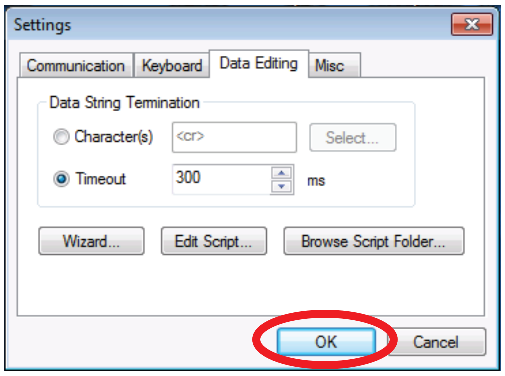
```
SE-955 Scanner reader wedge for the M
ersion 1.0.0
opyright DAP Technologies inc.

ire "extra"
```

Alternatively, the text editor **SciTE** knows the syntax of Lua. It might be useful when writing Link*One scripts.

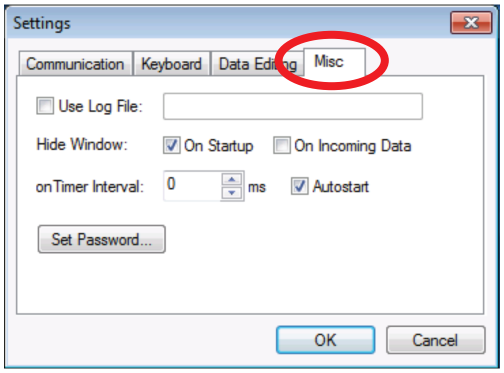
2.9.4.3.3 Browse Script Folder

Opens Windows Explorer in the folder that contains the script, configuration, and the license file. For more information about scripting in Link*One, see the topic **Link*One Scripting**.

Tap the **OK** button to save all changes.

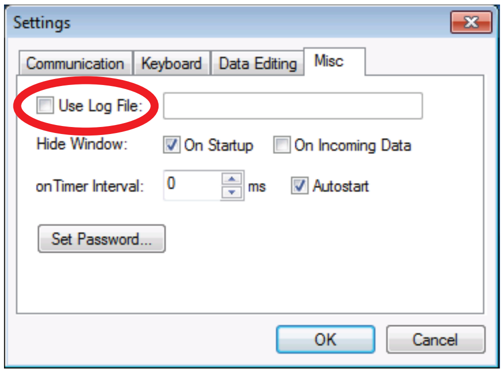


2.9.4.4 Misc Tab



2.9.4.4.1 Use Log File

If all the internal messages and events in Link*One are to be written to a log, check the setting **Use Log File**, and enter a valid path and filename in the edit box. The log file is mostly used for troubleshooting a script.



- To hide Link*One's main window on startup, check **On Startup** in the **Hide Window** area.
- To hide Link*One's main window when serial data is received, check **On Incoming Data** in the **Hide Window** area.

When a value greater than 0 is entered into **onTimer Interval**, the script method **onTimer()** will be called once during the specified time interval. For example: if you enter the value 3000, **onTimer()** will be called once every third second. Please take care when choosing a value here, if 1 ms is entered, **onTimer()** will be called 1000 times per second. This could make a PC unresponsive. Of course this will be highly dependant on what code the **onTimer()** method contains.

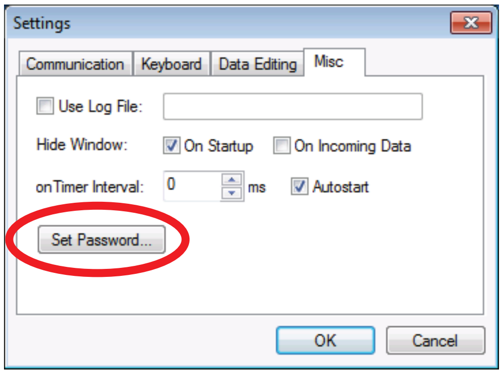
When the option **Autostart** is checked, Link*One will start automatically when Windows is launched. Link*One will then be started with the profile for which **Autostart** was activated . If there are two profiles—one serving COM1 and the other serving COM2—**Autostart** can be checked for each of those profiles. One instance of Link*One will be started at login for every profile that has **Autostart** checked.

2.9.4.4.2 Set Password

Allows the user to set a password that is required when exiting Link*One and when clicking on the Settings... button in the main window.

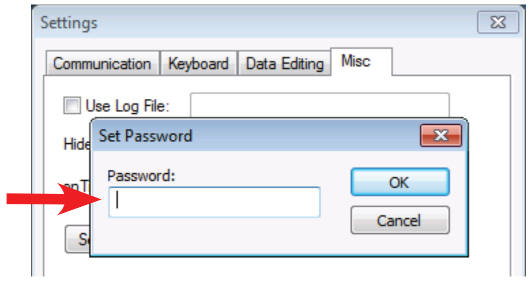
To set a password:

1. Tap the **Set Password** button.



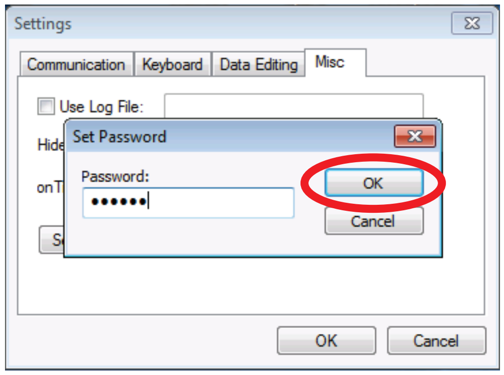
☐ Use Log File:
Hide Window: ☑ On Startup ☐ On Incoming Data
onTimer Interval: 0 ms ☑ Autostart

2. Enter a password into the **Password** box.



Password:

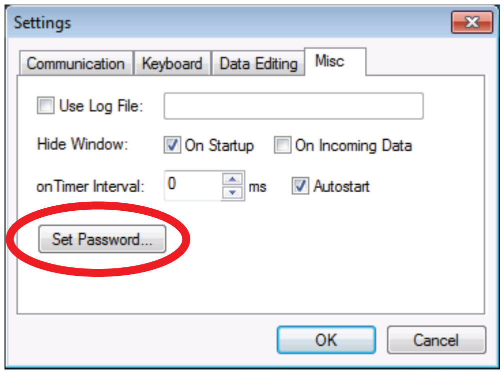
3. Tap the **OK** button to save the password.



Password: ••••••

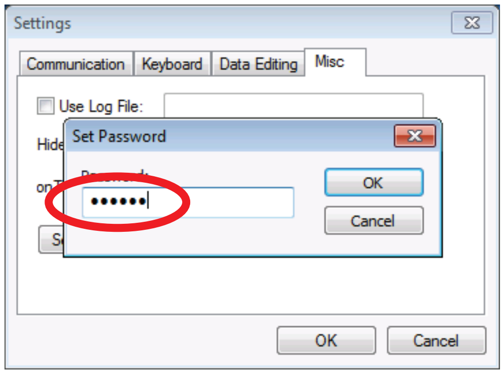
To remove a password:

1. Tap the **Set Password** button to open the **Set Password** window.



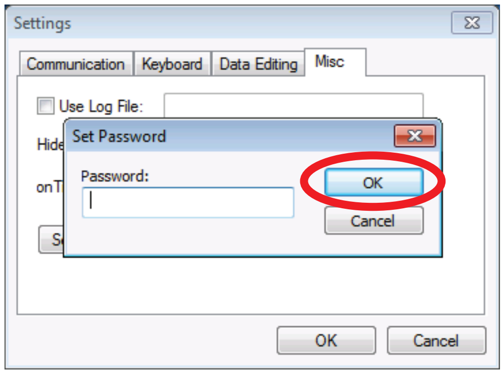
☐ Use Log File:
Hide Window: ☑ On Startup ☐ On Incoming Data
onTimer Interval: 0 ms ☑ Autostart

2. Delete the text in the **Password** box.



••••••

3. Tap the **OK** button.



Password:

2.9.4.4.3 Settings Location

A Link*One configuration consist of mainly two parts, the settings (serial port configuration, hot keys, etc.) and the script file(s).

The settings are file-based to enable different users on the same PC to share the same Link*One configuration. The configuration is stored in the file **Config.dat**. Do not edit this file manually.

To determine where script and configuration files are located, click on the **Browse Script Folder** button in the **Data Editing** tab of the **Settings** window.

2.10 Link*One Scripting

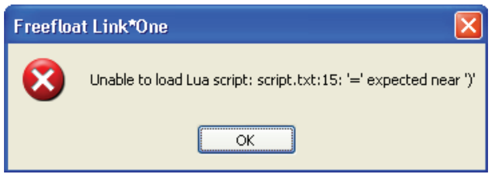
2.10.1 Overview

Link*One has an embedded script language called Lua. When Link*One receives data from a device, a hot key is pressed etc. certain methods in the script are called. The code in these scripts determines what action is taken.

The name of the script file is Script.txt and is placed in the Link*One application data folder. The location of this folder varies depending on what operating system you are using. If you need to make a backup of the script or copy it to another PC, click on the Browse Script Folder... on the Misc tab in the Settings dialog. Windows Explorer is opened and displays the contents of the script folder.

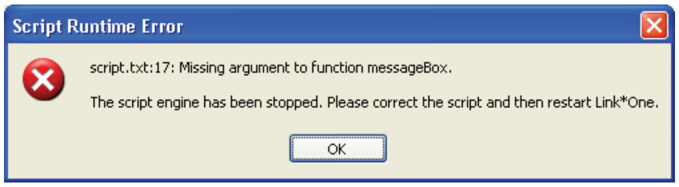
When you edit the script, remember to restart Link*One to recompile the script or use the faster alternative of entering the Settings dialog and then exiting it.

If you make a mistake, for example create a syntax error, an error message is displayed when the script is compiled:



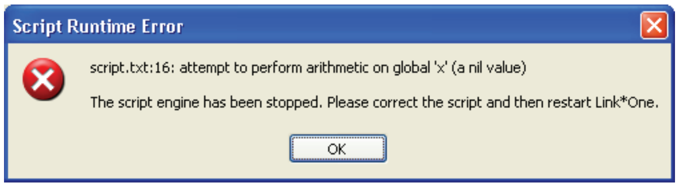
Unable to load Lua script: script.txt:15: '=' expected near ')'

Also, some errors can appear when the script is running, so called runtime errors. Here are a couple of examples:



script.txt:17: Missing argument to function messageBox.

The script engine has been stopped. Please correct the script and then restart Link*One.



script.txt:16: attempt to perform arithmetic on global 'x' (a nil value)

The script engine has been stopped. Please correct the script and then restart Link*One.

2.10.2 Lua Language

From **http://www.lua.org/about.html**:

Lua is a powerful, fast, light-weight, embeddable scripting language.

Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics.

Put simply, Lua is what makes data processing in Link*One very flexible and powerful. The reference manual for Lua can be found at the Lua site:

http://www.lua.org/

There is also a printed book on the Lua language, called Programming in Lua, which is more accessible than the reference manual.

Apart from Lua and its built-in language, Link*One exposes a number of useful methods to the script.

2.10.3 Script Events

When things happen in Link*One, for example a hot key or a data string is received on the serial port, an event is generated. This results in a script method being called. The methods called when events happen are called event methods.

The table below is an overview and short description of all the different event methods. For a more detailed explanation, see the topic Event Methods below.

Event Handler	When Called
onStart	Link*One is started
onEnd	Link*One is exited
onData	A data string is received on the serial port
onHotKey	A hot key is pressed
onKeyboardCapture	A data string is received from a HID device
onExternalData	A data string is received from an external application
onTimer	The timer interval has elapsed
onCTS	Status change on CTS
onDSR	Status change on DTR
onRI	Status change on RI
onDCD	Status change on DCD

2.10.4 Event Methods

In this topic all the event methods are explained in detail.

2.10.4.1 onStart()

This method is called when Link*One is started. It is also called when you exit the **Settings** dialog.

This method receives no arguments.

Example: Beep on start

```
function onStart()
  -- Issue a short beep (3000 Hz, 50 ms)
  app.beep( 3000, 50 )
end
```

2.10.4.2 onEnd()

Called when Link*One is exited. It is also called when you enter the **Settings** dialog.

This method receives no arguments.

Example: Beep on exit

```
function onEnd()
  -- Issue a short beep (1000 Hz, 50 ms)
  app.beep( 1000, 50 )
end
```

2.10.4.3 onData(data, length)

Called when a data string is received from the serial port.

This method receives the data string in data and the length of the string in length.

Data may contain binary characters including the null character.

Please note that if the **Data String Termination** is set to be a character and that character does not match the terminator used by the serial device, this method is never called.

Example: Hex dump of serial data

```
function onData( data, length )
  local numbers = ""
  local text = ""

  -- Loop for each character in data
  for i=1,length do
    -- Append character to text part
    if string.byte( data, i ) >= 32 then
      text = text .. string.sub( data, i, i )
    else
      -- Control characters are replace with '.'
      text = text .. "."
    end

    -- Add hex representation of the character
    numbers = numbers .. string.format( "%02x ", string.byte( data, i ) )

    -- Break lines at eight characters
    if (i % 8) == 0 then
      app.send( numbers .. text .. "{Enter}" )
      numbers = ""
      text = ""
    end
  end

  -- Handle the tail of the dump
  local c = length
  if (c % 8) ~= 0 then
    while (c % 8) ~= 0 do
      numbers = numbers .. "   "
      c = c + 1
    end
    app.send( numbers .. text .. "{Enter}" )
  end
end
```

To test the above example, copy and paste the code into the script replacing the default implementation of onData(). Use Timeout as the Data String Terminator. Connect a serial device to the serial port and make it generate some data. Below is the output when reading a barcode containing “W1711010814107013621” using a serial barcode reader:

```
57 31 37 31 31 30 31 30 W1711010
38 31 34 31 30 37 30 31 81410701
33 36 32 31 0d          3621.
```

The last character 0d (hexadecimal) is the same as 13 in decimal notation. The ASCII character with code 13 is carriage return. This means the barcode reader is using carriage return (<cr>) as its data string terminator.

2.10.4.4 onHotKey(name)

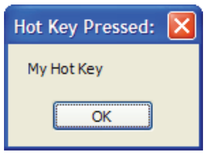
This method is called when you press a hot key.

The argument to this method is the name of the hot key that was pressed.

Example: Message box displaying the hot key’s name

```
function onHotKey( name )
  app.messageBox( "Hot Key Pressed:", name )
end
```

When executed, the above method will display a message box with the name of the hot key:


Hot Key Pressed:

My Hot Key

2.10.4.5 onKeyboardCapture(name, data)

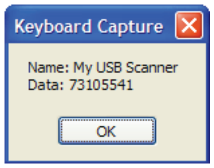
Called when a keyboard capture string has been received.

The arguments to this method are the name of the keyboard capture and the data.

Example: Display the name and data of a keyboard capture event

```
function onKeyboardCapture( name, data )
  app.messageBox( "Keyboard Capture", "Name: " .. name .. "\n" ..
    "Data: " .. data )
end
```

If you have a keyboard captured defined called My USB Scanner and it captures the string 73105541 the method in the above example will display this dialog:


Keyboard Capture

Name: My USB Scanner
Data: 73105541

2.10.4.6 onExternalData(data, length)

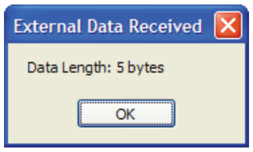
Called when an external application sends data to Link*One.

The arguments are the received data and the length of it.

External applications can send data to Link*One. They do it by finding the window handle of Link*One’s window and then send a WM_COPYDATA message to the window.

Example: Display data and length sent to Link*One from an external application

```
function onExternalData( data, length )
  app.messageBox( "External Data Received", "Data Length: " .. length .. " bytes" )
end
```


External Data Received

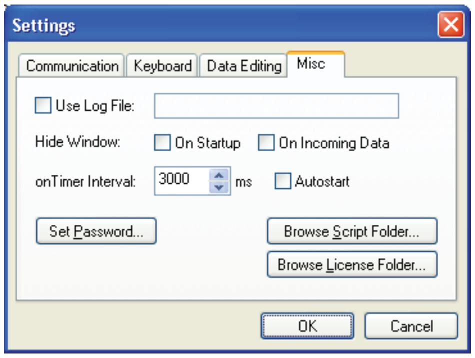
Data Length: 5 bytes

This feature makes it possible to create an application that integrates tightly with Link*One. For example, the code in onExternalData() could relay the data to a scanner to make it beep, initiate a scan, configure it etc.

2.10.4.7 onTimer()

Called periodically at the specified timer interval.

This method receives no arguments.


Settings

☐ Use Log File:

Hide Window: ☐ On Startup ☐ On Incoming Data

onTimer Interval: 3000 ms ☐ Autostart

In the example above, the onTimer Interval has been set to 3000 milliseconds (3 seconds). This means that the onTimer() script method will be called once every three seconds.

This method can be used for adding timeout logic to a solution.

Example: Output a time stamp to the active application at the onTimer interval

```
function onTimer()
  app.send( os.date( "%H:%M:%S" ) .. "{Enter}" )
end
```

2.10.4.8 onCTS(status)

This method is called when the hardware handshake signal changes state.

The argument status is true when the signal goes high and false when the signal goes low.

This description also applies to onDSR, onRI, and onDCD.

Example: Outputs the state of CTS when it is changed

```
function convertSignal( status )
  if status then
    return "High"
  else
    return "Low"
  end
end

function onCTS( status )
  app.send( "CTS " .. convertSignal( status ) .. "{Enter}" )
end
```

2.10.5 Script Methods

Lua is a generic script language and has methods to manipulate string, tables, files, and so on. However, it does not contain any methods to retrieve the title of a window, simulate keys etc. So in order to activate an application window, send key strokes and similar operations, a number of internal methods in Link*One have been exposed to the embedded Lua script engine.

The tables below offers an overview of the methods. The methods have been grouped into areas of interest for easier reference. Below the tables there is a reference section with a detailed description of each method.

When these methods are used in a script, you need to prefix them with "app.", for example app.sleep(100).

Output/User Feedback	
Method	**Description**
beep	Beeps with the internal PC speaker
blinkIcon	Changes the color of the notification area icon
log	Writes a text line to a log file
messageBox	Displays a message box
playSound	Plays a sound file
playSystemSound	Plays a sound associated with a system event
send	Sends keyboard data
sendSerialData	Sends serial data to the COM-port
sendSubscriberData	Sends data to subscribers

Windows	
Method	**Description**
enumWindows	Enumerates all windows
findWindow	Finds a window with the specified title and/or class
getForegroundWindow	Gets the handle of the foreground window
getWindowClass	Gets the class name of a window
getWindowText	Gets the title of a window
setForegroundWindow	Sets a window to be the foreground window
setWindowText	Sets the title of a window
windowOperation	Executes a window operation

Clipboard	
Method	**Description**
getClipboardData	Gets the text on the clipboard
setClipboardData	Sets the text on the clipboard

Application Launch	
Method	**Description**
closeAppHandle	Closes the application handle
isAppRunning	Determines if an application is still running
run	Starts a program

Serial Port	
Method	**Description**
closePort	Closes the serial port
getDTR	Gets the status of the DTR signal
getRTS	Gets the status of the RTS signal
openPort	Opens the serial port
setDTR	Sets the status of the DTR signal
setRTS	Sets the status of the RTS signal

Miscellaneous	
Method	**Description**
ean128	Parses the contents of a GS1-128/UCC/EAN-128 code
exit	Exits Link*One
exitWindows	Either logs off the current user, shuts down the PC, or shuts down and restarts the PC
getProfile	Gets the current Link*One profile
getTickCount	Gets the number of milliseconds elapsed since the system was started
lockWorkStation	Locks Windows
setProfile	Sets the current profile
setTimer	Sets the script timer
sleep	Delays the script for some time

2.10.6 Output/User Feedback

2.10.6.1 beep(frequency, duration)

2.10.6.1.1 Description

Makes the internal PC speaker beep with the specified frequency and duration.

2.10.6.1.2 Arguments

Argument	Type	Description
frequency	Number	The frequency of the beep in Hertz.
duration	Number	The duration of the beep in milliseconds.

2.10.6.1.3 Returns

true if successful, false otherwise.

2.10.6.1.4 Constants

true if successful, false otherwise.

2.10.6.1.5 Example

```
-- Issue a short beep (3000 Hz, 50 ms) when Link*One is started
function onStart()
  app.beep( 3000, 50 )
end
```

2.10.6.2 blinkIcon(icon, duration)

2.10.6.2.1 Description

Changes the notification icon color for the specified duration. After the duration has elapsed the icon will return to the default color grey.

The operation is asynchronous, in other words, if blinkIcon is called again before the duration for the first call has elapsed, the new icon is set immediately by the second call.

2.10.6.2.2 Arguments

Argument	Type	Description
icon	Number	The icon color index.
duration	Number	The duration in milliseconds the color should be displayed before restoring the default color.

2.10.6.2.3 Returns

Nothing.

2.10.6.2.4 Constants

```
iconColors =
{
  ["GREY"] = 0,
  ["GREEN"] = 1,
  ["YELLOW"] = 2,
  ["BLUE"] = 3,
  ["RED"] = 4,
}
```

2.10.6.2.5 Example

Please note the delay in the loop is needed to display each icon color for half a second.

```
-- Cycle through icon colors
function onStart()
  for i = iconColors["GREY"], iconColors["RED"] do
    app.blinkIcon( i, 500 )
    app.sleep( 500 )
  end
end
```

2.10.6.3 log(filename, message)

2.10.6.3.1 Description

Writes the message to the specified log file. Note that logging with this method from the script is separate from the built-in logging facility.

2.10.6.3.2 Arguments

Argument	Type	Description
filename	String	The path and name of the log file to write to.
message	String	The log message to write.

2.10.6.3.3 Returns

Nothing.

2.10.6.3.4 Example

```
function onStart()
  app.log( "c:\\myscript.log", "onStart() called" )
end
```

2.10.6.4 messageBox(title, message, type)

2.10.6.4.1 Description

Displays a message box with the specified title and message. The type argument specifies the number and type of buttons used.

2.10.6.4.2 Arguments

Argument	Type	Description
title	String	The message box title.
message	String	The message to be displayed.
type	Number, optional	The type of buttons to use.

2.10.6.4.3 Returns

A value from the constants in mbResults (see below) indicating the button clicked.

2.10.6.4.4 Constants

Use one of the following constants for the type argument:

```
mbTypes =
{
  ["OK"] = 0,
  ["OKCANCEL"] = 1,
  ["YESNOCANCEL"] = 3,
  ["YESNO"] = 4,
  ["ICONHAND"] = 16,
  ["ICONQUESTION"] = 32,
  ["ICONEXCLAMATION"] = 48,
  ["ICONASTERISK"] = 64,
  ["DEFBUTTON1"] = 0,
  ["DEFBUTTON2"] = 256,
  ["DEFBUTTON3"] = 512,
}
```

The possible return values are:

```
mbResults =
{
  ["OK"] = 1,
  ["CANCEL"] = 2,
  ["YES"] = 6,
  ["NO"] = 7,
}
```

2.10.6.4.5 Example

```
-- Ask the user if she/he wants to exit Link*One and acts on the answer
function onHotKey( name )
  local result = app.messageBox( "Link*One Script", "Are you sure you want to
exit?",
    mbTypes["YESNO"] + mbTypes["DEFBUTTON2"] )

  if mbResults["YES"] == result then
    app.exit()
  end
end
```

2.10.6.5 playSound(filename, options)

2.10.6.5.1 Description

Plays the sound specified by the filename argument.

2.10.6.5.2 Arguments

Argument	Type	Description
filename	String	The path and filename of the sound file.
options	Number	Specifies the behavior for the sound playback. Use the constants in soundOptions.

2.10.6.5.3 Returns

true if successful, false otherwise.

2.10.6.5.4 Constants

```
soundOptions =
{
  ["SYNC"] = 0,
  ["ASYNC"] = 1,
  ["NODEFAULT"] = 2,
  ["LOOP"] = 8,
  ["NOSTOP"] = 16,
  ["NOWAIT"] = 8192,
}
```

2.10.6.5.5 Example

```
-- Buzz like a bee
function onHotKey( name )
  app.playSound(
    "c:\\windows\\system32\\buzzingbee.wav", soundOptions["SYNC"] )
end
```

2.10.6.6 playSystemSound(systemEvent, options)

2.10.6.6.1 Description

Plays the sound specified by the given system event. Different system events can be mapped to sound files in the Control Panel.

2.10.6.6.2 Arguments

Argument	Type	Description
systemEvent	Number	The ID of the system event.
options	Number	Specifies the behavior for the sound playback. Use the constants in soundOptions.

2.10.6.6.3 Returns

true if successful, false otherwise.

2.10.6.6.4 Constants

Use one of the following values for the **systemEvent** argument:

```
systemSounds =
{
  ["ASTERISK"] = 10835,
  ["QUESTION"] = 16211,
  ["HAND"] = 18515,
  ["EXIT"] = 17747,
  ["START"] = 21331,
  ["WELCOME"] = 22355,
  ["EXCLAMATION"] = 8531,
  ["DEFAULT"] = 17491,
}
```

The possible values of the **options** argument:

```
soundOptions =
{
  ["SYNC"] = 0,
  ["ASYNC"] = 1,
  ["NODEFAULT"] = 2,
  ["LOOP"] = 8,
  ["NOSTOP"] = 16,
  ["NOWAIT"] = 8192,
}
```

2.10.6.6.5 Example

```
-- Play the sound mapped to the system event Exclamation
function onHotKey( name )
  app.playSystemSound( systemSounds["EXCLAMATION"], soundOptions["SYNC"] )
end
```

2.10.6.7 send(data)

2.10.6.7.1 Description

Sends keyboard data.

2.10.6.7.2 Arguments

Argument	Type	Description
data	String	The data to send.

The argument data is a string consisting of text, key names, and Unicode characters.

Argument	Type	Description
text	"Rob was here"	Regular characters.
Key Name	"{Enter}"	Key names corresponds to the keys defined in the dialog Key Settings.
Unicode	"{65}"	Character in Unicode decimal notation.
Unicode	"{0x0041}"	Character in Unicode hexadecimal notation.

2.10.6.7.3 Returns

Nothing.

2.10.6.7.4 Constants

None.

2.10.6.7.5 Example

```
function onHotKey( name )
  -- Send a regular string
  app.send( "Rob was here!" )

  -- Send the characters ABC by using Unicode notation
   app.send( "{0x0041}{0x0042}{0x0043}" )

  -- Send enter using its key name
  app.send( "{Enter}" )
end
```

Note:

- When you want send() to send the characters \ and { you need to escape those with a backslash. There is a helper method called escapeData() in the supplied file extra.lua.
- If you specify a key name for a key that is not defined, it will be ignored.
- When sending data to certain applications they might miss key presses if the key events are sent too fast or too early. You may need to increase the setting Interkey Delay and/or intersperse calls to send() with calls to sleep() or findWindow() depending on the situation.

2.10.6.8 sendSerialData(data, length)

2.10.6.8.1 Description

Sends serial data to the COM-port.

Note that Link*One needs to be configured to use a COM-port for this method to work.

2.10.6.8.2 Arguments

Argument	Type	Description
data	String	The data to send to the COM-port.
length	Number	The number of characters of data that should be sent.

2.10.6.8.3 Returns

Nothing.

2.10.6.8.4 Constants

None.

2.10.6.8.5 Example

```
-- Send a binary string to the COM-port
function onHotKey( name )
  -- Build a string containing the characters null, soh, and stx
  local s = string.char( 0 ) .. string.char( 1 ) .. string.char( 2 )
  app.sendSerialData( s, 3 )
end
```

2.10.6.9 sendSubscriberData(data, length)

2.10.6.9.1 Description

Sends data to subscribers.

If there are no subscribers, calling this method has no effect.

2.10.6.9.2 Arguments

Argument	Type	Description
data	String	The data to send to the COM-port.
length	Number	The number of characters of data that should be sent.

2.10.6.9.3 Returns

Nothing.

2.10.6.9.4 Constants

None.

2.10.6.9.5 Example

```
function onHotKey( name )
  -- Send a string to all connected subscribers
  local s = "Hello Subscriber!"

  app.sendSubscriberData( s, string.len( s ) )
end
```

2.10.7 Windows

2.10.7.1 enumWindows(handle)

2.10.7.1.1 Description

Enumerates all windows.

2.10.7.1.2 Arguments

Argument	Type	Description
handle	Number	The handle to the window whose child windows should be enumerated. Specify null (0) to enumerate all top level windows.

2.10.7.1.3 Returns

A table containing all the window handles of the enumerated windows.

2.10.7.1.4 Constants

None.

2.10.7.1.5 Example

```
-- Enumerate all top level windows, place handle values and window titles on clipboard
function onHotKey( name )
  local t = app.enumWindows( 0 )
  local stot = ""

  for k, v in pairs( t ) do
    s = string.format( "%08x", v )
    stot = stot .. s .. ": " .. app.getWindowText( v ) .. "\r\n"
  end

  -- Paste the clipboard into a program to display the result
  app.setClipboardData( stot )
end
```

2.10.7.2 findWindow(title, class)

2.10.7.2.1 Description

Finds a window with the specified title and class.

2.10.7.2.2 Arguments

Argument	Type	Description
title	String	The title of the sought window.
class	String, optional	The window class of the sought window.

2.10.7.2.3 Returns

The window handle if the window is found or null otherwise.

2.10.7.2.4 Constants

None.

2.10.7.2.5 Example

```
function onHotKey( name )
  -- Check if Notepad is running (with no doc name given)
  if app.findWindow( "Untitled - Notepad" ) then
    app.messageBox( "Link*One Script", "Notepad is running" )
  else
    app.messageBox( "Link*One Script", "Notepad is not running" )
  end
end
```

2.10.7.3 getForegroundWindow()

2.10.7.3.1 Description

Gets the handle of the foreground window.

2.10.7.3.2 Arguments

None.

2.10.7.3.3 Returns

The window handle of the foreground window. In special circumstances this can be null so you need to check the return value before further use of it.

2.10.7.3.4 Constants

None.

2.10.7.3.5 Example

```
function onHotKey( name )
  local handle = app.getForegroundWindow()

  if handle ~= 0 then
    app.messageBox( "Link*One Script", app.getWindowText( handle ) )
  end
end
```

2.10.7.4 getWindowClass(handle)

2.10.7.4.1 Description

Gets the class name of the specified window.

2.10.7.4.2 Arguments

Argument	Type	Description
handle	Number	The handle of the window.

2.10.7.4.3 Returns

A string containing the class name of the window or an empty string if the class name couldn't be retrieved.

2.10.7.4.4 Constants

None.

2.10.7.4.5 Example

```
-- Displays the class name of the foreground window
function onHotKey( name )
  local class = app.getWindowClass( app.getForegroundWindow() )
  app.messageBox( "Link*One Script", class )
end
```

2.10.7.5 getWindowClass(handle)

2.10.7.5.1 Description

Gets the title of the specified window. This method also works on child windows such as buttons, edit boxes, and similar controls.

2.10.7.5.2 Arguments

Argument	Type	Description
handle	Number	The handle of the window.

2.10.7.5.3 Returns

A string containing the window title of the window or an empty string if the window text couldn't be retrieved.

2.10.7.5.4 Constants

None.

2.10.7.5.5 Example

```
-- Displays the class name of the foreground window
function onHotKey( name )
  local class = app.getWindowClass( app.getForegroundWindow() )
  app.messageBox( "Link*One Script", class )
end
```

2.10.7.6 getWindowText(handle)

2.10.7.6.1 Description

Gets the title of the specified window. This method also works on child windows such as buttons, edit boxes, and similar controls.

2.10.7.6.2 Arguments

Argument	Type	Description
handle	Number	The handle of the window.

2.10.7.6.3 Returns

A string containing the window title of the window or an empty string if the window text couldn't be retrieved.

2.10.7.6.4 Constants

None.

2.10.7.6.5 Example

```
-- Displays the window title of the foreground window
function onHotKey( name )
  local title = app.getWindowText( app.getForegroundWindow() )
  app.messageBox( "Link*One Script", title )
end
```

2.10.7.7 setForegroundWindow(handle)

2.10.7.7.1 Description

Sets the specified window to be the foreground window.

2.10.7.7.2 Arguments

Argument	Type	Description
handle	Number	The handle of the window.

2.10.7.7.3 Returns

Nothing.

2.10.7.7.4 Constants

None.

2.10.7.7.5 Example

```
-- Bring the Windows Media Player window to the foreground
function onHotKey( name )
 local handle = app.findWindow( "Windows Media Player" )
 if handle ~= 0 then
   app.setForegroundWindow( handle )
 end
end
```

2.10.7.8 getWindowText(handle, text)

2.10.7.8.1 Description

Sets the title of the specified window. This method also works on child windows such as buttons, edit boxes, and similar controls.

2.10.7.8.2 Arguments

Argument	Type	Description
handle	Number	The handle of the window.
text	String	The title to set.

2.10.7.8.3 Returns

true if successful, false otherwise.

2.10.7.8.4 Constants

None.

2.10.7.8.5 Example

```
-- Set a new title for a Notepad window
function onHotKey( name )
 local handle = app.findWindow( "Untitled - Notepad" )
 if handle ~= 0 then
   app.setWindowText( handle, "My Text Editor" )
 end
end
```

2.10.7.9 windowOperation(handle, operation)

2.10.7.9.1 Description

Executes a window operation.

2.10.7.9.2 Arguments

Argument	Type	Description
handle	Number	The handle of the window.
operation	Number	The operation to perform on the window.

2.10.7.9.3 Returns

Nothing.

2.10.7.9.4 Constants

The following constants define the possible values of the operation argument:

```
windowOperations =
{
 ["CLOSE"] = 1,
 ["MAXIMIZE"] = 4,
 ["MINIMIZE"] = 5,
 ["RESTORE"] = 6,
 ["TASKLIST"] = 9,
 ["MONITORPOWER"] = 11,
 ["SCREENSAVE"] = 12,
}
```

2.10.7.9.5 Example

```
-- Maximizes a Notepad window then restores it
function onHotKey( name )
 local handle = app.findWindow( "Untitled - Notepad" )
 if handle ~= 0 then
   app.windowOperation( handle, windowOperations["MAXIMIZE"] )
   app.sleep( 2000 )
   app.windowOperation( handle, windowOperations["RESTORE"] )
 end
end
```

2.10.8 Clipboard

2.10.8.1 getClipboardData()

2.10.8.1.1 Description

Gets the text from the clipboard.

2.10.8.1.2 Arguments

None.

2.10.8.1.3 Returns

The text contents on the clipboard as a string and the length of the data. If the call fails or the clipboard doesn't have any text, an empty string and zero length is returned. Note that the terminating null is counted.

2.10.8.1.4 Constants

None.

2.10.8.1.5 Example

```
-- Displays the text length and content on the clipboard in a message box
function onHotKey( name )
  local text, textlen = app.getClipboardData()
  app.messageBox( "Clipboard Contents",
    string.format( "%d characters\r\n", textlen ) .. text )
end
```

2.10.8.2 setClipboardData(text)

2.10.8.2.1 Description

Sets the text on the clipboard.

2.10.8.2.2 Arguments

Argument	Type	Description
text	String	The text to set.

2.10.8.2.3 Returns

Nothing.

2.10.8.2.4 Constants

None.

2.10.8.2.5 Example

```
-- Sets the text contents of the clipboard and then retrieves it
function onHotKey( name )
 local s = "ABC"
 app.setClipboardData( s )

 local text, textlen = app.getClipboardData()
 app.messageBox( "Clipboard Contents", string.format( "%d characters\r\n", textlen
) .. text )
end
```

2.10.9 Application Launch

2.10.9.1 closeAppHandle(handle)

2.10.9.1.1 Description

Closes the application handle.

2.10.9.1.2 Arguments

Argument	Type	Description
handle	Number	The application handle.

2.10.9.1.3 Returns

Nothing.

2.10.9.1.4 Constants

None.

2.10.9.1.5 Example

See the **run()** method.

2.10.9.2 isAppRunning(handle)

2.10.9.2.1 Description

Determines if an application is still running.

2.10.9.2.2 Arguments

Argument	Type	Description
handle	Number	The application handle.

2.10.9.2.3 Returns

Nothing.

2.10.9.2.4 Constants

true if the application is still running, **false** otherwise.

2.10.9.2.5 Example

See the **run()** method.

2.10.9.3 run(program, argument, delay)

2.10.9.3.1 Description

Displays a message box with the specified title and message. The type argument specifies the number and type of buttons used.

2.10.9.3.2 Arguments

The full path to the executable.	Type	Description
program	String	The full path to the executable.
argument	String	The command line argument string.
delay	Number, optional	The number of milliseconds to wait until the started application is waiting for user input. If this argument isn't specified, the default wait time is 10000 ms.

2.10.9.3.3 Returns

The application handle. Note that this handle needs to be closed with the method **closeAppHandle()** to avoid memory leaks.

If the application couldn't be started, a runtime error occurs.

2.10.9.3.4 Constants

None.

2.10.9.3.5 Example

```
function onHotKey( name )
  local appHandle = app.run( "c:\\windows\\notepad.exe", "c:\\test.txt" )

  -- As an extra precaution
  app.sleep( 500 )

  -- Send some text to Notepad
  app.send( "Hi!" )

  -- Wait until the user exits Notepad
  while app.isAppRunning( appHandle ) do
    app.sleep( 100 )
  end

  -- Close the handle
  app.closeAppHandle( appHandle )

  -- Confirm exit with a message
  app.messageBox( "Link*One Script", "Notepad is dead" )
end
```

2.10.10 Serial Port

2.10.10.1 closePort()

2.10.10.1.1 Description

Closes the serial port.

2.10.10.1.2 Arguments

None.

2.10.10.1.3 Returns

Nothing.

2.10.10.1.4 Constants

None.

2.10.10.1.5 Example

See the **openPort()** method.

2.10.10.2 getDTR()

2.10.10.2.1 Description

Gets the status of the DTR signal of the serial port. DTR is an output signal.

2.10.10.2.2 Arguments

None.

2.10.10.2.3 Returns

A boolean which indicates the DTR signal status (**true** = high, **false** = low).

2.10.10.2.4 Constants

None.

2.10.10.2.5 Example

```
-- Display the status of the DTR signal
function displayDTRStatus()
  local s = "off"

  if app.getDTR() then
    s = "on"
  end

  app.messageBox( "Link*One Script", "DTR is " .. s )
end

function onHotKey( name )
  -- Display default (from settings)
  displayDTRStatus()

  -- Modify status and display it
  app.setDTR( false )
  displayDTRStatus()
  app.setDTR( true )
  displayDTRStatus()
end
```

2.10.10.3 getRTS()

2.10.10.3.1 Description

Gets the status of the RTS signal of the serial port. RTS is an output signal.

2.10.10.3.2 Arguments

None.

2.10.10.3.3 Returns

A boolean which indicates the RTS signal status (**true** = high, **false** = low).

2.10.10.3.4 Constants

None.

2.10.10.3.5 Example

See the **getDTR()** method.

2.10.10.4 openPort()

2.10.10.4.1 Description

Opens the serial port.

openPort() and **closePort()** can be used when you need to release the serial port, start an application that uses the port for a while, and then reopen the port.

2.10.10.4.2 Arguments

None.

2.10.10.4.3 Returns

A boolean indicating if the port could be opened. If the port is being held open by another application a call to this method will return false.

2.10.10.4.4 Constants

None.

2.10.10.4.5 Example

Note: This is only a code fragment.

```
function onHotKey( name )
  -- Close the serial port to let the external application use it
  app.closePort()

  Start external application and wait for it to exit

  -- Reopen the serial port
  app.openPort()
end
```

2.10.10.5 setDTR(status)

2.10.10.5.1 Description

Sets the status of the DTR signal. DTR is an output signal.

2.10.10.5.2 Arguments

Argument	Type	Description
status	Boolean	The status to set, (**true** = high, **false** = low).

2.10.10.5.3 Returns

Nothing.

2.10.10.5.4 Constants

None.

2.10.10.5.5 Example

See the **getDTR()** method.

2.10.10.6 setRTS(status)

2.10.10.5.1 Description

Sets the status of the RTS signal. RTS is an output signal.

2.10.10.5.2 Arguments

Argument	Type	Description
status	Boolean	The status to set, (**true** = high, **false** = low).

2.10.10.5.3 Returns

Nothing.

2.10.10.5.4 Constants

None.

2.10.10.5.5 Example

See the **getRTS()** method.

2.10.11 Miscellaneous

2.10.11.1 ean128(data, strict)

2.10.11.1.1 Description

Parses the contents of a GS1-128 code (earlier called UCC-128 or EAN-128). For variable length fields followed by another field, the data must be delimited by a Group Separator (GS, ASCII 29, hex 1D).

Please refer to **http://www.gs1.org** for information about GS1 Application Identifiers.

2.10.11.1.2 Arguments

Argument	Type	Description
data	String	The GS1-128 data to be parsed and split into separate fields.
strict	Boolean	In strict mode, spaces are not allowed in alphanumeric fields.

2.10.11.1.3 Returns

A table where the keys are the Application Identifiers (AIs) and the values are the contents of the fields. If the parsing fails, a nil value is returned. The parsing can fail if the code is not a GS1 code or if the code doesn't follow the standard.

2.10.11.1.4 Constants

None.

2.10.11.1.5 Example

```
-- Parses and outputs a list of AIs and values
function onData( data, length )
  -- Parse the GS1 code
  fields = app.ean128( data, true )

  if fields then
    -- Output AIs and values
    for k,v in pairs(fields) do
      app.send( "AI: " .. k .. "  Value: " .. v .. "{Enter}" )
    end
    app.send( "{Enter}" )
  else
    app.messageBox( "Link*One Script", "GS1 parsing failed." )
  end
end
```

2.10.11.2 exit()

2.10.11.2.1 Description

Exits Link*One. Please note that the exit is not immediate, Link*One will exit as soon as the current script has finished executing.

2.10.11.2.2 Arguments

None.

2.10.11.2.3 Returns

Nothing.

2.10.11.2.4 Constants

None.

2.10.11.2.5 Example

```
function onHotKey( name )
  -- Will exit Link*One as soon as this method has finished executing
  app.exit()

  app.messageBox( "Link*One Script", "Goodbye!" )
end
```

2.10.11.3 exitWindows(options)

2.10.11.3.1 Description

Either logs off the current user, shuts down the PC, or shuts down and restarts the PC.

2.10.11.3.2 Arguments

Argument	Type	Description
options	Number, optional	Type of action to be performed.

2.10.11.3.3 Returns

Nothing.

2.10.11.3.4 Constants

The following constants define the type of action to be performed:

```
exitWindowsOpts =
{
 ["LOGOFF"] = 0,
 ["SHUTDOWN"] = 1,
 ["REBOOT"] = 2,
 ["FORCE"] = 4,
 ["POWEROFF"] = 8,
}
```

Please note that the default value for options is Logoff (0). The Force (4) value must be used in combination with Logoff (0), Shutdown (1), Reboot (2), or Poweroff (8). Use Force (4) with care since it will end applications without asking the user to save data.

2.10.11.3.5 Example

```
-- Log off user
function onHotKey( name )
 app.exitWindows( exitWindowsOpts["LOGOFF"] )
end
```

2.10.11.4 getProfile()

2.10.11.4.1 Description

Gets the current Link*One profile.

2.10.11.4.2 Arguments

None.

2.10.11.4.3 Returns

A string containing the name of the current profile.

2.10.11.4.4 Constants

None.

2.10.11.4.5 Example

```
-- Display the current profile's name
function onHotKey( name )
  app.messageBox( "Link*One Script", "Current Profile: " .. app.getProfile() )
end
```

2.10.11.5 getTickCount()

2.10.11.5.1 Description

Gets the number of milliseconds elapsed since the system was started. This method can for example be used to take time between events in Link*One.

2.10.11.5.2 Arguments

None.

2.10.11.5.3 Returns

The number of milliseconds elapsed since the system was started.

2.10.11.5.4 Constants

None.

2.10.11.5.5 Example

```
lastTime = 0

-- Displays the time between each hot key event
function onHotKey( name )
 -- Is this the first time the event happens?
 if lastTime == 0 then
   app.send( "First event.{Enter}" )
 else
   -- Display the time elapsed since last time this event happened
   local timeElapsed = app.getTickCount() - lastTime
   app.send( timeElapsed .. " ms{Enter}" )
 end

 -- Remember time stamp for future calls
 lastTime = app.getTickCount()
end
```

2.10.11.6 lockWorkStation()

2.10.11.6.1 Description

Locks Windows.

2.10.11.6.2 Arguments

None.

2.10.11.6.3 Returns

true if successful, **false** otherwise.

2.10.11.6.4 Constants

None.

2.10.11.6.5 Example

```
-- Locks the Windows session
function onHotKey( name )
 app.lockWorkStation()
end
```

2.10.11.7 setProfile(profile)

2.10.11.7.1 Description

Sets the current profile in Link*One.

Please note that a profile change reinitializes the Lua script engine and because of this, any information held in global variables will be lost. If you need any information to survive over a profile switch, you will need to store it in a file.

The actual switch is delayed until the script has finished executing the current method.

2.10.11.7.2 Arguments

Argument	Type	Description
profile	String	The name of the profile to switch to.

2.10.11.7.3 Returns

Nothing.

2.10.11.7.4 Constants

None.

2.10.11.7.5 Example

```
myVar = 0

function onStart()
  app.messageBox( "onStart()", "myVar is " .. myVar )
  myVar = myVar + 1
end

-- Switches to the profile "My Profile"
function onHotKey( name )
  app.setProfile( "My Profile" )
end
```

2.10.11.8 setTimer(interval)

2.10.11.8.1 Description

Sets the script timer to the specified interval. This is the same setting as on the Misc tab in the Settings dialog. To turn off the timer, specify zero as the interval.

2.10.11.8.2 Arguments

Argument	Type	Description
interval	Number	The timer interval to set.

2.10.11.8.3 Returns

Nothing.

2.10.11.8.4 Constants

None.

2.10.11.8.5 Example

```
-- When a hot key is pressed, sets the script timer interval
function onHotKey( name )
 app.setTimer( 1000 )
end
```

2.10.11.9 sleep(duration)

2.10.11.9.1 Description

Delays the script for the specified time.

2.10.11.9.2 Arguments

Argument	Type	Description
duration	Number	The time to wait.

2.10.11.9.3 Returns

Nothing.

2.10.11.9.4 Constants

None.

2.10.11.9.5 Example

```
-- Outputs two periods with a one second pause between them
function onHotKey( name )
 app.send( "." )
 app.sleep( 1000 )
 app.send( "." )
end
```

2.10.12 Notification Area Icon

When started, Link*One adds an icon to the notification area (also called the Systray sometimes). It is used to indicate different states and events. Please note that the icon can also be modified by a script.

Appearance	Explanation
⬜	Link*One is idle.
🟩	Data was received from the serial port.
🟨	Data was sent to the serial port.
🟦	A serial hardware pin event was triggered. **OR** Data was received through a keyboard capture definition.
🟥	The serial port specified in the profile could not be opened.

2.10.13 Migration guide WLinq 3.x to Link*One

Link*One is based on the earlier product called WLinq. Many functions present in WLinq (3.x) has been removed in Link*One. The reason for this is to simplify Link*One and to avoid confusion if there are more than one way to achieve a task.

This guide is meant to ease the transition from WLinq 3.x data formats to the new script based approach in Link*One. Other types of features that has been affected are also explained in this chapter.

2.10.13.1 Duplicate String Filter

The **Duplicate String Filter** function has been removed from the **Communications** tab in the **Settings** window. The equivalent function can be achieved in a script:

```
duplicateFilterTime = 1000
timeStamp = app.getTickCount()
lastCode = ""

function duplicateFilter( data )
  -- Calculate the time elapsed since last code was read
  elapsed = app.getTickCount() - timeStamp

  -- Do not filter the code if:
  --  the code is different OR
  --  the time elapsed since last code was read is greater than the duplicate filter time OR
  --  the timer has wrapped around (extremely rare)
  if (lastCode ~= data) or (elapsed > duplicateFilterTime) or (elapsed < 0) then
    -- Update last code and time
    lastCode = data
    timeStamp = app.getTickCount()

    -- Do not filter the code
    return false
  else
    -- Filter the code
    return true
  end
end

function onData( data, length )
  if not duplicateFilter( data ) then
    app.send( data .. "{Enter}" )
  end
end
```

2.10.13.2 Case Setting

The **Case Setting** function has been removed from the **Keyboard** tab in the **Settings** window. To achieve the same in a script, use the appropriate sample from below:

```
-- Normal Case
function onData( data, length )
  app.send( data .. "{Enter}" )
end
```

```
-- Upper Case
function onData( data, length )
  app.send( string.upper( data ) .. "{Enter}" )
end
```

```
-- Lower Case
function onData( data, length )
  app.send( string.lower( data ) .. "{Enter}" )
end
```

2.10.13.3 Character Translation

In WLinq 3.x, the only way to have WLinq press special keys like Home, Page Down, and similar was awkward. First you had to choose a character position, then redefine that position to map the character to for example the Home key. Then in the data output format, you had to use that character in the output string, for example: Input() + "\x81".

Link*One has no character translation table, instead you can record a custom key sequence and give it a name. See Section **2.9.4.2.2 Key Settings** for instructions on creating a custom key sequence.

The key name can then be used as an expression in the string passed to the **send()** method:

```
function onData( data, length )
  app.send( "{Page Down}" .. data )
end
```

2.10.13.4 Send Pre- and Postfix Keys

This feature mainly existed for the integration of WLinq to Freefloat Access*One. When activated, the key sequence Ctrl + Alt + 1 was sent before the data string and Ctrl + Alt + 2 was sent after the data string. It enabled Access*One to distinguish between keyboard and barcode data. To achieve the same result, record the key sequences and given them the names {Prefix} and {Postfix} and then use them in an expression:

```
function onData( data, length )
  app.send( "{Prefix}" .. data .. "{Postfix}" )
end
```

2.10.13.5 Lock Output Window

The **Lock Output Window** function can be implemented in a script. The following script only sends data to a window who's title contain the text "- Notepad":

```
function onData( data, length )
  windowTitle = app.getWindowText( app.getForegroundWindow() )
  if string.find( windowTitle, "- Notepad" ) then
    app.send( data .. "{Enter}" )
  end
end
```

2.10.13.6 Initialization String

The **Initialization String** can be used for sending a command to for example a barcode scanner that needs some enabling or configuration command at startup.

In Link*One, the following script could instead be used to send commands to the equipment attached to the serial port:

```
function onStart()
  output = "Hello scanner!"
  app.sendSerialData( output, string.len( output ) )
end
```

Note: **onStart()** is called when a profile is activated. This happens when Link*One starts but also when you click **OK** in the **Settings** dialog. Similarly, **onEnd()** is called when Link*One is exited and also when the Settings dialog is entered by clicking the Settings button in the main window.

2.10.13.7 Filter Unknown Data Strings

In WLinq 3.x, if no data editing format matched the input data, the option Filter Unknown Data Strings determined whether the input data should be discarded or let through unmodified. The same effect can easily be implemented in a Link*One script:.

```
function onData( data, length )
  if string.find( data, "K06", 1, true ) then
    app.send( data .. "{Enter}" )
  end
end
```

The above script makes Link*One filter all input data that doesn't start with the characters K06.

2.10.13.8 Input Data Replacements

The replacement function in earlier WLinq versions was quite easy to use. But it lacked power and flexibility. Below is an example of a simple substring replacement. It replaces all occurrences of the character K with the character X.

```
function onData( data, length )
  app.send( string.gsub( data, "K", "X" ) .. "{Enter}" )
end
```

Multiple replacements can be done by storing the result in a string variable and repeat the process. Here K is replaced with X and A is replaced with TEST.

```
function onData( data, length )
  local result = string.gsub( data, "K", "X" )

  result = string.gsub( result, "A", "TEST" )

  app.send( result .. "{Enter}" )
end
```

2.10.13.9 Criteria

In the previous WLinq generation, a data format was activated for a data string when the criteria of the data format matched. Two types of criteria were supported, length and pattern.

In Link*One the same function as a length criteria is implemented using an if-statement.

```
function onData( data, length )
  if (length >= 9) and (length <= 13) then
    app.send( data .. "{Enter}" )
  end
end
```

An alternative approach could be:

```
function onData( data, length )

  Code modifying the contents of data so that length is no longer valid

  if (string.len( data ) >= 9) and (string.len( data ) <= 13) then
    app.send( data .. "{Enter}" )
  end
end
```

A pattern criteria like the one above could be implemented using the string.find() pattern matching method:

```
function onData( data, length )
  if string.find( data, "K06.*F" ) then
    app.send( data .. "{Enter}" )
  end
end
```

The format used for patterns in string.find() and the format in WLinq 3.x is different. Please refer to the Lua documentation for the Lua pattern format.

A big advantage with scripting in Link*One is that more complex decisions can be made, for example mixing length and pattern matching, something that was not possible in WLinq 3.x. Multiple criteria used in WLinq 3.x can be implemented by chaining if-elseif-statements:

```
function onData( data, length )
  if length == 9 then
    app.send( "Nine characters: " .. data .. "{Enter}" )
  elseif length == 13 then
    app.send( "Thirteen characters: " .. data .. "{Enter}" )
  else
    app.send( "Not 9 and not 13: " .. data .. "{Enter}" )
  end
end
```

2.10.13.10 Data Format Output

In a WLinq 3.x data format, expressions was entered into the data format output edit box and combined with plus (+). In Link*One, all the string operations are using the facilities of the embedded script language. To make Link*One simulate a possibly modified string as keyboard output, you need to pass the string to the method **app.send()**.

Use the table below as a guide for converting expressions in WLinq 3.x to Link*One. Most string operations in WLinq 3.x operated on the data input string implicitly. In Link*One, the data string is an argument sent to the script methods **onData()**, **onKeyboardCapture()**, and **onExternalData()**.

Constant String	
WLinq 3.x	**Link*One**
"ABC"	"ABC"

Extract a substring from the start of the string	
WLinq 3.x	**Link*One**
Left(3)	string.sub(data, 1, 3)

string.sub(data, -3)	
WLinq 3.x	**Link*One**
Right(3)	string.sub(data, -3)

Extract characters from position three up to position four. Please note the difference in parameters!	
WLinq 3.x	**Link*One**
Mid(3, 2)	string.sub(data, 3, 4)

From the first A in the string, extract five characters including the A	
WLinq 3.x	**Link*One**
Mid("A", 5)	string.gsub(data, ".*(A....).*", "%1")

Extracts characters from position six to the end of the string	
WLinq 3.x	**Link*One**
Mid(6)	string.sub(data, 6)

Scans for the first string and extracts all characters up to the second string. 23 and CD is not included in the result.	
WLinq 3.x	**Link*One**
SubStr("23", "CD")	string.gsub(data, ".*23(.*)CD.*", "%1")

The entire data string	
WLinq 3.x	**Link*One**
Input()	data

Inserts the current date in the specified format	
WLinq 3.x	**Link*One**
Date("%Y-%m-%d")	os.date("%Y-%m-%d")

Inserts the current time in the specified format

WLinq 3.x	Link*One
Time("%H:%M")	os.date("%H:%M")

Concatenations of expressions

WLinq 3.x	Link*One
"X" + Left(2) + Right(2)	"X" .. string.sub(data, 1, 2) .. string.sub(data, -2)

Control characters

WLinq 3.x	Link*One
"<cr><tab>"	"{13}{9}"
"\x09"	"{9}"
"\d013"	"{13}"

Combining text and key presses

WLinq 3.x	Link*One
Input() + "<tab>1<cr>"	data .. "{Tab}1{Enter}"

Reboot Windows

WLinq 3.x	Link*One
Reboot()	app.exitWindows(exitWindowsOpts["REBOOT"])

Reboot Windows (forced)

WLinq 3.x	Link*One
RebootForced()	app.exitWindows(exitWindowsOpts["FORCE"])

Starts the specified program

WLinq 3.x	Link*One
RunApp("notepad.exe")	h = app.run("notepad.exe") closeAppHandle(h)

Please note that the Link*One sample code below more realistically demonstrates what is needed when switching to another application. A small delay is needed before sending input to the activated window or characters may be lost. Also the example avoids an unnecessary delay when the target window already is the foreground window.

Activates the first window that has a caption that matches the window caption pattern

WLinq 3.x	Link*One
SetFocus("*Notepad")	

```
function setForegroundWindow( pattern )
  local t = app.enumWindows( 0 )
  for k,v in pairs(t) do
    local title = app.getWindowText(v)
    if string.match( title, pattern ) then
      app.setForegroundWindow( v )
      return
    end
  end
end

function onData( data, length )
  local pattern = ".*Notepad"
  local curWindow = app.getForegroundWindow()
  local title = app.getWindowText( curWindow )
  if not string.match( title, pattern ) then
    setForegroundWindow( pattern )
    app.sleep( 250 )
  end
  app.send( data .. "{Enter}" )
end
```

There is no direct equivalent function for the WLinq 3.x WaitForWindow. Below is a full example of a script which waits for a Notepad window to appear, activates the window, and after a small delay sends the data to the window.

Some common situations where you need to wait for a window are when waiting for an Open dialog to appear (after sending Ctrl+O) or when you have launched an application with app.run() and need to wait for it to be ready to receive input.

Wait for a window to appear

WLinq 3.x	Link*One
WaitForWindow("*Notepad", 3000)	

```
function waitForWindow( pattern, waittime )
  local maxwaittime = app.getTickCount() + waittime
  local found = false

  while (app.getTickCount() < maxwaittime) and not found do
    local t = app.enumWindows( 0 )

    for k,v in pairs(t) do
      if string.match( app.getWindowText( v ), pattern ) then
        found = true
        break
      end
    end
  end

  return found
end

function onData( data, length )
  local pattern = ".*Notepad"
  if waitForWindow( pattern, 5000 ) then
    setForegroundWindow( pattern )
    app.sleep( 250 )
    app.send( data .. "{Enter}" )
  end
end
```

No direct equivalent function for WaitForAppExit() exists in Link*One. The same result can be achieved by using app.isAppRunning().

Even though the sample below demonstrates a script that pauses until you exits Notepad, Link*One is not intended to have a script that interact with the user (except for app.messageBox()) since there may be side effects.

app.isAppRunning() is intended to be used to synchronize the script with an external application that does its job and then exits.

Wait for a window to appear

WLinq 3.x	Link*One
WaitForAppExit(30000)	

```
function onData( data, length )
 -- Launch Notepad, if it could not be started, the script is aborted
 -- which means there is no need to check the handle
 local appHandle = app.run( "notepad.exe" )

 -- Wait until Notepad is exited
 while app.isAppRunning( appHandle ) do
  app.sleep( 100 )
 end

 -- Close the handle to avoid leaks
 app.closeAppHandle( appHandle )

 -- Tell the user we are done
 app.messageBox( "Link*One", "Notepad is gone!" )
end
```

If a script calls **app.closePort()**, the script can start an external application that uses the same serial port. When that external application is exited, the script can re-open the serial port by calling **app.openPort()**.

Open the serial port	
WLinq 3.x	**Link*One**
OpenPort()	app.openPort()

Close the serial port	
WLinq 3.x	**Link*One**
ClosePort()	app.closePort()

Send data to the serial port	
WLinq 3.x	**Link*One**

SendData("abc")	data = "abc"
	app.sendSerialData(data, data:len())

Note: The profile switch is performed when the script has finished its execution.

Switch profile	
WLinq 3.x	**Link*One**
SetProfile("Profile2")	app.setProfile("Profile2")

2.10.14 Support for Thin Clients, Java Applications, and Flash Applications

Normally, Link*One uses a Windows API function called SendInput to simulate key presses to the active application. This API is recommended by Microsoft because it takes care of differences between different keyboard locales. For example, on a French keyboard, the letter A is positioned where the letter Q is on a US/UK keyboard layout.

However, this technique of simulating keys doesn't work with all environments and applications used on the Windows platform. So far, problems have been spotted with thin clients (Terminal Services or Citrix), Java applications, and Flash applications.

To address this issue, key sequences for digits, lower case and upper case letters has been recorded and is present in the default configuration of Link*One. Script functions for translating digits and letters to key sequences are provided in **extra.lua**. Also, the function **sendData** in the default script.txt contains information about how to activate this feature.

Please note that the key sequences are tailored for the most common keyboard layouts, QWERTY with non-shifted keys for digits. You need to modify some of these key sequences to make it work on for example AZERTY (French) and QWERTZ (German) keyboard layouts.

2.10.15 Lua Copyright

2.10.16 Version History

Version	Changes	Date
5.0	Major upgrade, product name changed, and OEM version created.	2008-11-26
5.1	Changed name of Lua DLL file to make it compatible with a Lua addon. Fixed problem with empty hot key sequences.	2009-03-06
5.2	Added the wizard. Updates to the manual.	2009-04-02
5.3	When "{65}" was used as a character notation there was a bug in the parser that sometimes made the character disappear. On slow systems with a single CPU core there could be a race condition between the main program module and the licenser module.	2009-10-29
5.4	Added 30-day trial period. Added scan codes for all default key sequence definitions. This was done to avoid problems sending Tab and similar keys in a thin-client environment.	2010-10-05
5.5	OEM version released for testing.	2010-12-22
5.6	OEM version with extra tab in Settings dialog for easier configuration of scanner key.	2011-05-05
5.7	Now blocks key sequences named {numbers} since they won't work. The reason is that the syntax {number} is used as a format to specify the ASCII/Unicode code for characters in the string sent to app.send(). Added built-in support for applications/environments not supporting the way Link*One simulates most keys. Added appendix that explains how this feature is activated and how it works.	2011-05-10
5.8	Removed remnants of code for overlapped I/O that wasn't used. It made third-party serial drivers upset and caused Link*One to hang. This problem was noticed when trying to use Link*One together with BlueSoleil Bluetooth software.	2011-11-01

3.0 Operating the Unit

3.1 GPS Instructions

3.1.1 Requirements:

The WWAN module of the M9010 includes a GPS.

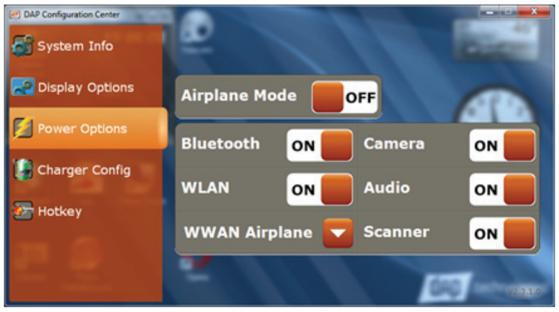
3.1.2 Set up to use the GPS

To use the GPS, the WWAN module must be either fully turned on, or in airplane mode.

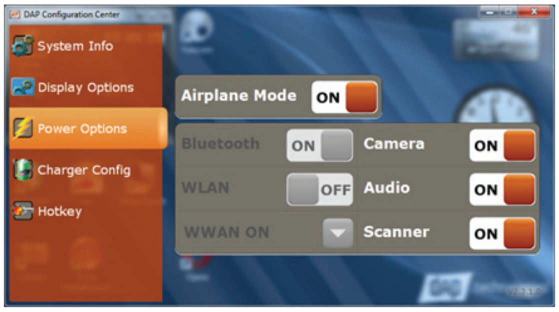
To ensure WWAN is **ON** or in **Airplane Mode**:

1. Open DAP Configuration Center
2. Select the **Power Options** tab
3. Ensure you are in one of the following combinations:
   - Global airplane mode is **OFF** and WWAN is **ON** or in **Airplane Mode**
   - Global airplane mode is **ON** and WWAN was either **ON** or in **Airplane Mode**

Here are some acceptable settings:

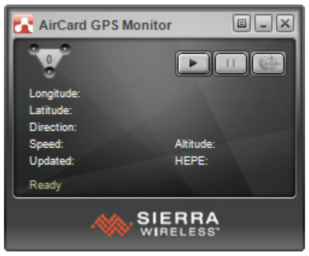


Airplane Mode OFF
Bluetooth ON
Camera ON
WLAN ON
Audio ON
WWAN Airplane
Scanner ON



Airplane Mode ON
Bluetooth ON
Camera ON
WLAN OFF
Audio ON
WWAN ON
Scanner ON

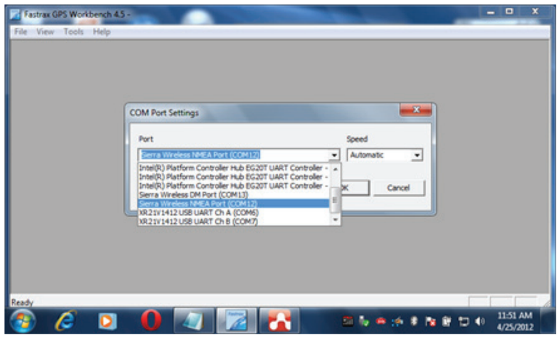
4. Open the GPS monitor (Fn + F6), and click on the 'play' button:



Longitude:
Latitude:
Direction:
Speed: Altitude:
Updated: HEPE:
Ready

You'll get **Session started, waiting for fix...**

5. To get more detailed information:
   a. Run **Fastrax GPS Workbench** available from the desktop
   b. Select **File > Connect**:



COM Port Settings

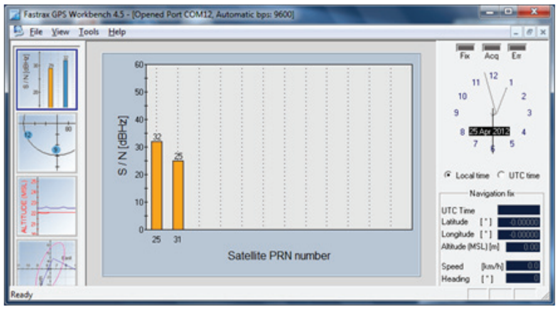
Port: Sierra Wireless NMEA Port (COM12)

Speed: Automatic

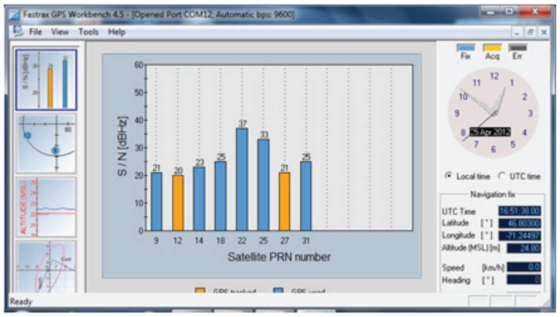
Intel(R) Platform Controller Hub EG20T UART Controller -
Intel(R) Platform Controller Hub EG20T UART Controller -
Intel(R) Platform Controller Hub EG20T UART Controller -
Sierra Wireless DM Port (COM13)
Sierra Wireless NMEA Port (COM12)
XR21V1412 USB UART Ch A (COM6)
XR21V1412 USB UART Ch B (COM7)

OK Cancel

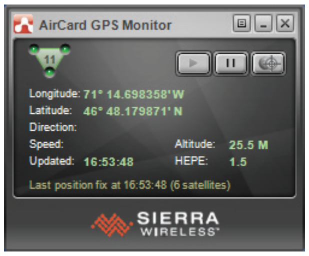
6. Select **Sierra Wireless NMEA Port (COM12)**, then **OK**. The following screenshot shows 2 satellites are detected (not enough):



Once the GPS is able to get a fix (enough satellites), Fastrax shows something like:



The GPS Monitor shows:



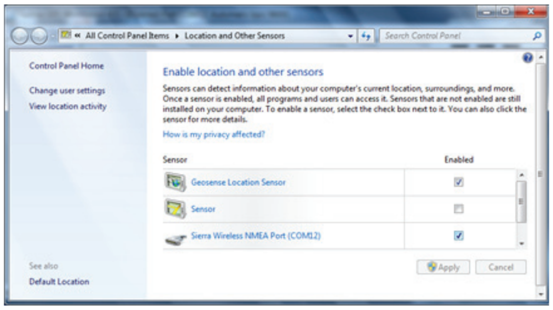
Longitude: 71° 14.698358' W
Latitude: 46° 48.179871' N
Direction:
Speed: Altitude: 25.5 M
Updated: 16:53:48 HEPE: 1.5
Last position fix at 16:53:48 (6 satellites)

7. Clicking on the third button shows a map.
8. GPS is now ready.

3.0 Operating the Unit

3.1.3 Integration to Windows 7

The GPS is handled as a standard sensor in Windows 7:



All Control Panel Items ▸ Location and Other Sensors

Search Control Panel

Control Panel Home

Change user settings

View location activity

See also

Default Location

Enable location and other sensors

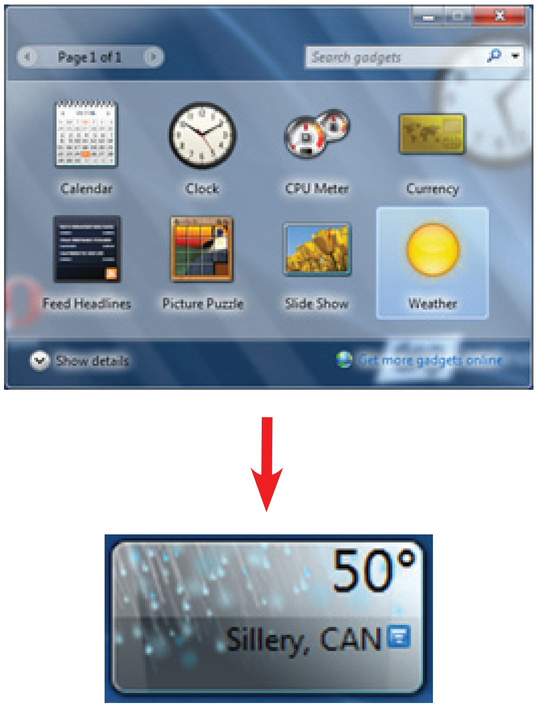
Sensors can detect information about your computer's current location, surroundings, and more. Once a sensor is enabled, all programs and users can access it. Sensors that are not enabled are still installed on your computer. To enable a sensor, select the check box next to it. You can also click the sensor for more details.

How is my privacy affected?

Sensor	Enabled
Geosense Location Sensor	✓
Sensor	☐
Sierra Wireless NMEA Port (COM12)	✓

Apply Cancel

If the Weather gadget is opened, it will update with a city nearby.

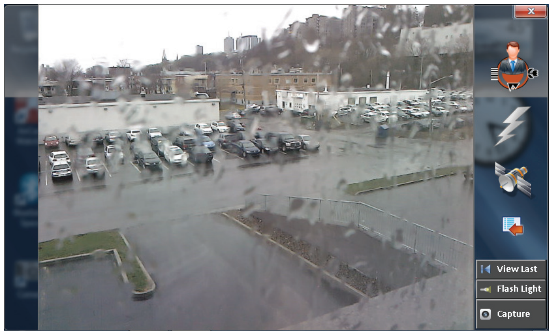


DAP-Imager uses the same method to get the current position in order to geotag images.

3.2 DAP-Imager Instructions

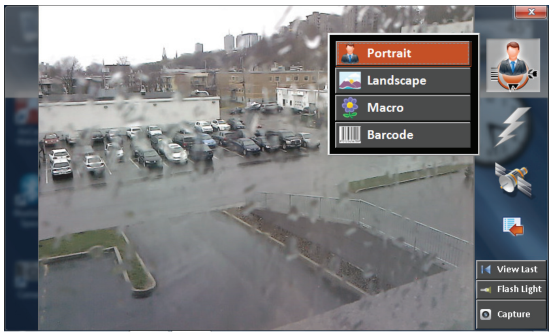
3.2.1 What is DAP-Imager

DAP-Imager allows taking pictures using the built-in camera. It also features a barcode decoding engine to read 1D and 2D barcodes, usually used with ScannerManager.



3.2.2 Selecting the Right Mode

The upper right icon (or lower left if unit is in portrait) is used to select the mode:



The portrait, landscape and macro modes are used to take pictures, whereas barcode mode is used to read barcodes. Mode settings are defined in the INI file. Refer to that section for more details.

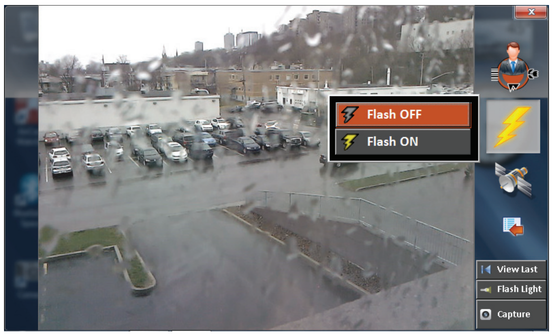
3.2.3 Pictures

3.2.3.1 How to Take a Picture

To take a picture, press and release the trigger button on the back of the unit. Alternatively, you can click on **Capture**.

3.2.3.2 Flash

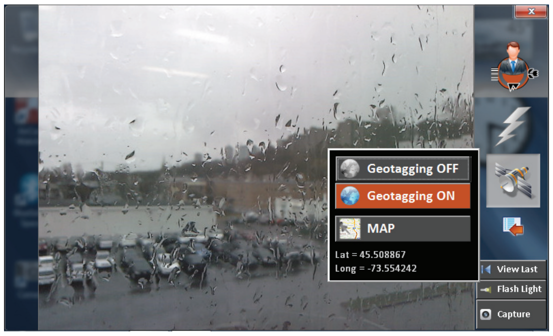
The flash can be turned on or off using the flash menu.



No automatic flash is supported at this time.

3.2.3.3 Geotagging

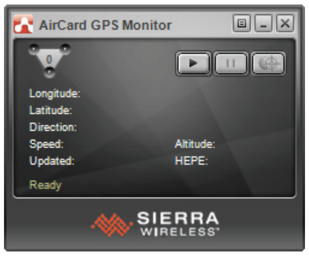
The geotagging menu allows enabling the feature and showing a map centered on the current position. The current coordinates are written at the bottom of the menu.



Once a picture has been taken, the current location is saved as an EXIF metadata in the JPEG file (GPS sub-IFD).

3.2.3.3.1 How to enable the GPS

DAP-Imager has been built to work with the standard location sensors supported by Windows 7. The WWAN module is equipped with a GPS, which maps to "Sierra Wireless NMEA Port" in the "Location and Other Sensors" section of the control panel. You may also use the "Geosense Location Sensor" that retrieves the current position by looking to the WLAN used.



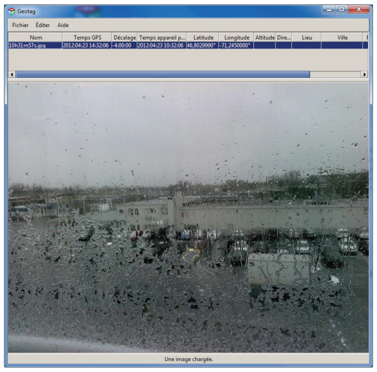
Refer to Section **3.1 GPS Instructions** for more information.

Once a fix is available, DAP-Imager should show the current position in the geotagging menu.

Refer to the GPS section of the user's manual for more information on troubleshooting the GPS.

3.2.3.3.2 How to View Geotagging Data

To view the location where the picture has been taken, you may use any geotagging software. For example, the "geotag" software is an open source java program that can be run from the web (open **http://geotag.sourceforge.net**, then click on "Run it now").

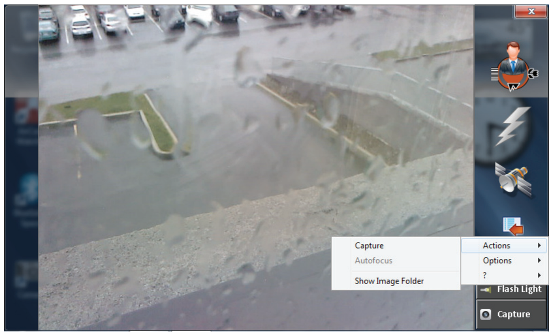


Nom	Temps GPS	Décalage	Temps appareil p...	Latitude	Longitude	Altitude	Dire...	Lieu	Ville
10h31m57s.jpg	2012:04:23 14:32:06	-4:00:00	2012:04:23 10:32:06	46,8029990°	-71,2450000°				

The coordinates are shown on top after having added the file to the list.

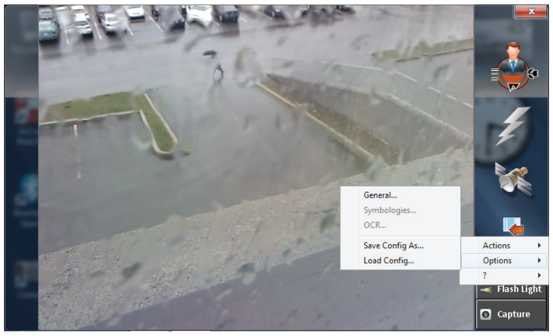
3.2.3.4 How to Locate a Saved Picture

To open the folder where images are saved, choose "Actions > Show Image Folder" from the 'more' menu.

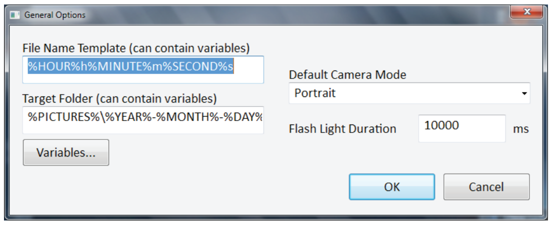


3.2.3.5 General Options

To access the general options, choose "Options > General…" from the 'more' menu:



Option screen:



File Name Template (can contain variables): %HOUR%h%MINUTE%m%SECOND%s

Target Folder (can contain variables): %PICTURES%\%YEAR%-%MONTH%-%DAY%

Default Camera Mode: Portrait

Flash Light Duration: 10000 ms

The file name and target folder templates can be changed here. They define where the image is to be saved. The list of variables can be found in the TargetFolder option of the [General] section.

The default camera mode is the mode chosen when opening DAP-Imager.

The flash light duration is the number of milliseconds the flash light remains lit when pressing the <Flash Light> button.

3.2.4 Barcodes

DAP-Imager supports the following barcode symbologies:

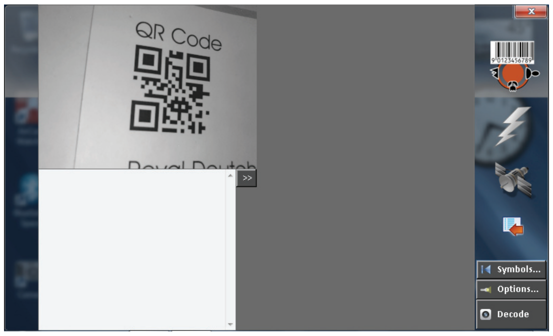
1D:	Code 11, Code 39 (+extended), Code 93, Interleaved 2 of 5, Codabar, Code 128, EAN13, EAN8, PatzchCode, UPC-A and UPC-E.
2D:	PDF417, DataMatrix, QR Code and MicroQR Code
Postal:	AustrialiaPost, IntelligentMail, Planet, Postnet and RM4SCC

3.0 Operating the Unit

DAP-Imager can be used as a standalone application, or used in conjunction with ScannerManager. In both cases, you will probably want to leave the application hidden to wedge barcodes.

3.2.4.1 How to Scan Barcodes

To scan barcodes, first ensure DAP-Imager is in barcode mode. To do that, open the application (double click the icon in the notification area) and select **Barcode** from the mode menu.



The main window shows a preview and a text box with the results scanned. At this point we can start scanning to test the capabilities.

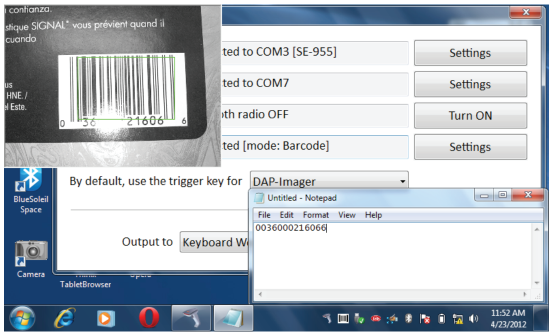
Steps:

- Press and hold the trigger.
- Move the unit so that it is almost perpendicular to the barcode. The barcode should fill 30% to 75% of the preview window, depending on the density of the barcode.
- If the image is not on focus, move unit to force an automatic focus.
- It usually takes around 1 second to decode a barcode. On a successful decode attempt, the barcode is surrounded by a green box and a single beep is heard. Two beeps indicate failure.

Close DAP-Imager with the top right **X** button; the program remains in background.

3.2.4.1.1 Using ScannerManager

Normally you will want to use ScannerManager, like in the following screen:



ted to COM3 [SE-955]	Settings
ted to COM7	Settings
oth radio OFF	Turn ON
ted [mode: Barcode]	Settings

By default, use the trigger key for DAP-Imager

Output to Keyboard W

Untitled - Notepad

```
0036000216066
```

ScannerManager configures DAP-Imager automatically and takes care of the wedging.

When DAP-Imager is in background, it shows a live preview as long as the trigger is held down. It helps ensuring the focus, position and distance are correct. As soon as the barcode can be decoded, ScannerManager receives the data and wedge it (if the output is set to Keyboard Wedge).

3.2.4.1.2 Using DAP-Imager as a Stand-Alone Application

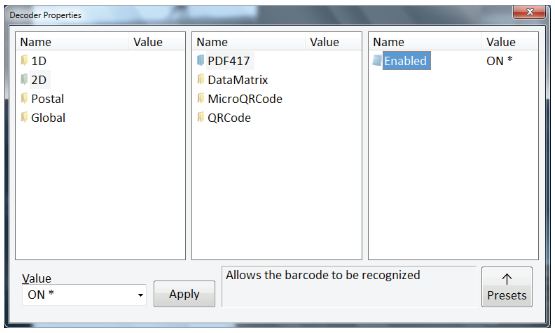
DAP-Imager can be used without ScannerManager. In that case, ScannerManager must not be running, so that the trigger will be used exclusively for DAP-Imager.

NOTE: If you have used ScannerManager before, the keyboard wedge will have been disabled. To enable it, turn "KbWedge" **ON** in the INI file. Check the ".INI Configuration File" section for more information.

You can scan barcodes the same way it's done in ScannerManager even if using DAP-Imager separately.

3.2.4.2 Decoder Configuration

The **Symbols** button shows the decoder properties configuration screen. It allows enabling or disabling specific barcode types and setting advanced parameters.



Decoder Properties

Name	Value
1D	
2D	
Postal	
Global	

Name	Value
PDF417	
DataMatrix	
MicroQRCode	
QRCode	

Name	Value
Enabled	ON *

Value: ON * — Apply

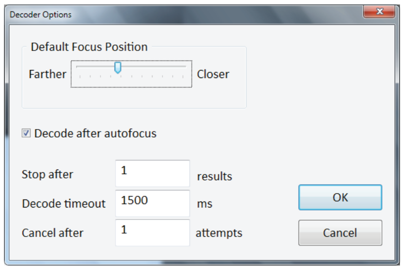
Allows the barcode to be recognized

Double click on an item to change its value. If it's an ON/OFF value, the change is applied immediately. For other value types, set the value in the "Value" field, and then click on **Apply**.

An asterisk (*) indicates default values.

Enabling all barcode types will make decoding slower. You may start the configuration by selecting a preset (**Presets** button).

The **Options** button pops up the Decoder Options screen.



Decoder Options

Default Focus Position: Farther — Closer

☑ Decode after autofocus

Stop after	1	results
Decode timeout	1500	ms
Cancel after	1	attempts

These parameters should not be changed, except if suggested by a DAP technical support representative.

3.2.5 .INI Configuration File

DAP-Imager uses a .INI configuration file located, by default, in

C:\ProgramData\DAP-Imager\DAP-Imager.ini

Before changing it, you must unload DAP-Imager by clicking its icon in the notification area and choosing **Quit**. Otherwise, the program rewrites the INI file when it quits.

If no DAP-Imager.ini file is found in the directory, a new one is automatically created with default values.

3.2.6 [General]

3.2.6.1 TargetFolder = %PICTURES%\%YEAR%-%MONTH%-%DAY%

To scan barcodes, first ensure DAP-Imager is in barcode mode. To do that, open the application (double click the icon in the notification area) and select "Barcode" from the mode menu.

Specifies the path where the picture is to be taken.

Supported variables:

Variables	
Name	**Description**
%YEAR%	4-digit year
%MONTH%	2-digit month
%DAY%	2-digit day of month
%DAYOFWEEK%	Name of the week day
%HOUR%	Hour
%MINUTE%	2-digit minute
%SECOND%	2-digit second
%INDEX%	Sequential index, incremented each time a picture is taken. The number is saved in "C:\ProgramData\DAP-Imager\NextImageIndex.txt".
%PICTURE%	Path of the default Windows folder to save pictures (C:\Users\username\Pictures)

3.2.6.2 FileNameTemplate = %HOUR%h%MINUTE%m%SECOND%s

Specifies the file name of images taken. Supports the same variables than TargetFolder.

3.2.6.3 DefaultImagerMode = Portrait

Name of the imager mode to be selected when DAP-Imager starts. Notice that if ScannerManager is present, it starts DAP-Imager in barcode mode.

Default supported values: Portrait, Landscape, Macro, Barcode

They correspond to the name of the corresponding section [ImagerMode:XXXX].

3.2.6.4 FlashLightDurationMs = 10000

Number of milliseconds the flash light remains lit when its button is clicked (any camera mode). The flash light is automatically turned off after a delay to save power.

3.2.6.5 Func1VirtualKey = 135

Virtual key code used for the main trigger key. For the integration with ScannerManager to work, you must use the default value (corresponds to the trigger on back of the unit).

For other virtual-key codes, refer to the "Virtual-Key Codes" section of the Windows Application UI Development guide. Notice that the codes here are in DECIMAL.

3.2.6.6 Func2VirtualKey = 117

Sets the key used to force an autofocus.

For other virtual-key codes, refer to the "Virtual-Key Codes" section of the Windows Application UI Development guide. Notice that the codes here are in DECIMAL.

3.2.6.7 Func1KeyModifiers = 0

Determines the key that must be pressed in combination with Func1VirtualKey.

3.2.6.8 Func2KeyModifiers = 0

Determines the key that must be pressed in combination with Func2VirtualKey. Use the key modifiers shown for Func1VirtualKey.

3.2.6.9 Func1KeySystemWide = 1

When set to 1, the main trigger key is registered as a global hotkey, so that DAP-Imager captures it even if another application has the focus.

3.2.6.10 Func2KeySystemWide = 0

When set to 1, the second trigger key is registered as a global hotkey, so that DAP-Imager captures it even if another application has the focus.

3.2.7 [Camera]

Options specific to the camera modes that take a picture.

3.2.7.1 InactiveTimeBeforeStandbyLevel1 = 10000

Number of milliseconds before the camera is stopped when the application is in background. Waking up the camera takes a few seconds. If you use the camera often, you may want to increase this value. Decrease it to save power.

3.2.7.2 ActivateDapImagerOnTrigger = OFF

When in a camera mode (portrait, landscape or macro), the trigger key is never global, except if this option is set. If set and Func1KeySystemWide is 1, pressing the trigger will show up DAP-Imager. Pressing another time takes a picture.

3.2.7.3 ShowImageNameOnPreview = OFF

When ON, the image file path is written on the image when the picture is taken.

3.2.8 [Barcodes]

Options specific to the barcode mode.

3.2.8.1 EnableAutoPreview = ON

When ON, DAP-Imager shows the camera preview in a top level window while the trigger key is pressed.

3.2.8.2 PreviewWndRect = 0 0 320 240

Size of the auto preview window. Should not be changed.

3.2.8.3 UIPolicy = Legacy

Sets the way the trigger key is handled. Only the "Legacy" UI policy is officially supported, but you may experiment with the other modes.

UIPolicy	
Name	**Description**
Legacy	Works like a regular handheld scanner: press and hold the trigger key to decode, release to cancel.
Standard	Press and release the trigger key to decode. DAP-Imager makes MaxNbrAttempts decoding attempts.
DecTrigUp	Attempts to decode when the trigger key is released.

3.2.8.4 DefaultFocus = 3733

Not used in this version of DAP-Imager.

3.2.8.5 Aimer = ON

Not used in this version of DAP-Imager.

3.2.8.6 DecodeAfterAutofocus = ON

Not used in this version of DAP-Imager.

3.2.8.7 MaxNbrResults = 1

When several barcodes are visible in an image, the decoder can return more than one result. Set this value to the maximum number of results that are considered. If 1, the first result is returned and the others are discarded.

3.2.8.8 DecodeTimeoutMs = 1500

Maximum duration of a decode operation, in milliseconds. If the decode operation takes longer, it is cancelled. Using a small timeout won't allow decoding most barcodes. Using a higher value may have an impact on the user interface responsiveness.

3.2.8.9 MaxNbrAttempts = 1

In the Standard or DecTrigUp UI policies, number of attempts DAP-Imager tries to decode before returning NO READ.

3.2.8.10 InactiveTimeBeforeStandbyLevel1 = 10000

Number of milliseconds before the flash light is turned off.

3.2.8.11 InactiveTimeBeforeStandbyLevel2 = 10000

Number of milliseconds before the camera is stopped when the trigger is not pressed. Waking up the camera takes a few seconds. If you scan barcodes often, you may want to increase this value. Decrease it to save power.

3.2.8.12 KbWedge = OFF

Set this option to ON if you don't use ScannerManager and want to wedge barcodes.

3.2.8.13 AddTab = OFF

When KbWedge is ON and a barcode is wedged, simulates a TAB key after the barcode data.

3.2.8.14 AddEnter = ON

When KbWedge is ON and a barcode is wedged, simulates a RETURN key after the barcode data.

3.2.8.15 Preamble =

When KbWedge is ON and a barcode is wedged, this value is prefixed to the barcode data.

3.2.8.16 Postamble =

When KbWedge is ON and a barcode is wedged, this value is appended to the barcode data.

3.2.8.17 InterCharDelay = 0

When KbWedge is ON and a barcode is wedged, sets the delay between each key that is simulated, in milliseconds.

3.2.8.18 MaxGainWithoutMVLight = 2500

Not used in this version of DAP-Imager.

3.2.8.19 MinGainWithMovieLight = 1000

Not used in this version of DAP-Imager.

3.2.8.20 MaxGain = 4000

Not used in this version of DAP-Imager.

3.2.8.21 GainStep = 200

Not used in this version of DAP-Imager.

3.2.8.22 IdealGain = 2000

Not used in this version of DAP-Imager.

3.2.8.23 FlashIntensityStep = 100

Not used in this version of DAP-Imager.

3.2.8.24 MaxFlashIntensity = 100

Not used in this version of DAP-Imager.

3.2.8.25 MaxShutter = 4000

Not used in this version of DAP-Imager.

3.2.8.26 ShutterStep = 260

Not used in this version of DAP-Imager.

3.2.8.27 IdealShutter = 575

Not used in this version of DAP-Imager.

3.2.8.28 AppendSymbology = OFF

When ON, DAP-Imager will wedge the name of the symbology after the barcode data. It can be used to find out the symbology of a barcode.

3.2.9 [OCR]

OCR not yet supported in this version of DAP-Imager.

3.2.10 [ImagerModes]

3.2.8.1 ModeList = Portrait,Landscape,Macro,Barcode,

Lists the modes that appear in the camera mode menu. The modes specified here must be defined in the sections that follow.

3.2.11 [ImagerMode:XXXX]

This sections defines a given imager mode (XXXX), as listed in ModeList. Not all values are used; it depends on the ModeType option.

3.2.11.1 ModeType = 0

A value of 0 indicates that it's a mode to take pictures.

A value of 1 is for barcode decoding.

3.2.11.2 IconID = 142

Resource ID of the icon to be shown in the mode menu. You can use a resource editor to view the icons that are available (for example: http://www.resedit.net/).

3.2.11.3 SelectionButtonImageFileName = res\button-mode-portrait80.png

File path of the image shown when the mode is selected (mode button).

3.2.11.4 Enabled = ON

ON if the mode is enabled. If disabled, it will be grayed out in the mode menu.

3.2.11.5 AutoFlash = ON

Not used in this version of DAP-Imager.

3.2.11.6 GpsReportTypes = 1

Set this option to 1 to enable geotagging, or 0 to disable it.

3.2.11.7 PreviewWidth = 640

Size of the image preview. Should not be changed.

3.2.11.8 PreviewHeight = 480

Size of the image preview. Should not be changed.

3.2.11.9 StillWidth = 1600

Width of the images captured (resolution).

3.2.11.10 StillHeight = 1200

Height of the images captured (resolution).

3.2.11.11 ColorSpace = 16

Not used, leave unchanged.

3.2.11.12 FrameRate = 30.000000

Not used, leave unchanged.

3.2.11.13 Shutter = 10000

Exposition duration, when AutoExposure is OFF. The value must be between 0 and 10000.

3.2.11.14 Brightness = 5000

Brightness level. It shifts pixel values so that the image is either lighter or darker. The value must be between 0 and 10000 (higher is lighter).

3.2.11.15 GlobalGain = 0

Not yet supported by the camera.

3.2.11.16 Exposure = 5000

Not yet supported by the camera.

3.2.11.17 FlipMode = 1

Not supported.

3.2.11.18 AutoExposure = ON

When ON, the camera finds the best exposition and gain for the current lighting conditions. Using OFF allows setting Shutter manually, but since the camera doesn't support the GlobalGain option, it should not be used.

3.2.11.19 LightingMode = 0

0: Flash (light pulse)

1: Continuous lighting (flash light)

3.2.11.20 LightingPower = 0

The lighting power must be 0 (turned off) or 100 (turned on).

3.2.11.21 Aimer = OFF

Not supported by the camera.

3.2.11.22 Compression = ON

Set to ON to preview in JPEG format, or OFF to preview in YUV format.

3.2.11.23 CompressionRatio = 13

Not supported by the camera.

3.2.11.24 FocusPosition = 500

Defines where to focus (0: infinite, 10000: closest position). Not used when Autofocus is ON.

3.2.11.25 Autofocus = ON

Enables or disables continuous autofocus. When the camera is moved, it automatically tries to autofocus.

3.2.11.26 WhiteBalancePreset = 0

Defines how colors are balanced.

Variables	
Name	**Description**
0	Automatic
1	Fluorescent
2	Incandescent
3	Sunny
4	Cloudy
5	Movie Light
6	Flash
7	Hybrid

3.2.11.27 ManualWhiteBalance = OFF

Set to ON to use WhiteBalanceKelvin. Otherwise, WhiteBalancePreset applies.

3.2.11.28 WhiteBalanceKelvin = 8267

White balance value; 0 is the coldest (bluish), 10000 is the warmest.

3.2.11.29 PreviewToWindow = ON

Set to ON to have DirectShow paint the preview (improves performances). Should not be ON in barcode mode.

3.2.12 [Permissions]

3.2.12.1 Option(More) = 3

Set this value to 0 to prevent a user from accessing the "more" menu after the geotagging icon. When clicked, it simply shows the About box.

A value of 3 grants all permissions.

3.3 Command-Line Options

DAP-Imager includes several commands to control it from an external application.

3.3.1 Syntax

DAP-Imager [configFilePath] [-b] [-q] [-NextImageFilePath FilePath] [-OneShotCapture] [-SelectMode \"modeName\"] [-SetAutoFlash state] [-WaitUntilWndClosed] [-SetTopMost]

Command-Line Arguments

Argument	Description
configFilePath	.ini file to load (optional; default path is C:\ProgramData\DAP-Imager\DAP-Imager.ini).
-b	Execute in background
-q	Quit any instance already running
-NextImageFilePath	Sets the path of the next image file saved (one-shot). Normally used with "-OneShot-Capture".
-OneShotCapture	Shows the preview, let the user press the trigger and take a picture. When taken, DAP-Imager hides. Can be used with "-NextImageFilePath" to allow a user taking a picture that is then retrieved by an external application.
-SelectMode	The next parameter is the name of the mode to select (Portrait, Landscape, Macro, Barcode)
-ResetOptions:	Ignores DAP-Imager.ini and use the default settings
-WaitUntilWndClosed	Shows DAP-Imager and do not return before it's hidden

4.0 Programming the Unit

4.1 Bar Code Parameter Menus

This chapter describes the programmable parameters, provides bar codes for programming, and hexadecimal equivalents for host parameter programming through SSI.

Operational Parameters

The SE-955 is shipped with the factory default settings shown in Table 8-1 on page 8-5. These factory default values are stored in non-volatile memory and are preserved even when the scanner is powered down. Changes to the factory default values can be stored as custom defaults. These values are also stored in non-volatile memory and are preserved even when the scanner is powered down.

To change the parameter values:

- Scan the appropriate bar codes included in this chapter. The new values replace the existing memory values. To set the new values as custom defaults, scan the Write to Custom Defaults bar code. The factory default or custom default parameter values can be recalled by scanning the SET FACTOR DEFAULT bar code or the RESTORE DEFAULTS bar code on page 8-10.

– or –

- Send the parameter through the scan engine's serial port using the SSI command PARAM_SEND. Hexadecimal parameter numbers are shown in this chapter below the parameter title, and options appear in parenthesis beneath the accompanying bar codes. Instructions for changing parameters using this method are found in Chapter 9, Simple Serial Interface.

The table below lists the factory defaults for all parameters. To change any option, scan the appropriate bar code(s).

Parameter	Parameter Number (Hex)	Factory Default	Section Number
Set Factory Default		All Defaults	4.2.1
Beeper Volume	0x8C	Medium	4.2.2
Beeper Tone	0x91	Medium Frequency	4.2.3
Beeper Frequency Adjustment	0xF0 0x91	2500 Hz	4.2.4
Laser On Time	0x88	3.0 sec	4.2.5
Aim Duration	0xED	0.0 sec	4.2.6
Scan Angle	0xBF	Medium (46°)	4.2.7
Power Mode	0x80	Low Power	4.2.8
Trigger Mode	0x8A	Level	4.2.9
Time-out Between Same Symbol	0x89	1.0 sec	4.2.10
Beep After Good Decode	0x38	Enable	4.2.11
Transmit "No Read" Message	0x5E	Disable	4.2.12
Parameter Scanning	0xEC	Enable	4.2.13
Linear Code Type Security Levels	0x4E	1	4.2.14
Bi-directional Redundancy	0x43	Disable	4.2.15
UPC/EAN			**5.1**
UPC-A	0x01	Enable	5.1.1
UPC-E	0x02	Enable	5.1.2
UPC-E1	0x0C	Disable	5.1.3
EAN-8	0x04	Enable	5.1.4
EAN-13	0x03	Enable	5.1.5
Bookland EAN	0x53	Disable	5.1.6
Decode UPC/EAN Supplementals	0x10	Ignore	5.1.7
Decode UPC/EAN Supplemental Redundancy	0x50 7	8-25	5.1.8
Transmit UPC-A Check Digit	0x28	Enable	5.1.9
Transmit UPC-E Check Digit	0x29	Enable	5.1.10
Transmit UPC-E1 Check Digit	0x2A	Enable	5.1.11
UPC-A Preamble	0x22	System Character	5.1.12
UPC-E Preamble	0x23	System Character	5.1.13
UPC-E1 Preamble	0x24	System Character	5.1.14

Parameter	Parameter No. (Hex)	Factory Default	Page Number
Convert UPC-E to A	0x25	Disable	5.1.15
Convert UPC-E1 to A	0x26	Disable	5.1.16
EAN-8 Zero Extend	0x27	Disable	5.1.17
Convert EAN-8 to EAN-13 Type	0xE0	Type is EAN-13	5.1.18
UPC/EAN Security Level	0x4D	0	5.1.19
UCC Coupon Extended Code	0x55	Disable	5.1.20
Code 128			**5.2**
Code-128	0x08	Enable	5.2.1
UCC/EAN-128	0x0E	Enable	5.2.2
ISBT 128	0x54	Enable	5.2.3
Code 39			**5.3**
Code 39	0x00	Enable	5.3.1
Trioptic Code 39	0x0D	Disable	5.3.2
Convert Code 39 to Code 32	0x56	Disable	5.3.3
Code 32 Prefix	0xE7	Disable	5.3.4
Set Length(s) for Code 39	0x12 0x13	2-55	5.3.5
Code 39 Check Digit Verification	0x30	Disable	5.3.6
Transmit Code 39 Check Digit	0x2B	Disable	5.3.7
Code 39 Full ASCII Conversion	0x11	Disable	5.3.8
Code 93			**5.4**
Code 93	0x09	Disable	5.4.1
Set Length(s) for Code 93	0x1A 0x1B	4-55	5.4.2
Code 11			**5.5**
Code 11	0x0A	Disable	5.5.1
Set Lengths for Code 11	0x1C 0x1D	4 to 55	5.5.2
Code 11 Check Digit Verification	0x34	Disable	5.5.3
Transmit Code 11 Check Digit(s)	0x2F	Disable	5.5.4
Interleaved 2 of 5			**5.6**
Interleaved 2 of 5	0x06	Enable	5.6.1
Set Length(s) for I 2 of 5	0x16 0x17	14	5.6.2
Interleaved 2 of 5 Check Digit Verification	0x31	Disable	5.6.3
Transmit Interleaved 2 of 5 Check Digit	0x2C	Disable	5.6.4
Convert Interleaved 2 of 5 to EAN 13	0x52	Disable	5.6.5
Discrete 2 of 5			**5.7**
Discrete 2 of 5	0x05	Disable	5.7.1
Set Length(s) for Discrete 2 of 5	0x14 0x15	12	5.7.2
Chinese 2 of 5			**5.8**
Chinese 2 of 5	0xF0 0x98	Disable	5.8.1

Parameter	Parameter No. (Hex)	Factory Default	Page Number
Codabar			**5.9**
Codabar	0x07	Disable	5.9.1
Set Lengths for Codabar	0x18 0x19	5-55	5.9.2
CLSI Editing	0x36	Disable	5.9.3
NOTIS Editing	0x37	Disable	5.9.4
MSI			**5.10**
MSI	0x0B	Disable	5.10.1
Set Length(s) for MSI	0x1E 0x1F	6-55	5.10.2
MSI Check Digits	0x32	One	5.10.3
Transmit MSI Check Digit	0x2E	Disable	5.10.4
MSI Check Digit Algorithm	0x33	Mod 10/Mod 10	5.10.5
RSS			**5.11**
RSS-14	0xF0 0x52	Disable	5.11.1
RSS-Limited	0xF0 0x53	Disable	5.11.2
RSS-Expanded	0xF0 0x54	Disable	5.11.3
Data Options			**5.12**
Transmit Code ID Character	0x2D	None	5.12.1
Prefix/Suffix Values Prefix Suffix 1 Suffix 2	 0x69 0x68 0x6A	 NULL LF CR	5.12.2
Scan Data Transmission Format	0xEB	Data as is	5.12.3
Serial Interface			**5.13**
Baud Rate	0x9C	9600	5.13.1
Parity	0x9E	None	5.13.2
Software Handshaking	0x9F	Enable	5.13.3
Decode Data Packet Format	0xEE	Unpacketed	5.13.4
Host Serial Response Time-out	0x9B	2 sec	5.13.5
Stop Bit Select	0x9D	1	5.13.6
Intercharacter Delay	0x6E	0	5.13.7
Host Character Time-out	0xEF	200 msec	5.13.8
Event Reporting*			**5.14**
Decode Event 0xF0	0x00	Disable	5.14.1
Boot Up Event 0xF0	0x02	Disable	5.14.2
Parameter Event 0xF0	0x03	Disable	5.14.3
Numeric Bar Codes			**5.15**
Cancel			5.15.1
*See Table 9-9 on page 9-20 for formatting of any parameter whose number is 0x100 or greater.			

4.2 Bar Code Settings

4.2.1 Set Default Parameter

The SE-955 can be reset to two types of defaults: factory defaults or custom defaults. Scan the appropriate bar code below to reset the SE-955 to its default settings and/or set the scanner's current settings as the custom default.

- **Restore Defaults** - Scan this bar code to reset all default parameters as follows.
  - If custom defaults were set by scanning **Write to Custom Defaults**, scan **Restore Defaults** to retrieve and restore the scanner's custom default settings.
  - If no custom defaults were set, scan **Restore Defaults** to restore the factory default values.

Restore Defaults

- **Set Factory Defaults** - Scan this bar code to restore the factory default values. If custom defaults were set, they are eliminated.

Set Factory Defaults

- **Write to Custom Defaults** - Scan this bar code to store the current scanner settings as custom defaults. Once custom default settings are stored, they can be recovered at any time by scanning **Restore Defaults**.

Write to Custom Defaults

4.2.2 Beeper Volume

Parameter # 0x8C

To select a decode beep volume, scan the appropriate bar code.

Low (0x02)

***Medium (0x01)**

High (0x00)

4.2.3 Beeper Tone

Parameter # 0x91

To select a decode beep frequency (tone), scan the appropriate bar code.

Low Frequency (0x02)

***Medium Frequency (0x01)**

High Frequency (0x00)

4.2.4 Beeper Frequency Adjustment

Parameter # 0xF0 0x91

This parameter adjusts the frequency of the high beeper tone from the nominal 2500 Hz to another frequency matching the resonances of the installation. It is programmable in 10 Hz increments from 1220 Hz to 3770 Hz.

To increase the frequency, scan the bar code below, then scan three numeric bar codes in **Section 5.5** on page **95** that correspond to the desired frequency adjustment divided by 10. For example, to set the frequency to 3000 Hz (an increase of 500 Hz), scan numeric bar codes 0, 5, 0, corresponding to 50, or (500/10).

To decrease the frequency, scan the bar code below, then scan three numeric bar codes in **Section 5.5** on page **95** that correspond to the value (256 - desired adjustment/10). For example, to set the frequency to 2000 Hz (a decrease of 500 Hz), scan numeric bar codes 2, 0, 6, corresponding to 206, or (256 - 500/10).

To change the selection or cancel an incorrect entry, scan the Cancel bar code in **Section 5.5.1** on page **95**.

Beeper Frequency Adjustment
(Default: 2500 Hz)

4.2.5 Laser On Time

Parameter # 0x88

This parameter sets the maximum time decode processing continues during a scan attempt. It is programmable in 0.1 second increments from 0.5 to 9.9 seconds.

To set a Laser On Time, scan the bar code below. Next scan two numeric bar codes in **Section 5.5** on page **95** that correspond to the desired on time. Single digit numbers must have a leading zero. For example, to set an on time of 0.5 seconds, scan the bar code below, then scan the "0" and "5" bar codes. To change the selection or cancel an incorrect entry, scan the Cancel bar code in **Section 5.5.1** on page **95**.

Laser On Time
(Default: 3.0 sec.)

4.2.6 Aim Duration

Parameter # 0xED

When a scanner with an aim mode (see Table 9-10 on page 9-22) is triggered either by a trigger pull, or a START_DECODE command, this parameter sets the duration the aiming pattern is seen before a a scan attempt begins. It does not apply to the aim signal or the AIM_ON command. It is programmable in 0.1 second increments from 0.0 to 9.9 seconds. No aim pattern is visible when the value is 0.0.

To set an aim duration, scan the bar code below. Next scan two numeric bar codes beginning on page 8-71 that correspond to the desired aim duration. Single digit numbers must have a leading zero. For example, to set an aim duration of 0.5 seconds, scan the bar code below, then scan the "0" and "5" bar codes. To change the selection or cancel an incorrect entry, scan the Cancel bar code in **Section 5.5** on page **95**.

Aim Duration
(Default: 0.0 sec.)

4.2.7 Scan Angle

Parameter # 0xBF

This parameter sets the scan angle to narrow, medium or wide.

Narrow Angle (35°)
(0x05)

***Medium Angle (46°)**
(0x06)

Wide Angle (53°)
(0x07)

4.2.8 Power Mode

Parameter # 0x80

This parameter determines the power mode of the engine.

In Low Power mode, the scanner enters into a low power consumption Sleep power state whenever possible (provided all WAKEUP commands have been released).

In Continuous Power mode, the scan engine remains in the Awake state after each decode attempt.

The Sleep and Awake commands can be used to change the power state in either the Low Power mode or the Continuous Power mode.

Continuous Power (0x00)

Low Power (0x01)

4.2.9 Triggering Modes

Parameter # 0x8A

Choose one of the options below to trigger the scan engine. Bar codes and option numbers are on the following page.

- **Scan (Level)** - A trigger pull activates the laser and decode processing. The laser remains on and decode processing continues until a trigger release, a valid decode, or the Laser On Time-out is reached.

***Level (0X00)**

- **Scan (Pulse)** - A trigger pull activates the laser and decode processing. The laser remains on and decode processing continues until a valid decode or the Laser On Time-out is reached.

Pulse (0X02)

- **Continuous** - The laser is always on and decoding.

Continuous (0X04)

- **Blink** - This trigger mode is used for triggerless operation. Scanning range is reduced in this mode. This mode cannot be used with scanners that support an aim mode.

Blinking (0X07)

- **Host** - A host command issues the triggering signal. The scan engine interprets an actual trigger pull as a Level triggering option.

Host (0X08)

4.2.10 Time-out Between Same Symbol

Parameter # 0x89

When in Continuous triggering mode, this parameter sets the minimum time that must elapse before the scanner decodes a second bar code identical to one just decoded. This reduces the risk of accidently scanning the same symbol twice. It is programmable in 0.1 second increments from 0.0 to 9.9 seconds.

To set a time-out between same symbol, scan the bar code below. Next scan two numeric bar codes beginning on page 8-71 that correspond to the desired time-out. Single digit values must have a leading zero. For example, to set a time-out of 0.5 seconds, scan the bar code below, then scan the "0" and "5" bar codes. To change the selection or cancel an incorrect entry, scan the Cancel bar code in **Section 5.5.1** on page **95**.

Time-out Between Same Symbol
(Default: 1.0 sec.)

4.2.11 Beep After Good Decode

Parameter # 0x38

Scan this symbol to set the scanner to beep after a good decode.

***Beep After Good Decode**
(0x01)

Scan this symbol to set the scanner not to beep after a good decode. The beeper still operates during parameter menu scanning and indicates error conditions.

Do Not Beep After Good Decode
(0x00)

4.2.12 Transmit "No Read" Message

Parameter # 0x5E

Enable this option to transmit "NR" if a symbol does not decode during the timeout period or before the trigger is released. Any enabled prefix or suffixes are appended around this message.

Enable No Read
(0x01)

When disabled, and a symbol cannot be decoded, no message is sent to the host.

***Disable No Read**
(0x00)

4.2.13 Parameter Scanning

Parameter # 0xEC

To disable decoding of parameter bar codes, scan the bar code below. The Set Defaults parameter bar code can still be decoded. To enable decoding of parameter bar codes, either scan *Enable Parameter Scanning (0x01), Set Factory Defaults or set this parameter to 0x01 via a serial command.

***Enable Parameter Scanning (0x01)**

Disable Parameter Scanning (0x00)

4.2.14 Linear Code Type Security Level

Parameter # 0x4E

The SE-955 offers four levels of decode security for linear code types (e.g. Code 39, Interleaved 2 of 5). Select higher security levels for decreasing levels of bar code quality. As security levels increase, the scanner's aggressiveness decreases. Select the security level appropriate for your bar code quality.

Linear Security Level 1

The following code types must be successfully read twice before being decoded:

Code Type	Length
Codabar	All
MSI	4 or less
D 2 of 5	8 or less
I 2 of 5	8 or less

***Linear Security Level 1 (0x01)**

Linear Security Level 2

All code types must be successfully read twice before being decoded.

Linear Security Level 2 (0x02)

Linear Security Level 3

Code types other than the following must be successfully read twice before being decoded. The following codes must be read three times:

Code Type	Length
MSI	4 or less
D 2 of 5	8 or less
I 2 of 5	8 or less

Linear Security Level 3 (0x03)

Linear Security Level 4

All code types must be successfully read three times before being decoded.

Linear Security Level 4 (0x04)

4.2.15 Bi-directional Redundancy

Parameter # 0x43

Enable this option to transmit "NR" if a symbol does not decode during the timeout period or before the trigger is released. Any enabled prefix or suffixes are appended around this message.

Enable Bi-directional Redundancy (0x01)

When disabled, and a symbol cannot be decoded, no message is sent to the host.

***Disable Bi-directional Redundancy (0x00)**

5.0 UPC Types

5.1 UPC / EAN

5.1.1 Enable/Disable UPC-A : Parameter # 0x01

To enable or disable UPC-A, scan the appropriate bar code below.

***Enable UPC-A**
(0x01)

Disable UPC-A
(0x00)

5.1.2 Enable/Disable UPC-E : Parameter # 0x02

To enable or disable UPC-E, scan the appropriate bar code below.

***Enable UPC-E**
(0x01)

Disable UPC-E
(0x00)

5.1.3 Enable/Disable UPC-E1 : Parameter # 0x0C

To enable or disable UPC-E1, scan the appropriate bar code below.

Enable UPC-E1
(0x01)

***Disable UPC-E1**
(0x00)

Note

UPC-E1 is not a UCC (Uniform Code Council) approved symbology.

5.1.4 Enable/Disable EAN-8 : Parameter # 0x04

To enable or disable EAN-8, scan the appropriate bar code below.

***Enable EAN-8**
(0x01)

Disable EAN-8
(0x00)

5.1.5 Enable/Disable EAN-13 : Parameter # 0x03

To enable or disable EAN-13, scan the appropriate bar code below.

***Enable EAN-13**
(0x01)

Disable EAN-13
(0x00)

Note

UPC-E1 is not a UCC (Uniform Code Council) approved symbology.

5.1.6 Enable/Disable Bookland EAN : Parameter # 0x53

To enable or disable EAN Bookland, scan the appropriate bar code below.

Enable Bookland EAN
(0x01)

***Disable Bookland EAN**
(0x00)