Scripting Language and Scripting Engine

Programming Guide

Document Number 013-RD000-000-12- 201010

Offices: **Ocean Optics, Inc.**
830 Douglas Ave., Dunedin, FL, USA 34698
Phone 727.733.2447
Fax 727.733.3962
8 a.m.– 8 p.m. (Mon-Thu), 8 a.m.– 6 p.m. (Fri) EST

E-mail: **Info@OceanOptics.com** **(General sales inquiries)**
Orders@OceanOptics.com **(Questions about orders)**
TechSupport@OceanOptics.com **(Technical support)**

A HALMA GROUP COMPANY

Additional Offices:

Ocean Optics Asia

666 Gubei Road, Kirin Tower, Suite 601B, Changning District, Shanghai, 200336 PRC

Phone **86.21.6295.6600**
Fax **86.21.6295.6708**
E-Mail **Sun.Ling@OceanOptics.com**

Ocean Optics EMEA (Europe, Middle East & Africa)

Geograaf 24, 6921 EW DUIVEN, The Netherlands

Phone **31-(0)26-3190500**
Fax **31-(0)26-3190505**
E-Mail **Info@OceanOptics.eu**

Regional Headquarters
Maybachstrasse 11
73760 Ostfildern
Phone 49-711 34 16 96-0
Fax 49-711 34 16 96-85
E-Mail Sales@Mikropack.de

Table of Contents

Chapter 6: Functions Reference 17

About This Manual

Document Purpose and Intended Audience

This document provides you with instructions to install and use the Jaz Scripting Engine.

Document Summary

Chapter	Description
Chapter 1: *Introduction*	Provides an overview of the Jaz Scripting Language (JSL) software.
Chapter 2: *Jaz Scripting Engine Architecture*	Describes the structure of JSL.
Chapter 3: *Installation*	Provides installation instructions for JSL on a Windows or Linux operating system.
Chapter 4: *Windows Scriptor Launcher*	Contains instructions for using the Scriptor Launcher, a graphical interface for Ocean Optics' Scriptor Scripting for the Jaz system when used with Windows.
Chapter 5: *Using the Jaz Scripting Engine*	Describes how to use the Jaz Scripting Engine.
Chapter 6: *Functions Reference*	Contains an alphabetical list of JSL functions, including the syntax, arguments and intent for each.
Appendix A: *Example Scripts*	Provides several example scripts for reference.

Product-Related Documentation

- ***Jaz Installation and Operation Manual***

You can access documentation for Ocean Optics products by visiting our website at **http://www.oceanoptics.com**. Select *Technical → Operating Instructions*, then choose the appropriate document from the available drop-down lists. Or, use the **Search by Model Number** field at the bottom of the web page.

You can also access operating instructions for Ocean Optics products on the *Software and Technical Resources* CD included with the system.

Engineering-level documentation is located on our website at *Technical → Engineering Docs*.

Chapter 1

Introduction

Overview

Jaz is a community of smart sensing instruments that consists of a high-performance miniature spectrometer that accommodates USB and Ethernet connectivity (for PC-free performance), battery operation, multi-spectrometer channel capability, and a selection of light sources. It's the next generation spectrometer brought to you by Ocean Optics, the company that invented the world's first miniature fiber optic spectrometer and has delivered over 120,000 instruments for a wide range of applications since 1992.

The uniqueness of Jaz lies in its adaptability. You can easily mix and match Jaz modules to optimize setups for thousands of absorbance, reflectance and emission applications. Jaz comes with its own basic software, and SpectraSuite Spectroscopy Operating Software is also available for post-acquisition processing of spectral data. Jaz Scripting Language (JSL) software takes it one step further by offering a Jaz scripting engine, a powerful tool simple enough for nonprogrammers to create their own applications for Jaz. JSL allows you to build a sequence of steps for performing and analyzing spectral data for your unique application.

Goals of the Jaz Scripting Language

The Jaz Scripting Language (JSL) has been designed to provide a sequence of actions to be performed on either the Jaz hardware or a host computer connected to Jaz via Ethernet. The language is intended to provide the primitive functions that would be useful in performing automated tasks. Its syntax is skewed towards the linear model of BASIC with some enhancements.

The overall goals of the language are as follows:

- Simple non-programmer oriented syntax -- case insensitive and one directive per line syntax
- Primitives that map well to the tasks which may be performed on the target hardware
- Provision for user-defined routines to encapsulate repetitive tasks
- Integration into the Jaz matrix of daemon processes with minimal exposure of script authors to the underlying complexities of the Jaz architecture
- Simple algebraic expressions for basic computation and script flow logic
- Interpreted runtime with an engine that is multi-platform and may serve as an infrastructure component for graphical user script design and debugging.
- Strongly typed language with type coercion made by the script engine as appropriate

Operating System Support

- Microsoft Windows – Windows 7, 2000, XP; 32-bit and 64-bit
- Linux – Red Hat 9 or later, Fedora (any version), Debian 3.1 or later (Sarge), SUSE (9.0 or later), Centos (any version), and Ubuntu

Chapter 2

Jaz Scripting Engine Architecture

Overview

The Jaz Scripting engine comprises the linguistic analyzer for the language as well as the runtime driver for script execution. The engine is invoked in the standard UNIX command line format with command flags and arguments which determine modes of actions and the source of input.

Jaz Scripts

A script is an ASCII text file which can be composed by whatever means a user wishes to employ. Care has been taken to filter out extraneous characters used as formatting by some editors. Plain 7-bit ASCII text is the expected input.

Script General Layout

A script has the following sections:

- *Constants (optional)*
- *Variable Declaration*
- *Main Program*
- *User-defined Procedures* (optional)

Constants (optional)

Constants are values the script writer wishes to reference as if they were built-in elements. A Constants section is not mandatory. If defined, the script engine defines the constant's type via a strict parse of the value. Constants may not be changed once defined. The script engine will flag all attempts to redefine a constant at compilation time.

Variable Declaration

Variables must be declared before use in the program or in user-defined procedures. A variable has a name, a type, and possibly a modifier (depending on the type). The script engine checks the type of each variable for the appropriateness of its type in an expression. Type mismatches are flagged by the parser at compile time as errors. There must be a Variable Declaration section as the script would be useless without variables to operate on.

All variables are global in scope. That means that all variables to be used in the entire script must be declared in the Variable Declaration section. All variables are visible from both the main program and from user-defined procedures.

A variable is declared as `<name> <type> <optional count>`

Where:

`type` = one of the data types in the following **Data Types** table

`optional count` = the way an array is declared. See the example below.

Example:

LampIntensity INT_16 declares a single integer while LampIntensity INT_16 40 declares LampIntensity to be an array of 40 integers. Array indexing is inclusive [0...N-1] for a declaration of VariableName Type N.

Scripting language is case-insensitive so that VariableOne, variableOne, and VARIABLEONE all reference the same variable. Therefore, it is illegal to have a declaration section like the following:

```
VariableOne INT_16
VARIABLEONE INT_16 … error re-declaration or duplicate.
```

However, you can use any capitalization style you wish in your script since any sequence resolves to the same variable.

Data Types

Type Name	Contents	Notes
INT_8	8-bit integer	
INT_16	16-bit integer	This is the preferred integer form
INT_32	32-bit integer	
TEXT	Null-terminated character string	
REAL	Single-precision floating point	
SPECTRAL	Represents a spectrum reading	
FILE	File of spectral data	ASC = ascii CSV = comma separated RAW = tab delimited
TABLE	2-dimensional array of real numbers	Meant to be used in computation with sample spectrum

TEXT Literals

A literal of type text may need to have carriage returns inserted within them to make screen text readable. For example, to have the string `Test1Test2Test3` show up on separate lines, insert the '$' character wherever a new line is desired. Internally, the scripting engine inserts the carriage returns as in `Test1$Test2$Test3`.

Arithmetic

The scripting engine supports simple algebraic expressions including parenthesis nesting, unary minus (negation), and +,-,/,*. It does not support nested functions in expressions since scripting functions do not return a value. This is in keeping with the BASIC language paradigm.

Assignment: `MyIntegerVariable := (2 * (MyPixel[J] - BIAS/100))`, where **:=** is the assignment operator.

Evaluation: `IF(MyVariable1 = MyVariable2) GOTO RESET`, where **=** is the boolean comparison operator. Please note that **:=** and **=** are not interchangeable.

Operators

Symbol	Name	Type
+	Addition	Arithmetic
-	Subtraction	Arithmetic
/	Division	Arithmetic
*	Multiplication	Arithmetic
()	Parenthesis nesting	Arithmetic
-()	Negation of a quantity	Arithmetic
[]	Array Element	Indexing
=	Equals	Boolean
<>	Not Equals	Boolean
>=	Greater than or equal	Boolean
<=	Less than or equal	Boolean

Symbol	Name	Type
>	Greater than	Boolean
<	Less than	Boolean
:=	Assignment	Arithmetic

Main Program

The Main Program section typically defines the main logic flow of the script. The Main Program begins with the keyword `START` and is terminated with the keyword `END`. The keyword `STOP` marks the end of the script and is logically paired with the keyword SCRIPT. Labels can be placed before a statement to make that statement a destination for a GoTo statement.

User-defined Procedures

You can define routines to carry out repetitive tasks or other repeatedly used tasks. User-defined procedures are declared after the body of the main program (i.e., after the END of the Main Program section). The syntax is:

`[Process ProcessName]` and is terminated with the keyword END.

To invoke a user-defined procedure, use the following:

```
CALL ProcessName
```

A procedure shares the same variables as the main section but it may have its own labels to avoid confusion. A procedure may not execute a GoTo to a label outside itself. For example, if the Main Program section has a label ‘A’ and the procedure has a GoTo A then the procedure must also have a label ‘A’, or it is an error.

Flow Control

Control flow is achieved with GOTO and CALL statements. All loop iteration structures may be realized with judicious use of labels and GOTO statements. For example a FOR loop might be coded in the following way:

```
I := 0
Label FORLOOP
if(I >= N) GOTO ENDFOR
(your calculations here)
I := I + 1
GOTO FORLOOP
Label ENDFOR
(rest of program)
```

The DO ... DONE construct is a special iteration construct. It behaves exactly like a FOR loop, but with one special caveat: a DO ... DONE loop cannot have GOTO statements in it. CALL statements are allowed in a DO … DONE loop.

Built-In Variables

The intent of built-in variables is to make system-wide setting values available to the developer. For example, INTEGRATION_TIME_SECONDS establishes an integration time of 1 second by default. To change this, assign it a new value prior to taking a spectrum reading. Those that are marked as constants are not user-modifiable and are meant to convey values to system functions. For example, in specifying lamp intensity the value ALLBULBS means "set all bulbs in the light source to the value" while BULB1 specifies the first bulb in the source.

Built-in Variables for Use

NOTE: Variables in red are important settings for the spectrometer and should be set by the developer to alter the default behavior.

Variable	Type	Constant	Typical Use/Value
NONE	TEXT	YES	"NONE"
LED_LAMP_TYPE	INT_16	YES	6
UV_VIS_LAMP_TYPE	INT_16	YES	4
VIS_NIR_LAMP_TYPE	INT_16	YES	5
AVERAGES	INT_16		0
BOX_CAR	INT_16		0
LAMBDA_LOW	INT_16		0
LAMBDA_HIGH	INT_16		0
LIGHT	INT_16		0
RESPONSE_TIMEOUT_SECONDS	INT_16		5
NUMBER_OF_PIXELS	INT_16		2048
NON_LINEARITY	INT_16		0

Variable	Type	Constant	Typical Use/Value
ELECTRIC_DARK	INT_16		0
STRAY_LIGHT	INT_16		0
DISPLAY_BRIGHTNESS	INT_16		0
DISPLAY_WIDTH	INT_16		8
DISPLAY_PRECISION	INT_16		3
INTEGRATION_TIME_SECONDS	INT_32		1
INTEGRATION_TIME_NANOSEC	INT_32		0
STROBE_DELAY_USEC	INT_16		0
LAMP_CONTROL	INT_16		0
SPECTROMETER_CHANNEL_NUMBER	INT_16		0
PROCESSING_FLAG	INT_16	YES	2
STROBE_ENABLE	INT_16		1
SPECTROMETER	TEXT	YES	"SPECTROMETER"
LAMP	TEXT	YES	"LAMP"
BULB1	INT_16	YES	1
BULB2	INT_16	YES	2
ALLBULBS	INT_16	YES	3

Chapter 3

Installation

Overview

The JSL can be run under Jaz, Linux or Windows and installed from either the JSL CD or from the Ocean Optics **Software Downloads** page. See *Operating System Support* to determine if your computer operating system is supported by JSL.

Retrieving from a CD

Your JSL software is shipped to you from Ocean Optics on a CD. You will need the password located on the jacket of the CD containing your JSL software to complete the installation.

► *Procedure*

1. Insert the CD that you received containing your JSL software into your computer.
2. Browse to the appropriate JSL set-up file for your computer and double-click it to start the software installation. Set-up files are as follows:
   - Windows: Scriptor-1.0-windows-installer.exe
   - Linux: Scriptor-1.0-Linux-installer.bin
3. Save the software to the desired location. Your JSL password is located on the CD jacket.
   - Windows: The default installation directory is **C:\Program Files\Ocean Optics\JSL**. The installer wizard guides you through the installation process. The JSL icon location is **Start | Programs | Ocean Optics | JSL | Scriptor Launcher** and the current user's desktop.
   - Linux: The default installation directory is your home directory/Ocean Optics.

Downloading from the Ocean Optics Website

Installing on a Windows Platform

Total download is approximately 3.4 MB.

► *Procedure*

1. Close all other applications running on the computer.
2. Start Internet Explorer.
2. Browse to the Software Downloads page on the Ocean Optics website at http://www.oceanoptics.com/technical/softwaredownloads.asp.
3. Click on the JSL software for your Windows operating system.
4. Save the software to the desired location. The default installation directory is **C:\Program Files\Ocean Optics\JSL**. The installer wizard guides you through the installation process. Your JSL password is located on the jacket of the CD containing the JSL software. The JSL icon location is **Start | Programs | Ocean Optics | JSL | Scriptor Launcher** and your desktop.

Installing on a Linux Platform

Total download is approximately 3.9 MB.

► *Procedure*

1. Download the Linux installer from the Software Downloads page on the Ocean Optics website at http://www.oceanoptics.com/technical/softwaredownloads.asp.
2. Follow the wizard prompts. Your JSL password is located on the jacket of the CD containing the JSL software. The default installation directory is your home directory/Ocean Optics.

Transferring JSL to Jaz

To run JSL scripts on the Jaz directly (without the use of a computer) it is necessary to transfer the JSL application (Scriptor) and your desired scripts to a specially organized SD card. To simplify this process, Ocean Optics provides a ZIP file that contains the files and directories necessary to use the JSL directly on the Jaz. This ZIP file is password-protected using the same password as the installers. This ZIP file should be extracted and its contents copied onto the SD card such that the boot directory in the ZIP file is copied to the top (root) directory of the SD card. The Jaz will not recognize the JSL if the ZIP file is copied directly to the SD card without first extracting it. Note that multiple SD cards can be set up in this way and a freshly formatted SD card can be reinitialized for use with the JSL by following this procedure. The ZIP file may contain sample scripts to illustrate where they should be placed and how they may be named; these should be replaced with the desired scripts.

See your Jaz Installation and Operation Manual for instructions on using an SD card with Jaz (see *Product-Related Documentation*).

Chapter 4

Windows Scriptor Launcher

Introduction

Scriptor Launcher provides a graphical interface for Ocean Optics' Jaz Scripting Language Engine (JSL) when used with Windows. The scripting engine is the program that parses and executes user scripts for Jaz. By using the launcher, you need not interact with the system at a command line level. The launcher provides the means for observing the results from a tethered Jaz program by either viewing the console output or capturing the same data in a 'capture' file. A capture file is a plain text file which holds all the information generated by the scripting engine during a run. You may save this file for future reference.

Configuration

Installation of the Scriptor Launcher merely requires a copy of the file ScriptorLauncher.exe in the location of your choice on your local file system. It is suggested that you store the application in the default directory named C:\Program Files\Ocean Optics\JSL. You may wish to create a desktop shortcut to make application launch more convenient.

Prerequisites

The script launcher runs on Windows 7, XP, and Vista. The launcher should be installed in the same directory as the scripting engine, although this is not essential.

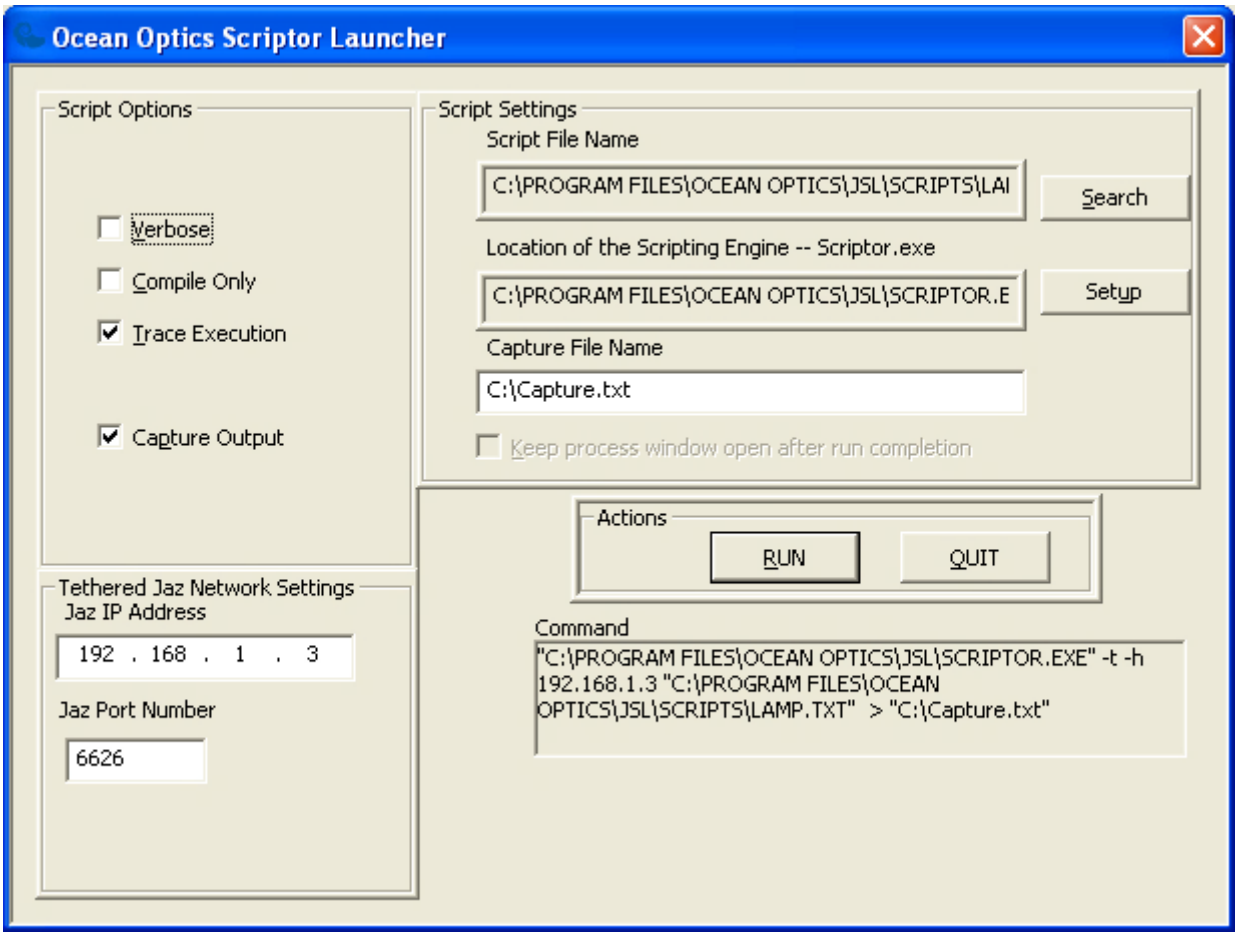
System requirements include the following:

- Computer running Windows 7, XP or Vista
- The Scriptor.exe scripting engine installed
- The scriptor launcher application installed
- To view captures from the application, WordPad.exe must be on your path. WordPad is used to view the output of your scripting section. See your Microsoft Windows manual for path setting tasks.

Running the Application

Scriptor Launcher Main Window

Double-click the launcher icon to display the application window.



The fields are populated with the settings last used so that you need only change the parameters necessary for the current operation.

The application window is divided into four sections:

- *Script Options*
- *Tethered Jaz Network Settings*
- *Script Settings*
- *Run Section*

Script Options

Option	Meaning
Verbose	Corresponds to the -v option and provides the most detailed output
Compile Only	Corresponds to the -c option and compiles but does not run the script
Trace Execution	Corresponds to the -t option and provides a full script engine output
Capture Output	Captures script engine output in the file you specify as the capture file and will open the capture file in WordPad.exe when script execution terminates.

Tethered Jaz Network Settings

This section allows you to set the IP address of the attached Jaz unit and change the port number. Usually, only the IP address needs to be set. Changing the port number arbitrarily may result in loss of function.

The IP address control allows you to enter the IP address as 4 octets. It will not allow the entry of illegal values. Consult your system administrator for additional network information. Leave the port number at 6626 unless you have been told otherwise by Ocean Optics technical support.

Script Settings

This section allows you to specify the location of your script and the location of the scripting engine. The buttons to the right of each field allow you to select the files graphically. The scripting engine is named Scriptor.exe (unless you have renamed it). The **Keep process window open after run completion** checkbox allows you to view the raw output of the script engine without capturing it to a file. Normally, the process window closes automatically as soon as the engine execution is finished. If this box is checked, the launcher will not be available until the process window is manually closed. Also, this option is incompatible with the capture option so only one of these features may be in force at any time. You may run the script engine without capture or watching the process window. In this case, no data reflecting the current run will be retained.

Run Section

This section has two buttons and a view: **Press Run to launch the scripting engine** and **Quit to end the application**. All pertinent settings are retained by the system so that most configuration tasks won't have to be repeated on every run. The Command view shows you what the command line would look like if launched manually.

Chapter 5

Using the Jaz Scripting Engine

Overview

The scripting engine is a program named Scriptor. Scriptor contains all the functionality needed to communicate with a Jaz unit and execute user scripts. Scriptor has several arguments that may be passed to it on the command line to modify its behavior.

The scripting engine has two script file input modes:

- In one mode, the names of the script(s) are provided on the command line. The engine will attempt to execute each script in order of appearance on the command line.
- In the second mode, a flag is passed with a text string naming a directory from which the engine should fetch scripts.

In either mode, each script will execute independently of any other scripts that have been or will be executed. Minimal use would pass the name of a single script on the command line.

The flags are specified on the command line as a hyphen followed by a letter. A flag may take no arguments or it may take one. The format follows POSIX program invocation conventions. Flags may appear in any order but the name of scripts must follow all flags or an error will be reported.

The invocation format is: `Scriptor <flags and options> <script name(s)>`

Engine Arguments

Flag	Arguments	Meaning
-v	none	verbose output -- enhances the trace option
-t	none	trace -- displays all engine actions
-c	none	compile only -- will check script for syntax but will not attempt to execute
-h	IP address of Jaz Unit Default: 127.0.0.1	Specifies the IP address to contact the Jaz unit. If not provided, engine assumes that the Jaz components are local. See *Tethered Mode*.

Flag	Arguments	Meaning
-p	IP Port number of Jaz unit Default: 6626	Specifies the IP port the Jaz unit is listening on. If not provided, engine uses the default.
-f	directory name	If the engine is to fetch scripts from a directory then this flag specifies the directory to fetch from.

Tethered Mode

In this mode, the engine runs on a PC and is connected to a Jaz unit via ethernet. All functionality is available on the Jaz. This provides an ideal way to author and test scripts prior to deployment. In this mode, you must specify the -h option for Jaz address. The port option need not be used if the Jaz unit is in its default setup. All trace and operational information is available if the -t and optionally -v flags are used. If the script name is my_script and we are running in Tethered mode, the command line might look like this:

```
Scriptor -t -v -h 192.168.0.23 my_script
```

When a script has been tested, it can be written to an SD memory card for use directly on Jaz (see *Jaz Mode*). When Jaz starts up it looks for files in its special directory. If any are found, a loader presents the files in a scrolling list. When a list item is selected, Scriptor will be invoked with the selected script name. Running a script from the SD card on Jaz is known as running in Jaz mode.

Jaz Mode

In Jaz mode, the script to be run is stored in a specific directory on an SD memory card. Note that in this mode, the only way to interact with a user is through menus and button presses. No other tracing or debugging output is generated or available. Only debugged scripts should be run this way. If Jaz is to be subsequently used in its default mode without being restarted, then scripts should be written so that the top menu has a selection that allows the user to exit the script. Otherwise, the SD card must be removed and Jaz restarted.

Files must be written to the SD card in the following layout:

```
/boot/Scriptor/Scriptor      Executable for Jaz

Script1.txt                  Can use any filename, but must have .txt extension
Script2.txt
.
.
.
/boot/boot-apps.txt          Contains one line: Scriptor
```

See the sample SD card layout provided in the zip file.

Chapter 6

Functions Reference

ACOS

Syntax: `ACos(A,B)`

Arguments

Name	Type	Special Considerations
A	INT_16 or REAL variable or literal	
B	REAL variable	B = arccos(A)

Adapt

Syntax: `Adapt(Channel)`

Arguments

Name	Type	Special Considerations
Channel	Integer	Desired spectrometer channel

The algorithm begins with the built-in variables INTEGRATION_TIME_SECONDS and INTEGRATION_TIME_NANOSEC as the integration time. It adjusts up or down by 100 each time a failure occurs until a maximum/minimum limit is reached or the response to the request is successful. The built-in variable assumes the value at that time.

Setting this value prior to making the call will allow for faster convergence.

Add

Syntax: `Add(A,B,C)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	
C	Spectral	A + B Pixel Values

Intent:

Add the pixel values of spectra A and B and store in C

ASIN

Syntax: `ASin(A,B)`

Arguments

Name	Type	Special Considerations
A	INT_16 or REAL variable or literal	
B	REAL variable	B = arcsin(A)

AssignLampType

Syntax: `AssignLampType(LAMP_CONSTANT,X)`

Arguments

Name	Type	Special Considerations
LAMP_CONSTANT	INT_16	Built-in like UV_VIS
X	Integer variable	

Intent:

If a lamp of the given type is present, set its X to its index number so that X can be used in controlling the lamp. X = -1 if an instance of the lamp type is not present.

Call -- UserFunctionInvocation

Syntax: `Call MyProcedure`

Arguments

Name	Type	Special Considerations
MyProcedure	User-defined procedure	A subroutine call

Intent:

To invoke user-defined code as a unit of execution.

Check

Syntax:

```
Check(DeviceClass, Instance,Setting RelOp Value
Check(DeviceClass, Instance,Setting IN [low,high]
Check(DeviceClass, Instance,Setting OUTSIDE [low,high]
Check(DeviceClass, Instance,Setting IN {a,b,c,…,}
Check(DeviceClass, Instance,Setting OUTSIDE {a,b,c,…,}
Check(DeviceClass, Instance,Setting OneOf {a,b,c,…,}
```

Arguments

Name	Type	Special Considerations
DeviceClass	SPECTROMETER, LAMP	Built-in Values
Instance	INT_16 or literal	Channel or unit number
Setting	INT_16 or literal	Built-In Values
Relop	One of <,>,<>,=,>=,<=	
Value	INT_16 or literal	

Name	Type	Special Considerations
IN	Operator for membership	
OUTSIDE	Operator for non-membership	
[a,b]	Numeric range (real or integer)	A < B
{a,b,c}	Set of discrete values	No imposed order

Intent:

Check that a system attribute, feature or device is or is not present or is sufficient to allow script to proceed. If a check statement fails, the script terminates.

CloseFile

Syntax: `CloseFile(F)`

Arguments

Name	Type	Special Considerations
F	FILE	

Intent:

Finalizes access to a file.

Comment

Syntax: `//This is a comment`

A comment is any text following '//' to the end of a line.

Name	Type	Special Considerations
//	Comment start	// to end of line ignored in script

Comp

Syntax: `Comp(A,B,C)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	
C	Real variable	C = Sum(Ai - Bi)

Intent:

Compute the sum of the pixel value differences of A and B and store in C.

Compabs

Syntax: `CompAbs(A,B,C)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	
C	Real variable	C = Sum(Abs(Ai - Bi))

Intent:

Compute the sum of the absolute values of the pixel values of A and B and store in C.

COS

Syntax: `Cos(A,B)`

Arguments

Name	Type	Special Considerations
A	INT_16 or REAL variable or literal	
B	REAL variable	B = cos(A)

Display

Syntax: `Display(Legend,Value,Units)`

Arguments

Name	Type	Special Considerations
Legend	TEXT	
Value	REAL	
Units	TEXT	

Intent:

Display output value message on Jaz OLED display.

Displaymsg

Syntax: `DisplayMsg(Message)`

Arguments

Name	Type	Special Considerations
Message	TEXT	

Intent:

Show the message on the Jaz OLED display.

Do … Done

Syntax: `Do V SV,EV,SS Done`

Example:

```
Do I 1,10,1
Sum := Sum + I
Done
```

This is the basic loop iteration structure where the loop body is delimited by the keywords DO and DONE.

Arguments

Name	Type	Purpose
V	integer	Iteration variable
SV	constant	initial value of iteration variable
EV	constant	value of variable to terminate loop
SS	constant	increment V by other than 1

Duplicate

Syntax: `Duplicate(A,B)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	B = A

EXP

Syntax: `Exp(A,B)`

Arguments

Name	Type	Special Considerations
A	INT_16 or REAL variable or literal	
B	REAL variable	B = e to the power of A

GetIntegrationTime

Syntax: `GetIntegrationTime(R)`

Arguments

Name	Type	Special Considerations
R	REAL variable	Value of INTEGRATION_TIME_SECONDS + INTEGRATION_TIME_NANOSECS

Intent:

Get current value of Integration Time variables.

GetLampIntensity

Syntax: `GetLampIntensity(L,V,M)`

Arguments

Name	Type	Special Considerations
L	Integer literal or variable	Lamp unit's number
V	Integer variable	Intensity value
M	BULB1, BULB2, ALLBULBS	Target bulb of the source to effect

Intent:

Fetch current value of light source bulb intensity setting.

GetLampShutter

Syntax: `GetLampShutter(L,V)`

Arguments

Name	Type	Special Considerations
L	Integer variable or literal	Lamp unit's number
V	Integer variable	

Intent:

Get the current light source shutter setting.

GetSpectrum

Syntax: `GetSpectrum(Index,A)`

Arguments

Name	Type	Special Considerations
Index	INT_16 constant or literal number Or FILE variable	Device must exist Or File must exist and be opened
A	Spectral	Spectral data read

Goto

Syntax: `Goto Label`

Arguments

Name	Type	Special Considerations
Label	A labeled statement in the script	Labels are local to main or a user-defined procedure

If

Syntax:

```
If(conditional expression) GOTO Label
If(conditional expression) Procedure Name
If(conditional expression) HALT
```

Arguments

Name	Type	Special Considerations
Conditional expression	Boolean expression	Example (A <> B)
Label	User defined Label	
Procedure Name	Name of a user-defined procedure	Call user expression and return
HALT	Termination of script	

Intent:

Conditional execution of script statements.

Label

Syntax: `Label <Label Name>`

Arguments

Name	Type	Special Considerations
Label Name	Tagged instruction in the script defined by the user	Target for Goto expressions

LocateWavelength

Syntax: LocateWavelength (A, B, C)

Arguments

Name	Type	Special Considerations
A	Spectral	
B	INT_16 or REAL variable or literal	
C	INT_16 literal	

Intent:

Finds location in spectrum of B wavelength.

Log

Syntax: `Log(A,B)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	Bi = Log10(Ai) iff Ai > 0 Bi = INF Otherwise

Intent:

Compute the element-wise logarithm base 10 of A and store in elements of B. If an element of A is ≤0 , then the value stored in INF (infinity) and is not computationally useful.

Log10

Syntax: `Log10(A,B)`

Arguments

Name	Type	Special Considerations
A	INT_16 or REAL variable or literal	Greater than 0
B	Real	B = Log10(A)

Intent:

Compute the base 10 log of the scalar variable, A. Do not confuse this with LOG(), which operates only on spectral data.

LogN

Syntax: `LogN(A,B)`

Arguments

Name	Type	Special Considerations
A	INT_16 or REAL variable or literal	A > 0
B	REAL variable	B = ln(A)

Mult

Syntax: `Mult(A,B,C)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	
C	Spectral	Ci = Ai * Bi

Intent:

Compute the DOT-PRODUCT of A,B and store in C.

Norm

Syntax: `Norm(A,B)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	Bi = Ai/Max(Ai)

Intent:

Normalize the pixel values of A by dividing through by the maximum pixel value in A and store the ratio in B. If Max(A) is zero then the result is INF and is not computationally useful.

OnButtonClick

Syntax: `OnButtonClick(ButtonSelection, TimeToWait)`

Arguments

Name	Type	Special Considerations
ButtonSelection	INT_16	Variable holding user's selection and -1 if timed out or some other selection activity was done. If TimeToWait is zero and a button press occurred the keycode value +1000 will be returned as the ButtonSelection.
TimeToWait	INT_16	The number of seconds the script should wait for user input. If zero is sent as the TimeToWait then the command will return -1 until a button press has occurred and then the keycode value +1000 will be returned.

Intent:

To interact with the user via the Jaz menu screen and the direction (arrow) buttons.

OnError

Syntax: `OnError(TextMessage) GUI Function`

Arguments

Name	Type	Special Considerations
TextMessage	Text N	

OnErrorGoto

Syntax:

```
OnErrorGoTo Label
OnErrorGoTo Procedure
OnErrorGoTo HALT
```

Arguments

Name	Type	Special Considerations
Label	A labeled statement in the script	Labels are local to main or a user-defined procedure
Procedure	The name of a user-defined procedure	
HALT	Terminate the script	

Intent:

The engine maintains a global error indicator that is set whenever a consistency error is detected and referenced by this function to determine whether to take the action specified.

OpenFile

Syntax: `OpenFile(F,Mode)` variant : `OpenFile(F,Mode,Sequence)`

Arguments

Name	Type	Special Considerations
F	FILE variable	
Mode	INT_16 constant	ForRead = file is for output ForWrite = file is for input

Name	Type	Special Considerations
Sequence	INT_16 variable or literal	Use if this parameter is not mandatory. If used and F was declared with name myfile.txt, then the actual file name becomes myfile.txt.nnn where : nnn is a three digit numerical field derived from this argument. User is responsible for setting, incrementing and contents of this parameter.

Intent:

Registers file variable with operating system controlled file. The sequence number allows a series of related measurements to be saved to files having a common base name.

Pause

Syntax: `Pause(Seconds)`

Arguments

Name	Type	Special Considerations
Seconds	INT_16	Number of seconds to pause the script execution

POW

Syntax: `Pow(A,B,C)`

Arguments

Name	Type	Special Considerations
A	INT_16 or REAL variable or literal	
B	INT_16 or REAL variable or literal	
C	REAL	C = A to the power of B

Prompt

Syntax: `Prompt(Message)`

Arguments

Name	Type	Special Considerations
Message	TEXT	

Intent:

Display a prompt message on the Jaz OLED display (usually to prompt the user to perform an action).

Ratio

Syntax: `Ratio(A,B,C)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	
C	Spectral	Ci = Ai / Bi iff Ci <>0 Ci = INF otherwise

Intent:

Compute the ratio of elements in A and B and store as elements of C.

ReadRealVector

Syntax: `ReadRealVector(F,V,Length)`

Arguments

Name	Type	Special Considerations
F	FILE	
V	Real Array variable	
Length	Integer variable or literal	Number of elements to read

Intent:

Read in a vector of real values from a user-supplied file. Useful in creating tables, references, lists, etc.

ReadTable

Syntax: ReadTable(FileVar,TableVar,NumRows,NumCols)

Arguments

Name	Type	Special Considerations
FileVar	FILE	Must have been opened with OpenFile()
TableVar	TABLE	#rows and #cols specified in declaration
NumRows	INT_16	Input file must match this specification
NumCols	INT_16	Input file must match this specification

Intent:

To read in a 2-dimensional array of real numbers from a file. The array columns represent reference spectra for use in pattern matching with a sample spectrum. The number of rows will typically be 2048, which is the default length of a spectrum.

Savereading

Syntax: `SaveReading (F,Legend,Index,Units Text)`

Arguments

Name	Type	Special Considerations
F	FILE	Must be ForWrite
Legend	TEXT	Like “Spot Value”
Index	Integer variable or literal	Index into pixel values
Units	TEXT	Like “nM or Angstroms”

Intent:

Put human-readable text into a file.

Example:

```
SaveReading(MyFile, "My Reading Text:" , Variable, "nm/s Units Text")
```

Scale

Syntax: `Scale(<arithmetic expression>,A)`

Arguments

Name	Type	Special Considerations
Arithmetic expression	Variable, constant or an expression that evaluates to a numeric value	Serves as multiplier
A	Spectral	A = z*A

SetDisplayPrecision

Syntax: `SetDisplayPrecision(W,P)`

Arguments

Name	Type	Special Considerations
W	INT_16	W is width of numeric field
P	INT_16	P is number of digits precision

Intent:

To set the display precision of real numbers.

SetIntegrationTime

Syntax: `SetIntegrationTime(R)`

Arguments

Name	Type	Special Considerations
R	REAL variable or literal	Set value of INTEGRATION_TIME_SECONDS + INTEGRATION_TIME_NANOSECS

Intent:

Set value of Integration Time variables.

SetLampIntensity

Syntax: SetLampIntensity(L, M,V)

Arguments

Name	Type	Special Considerations
L	Integer variable or literal	Lamp unit's number
M	BULB1, BULB2, ALLBULBS	Target bulb of the source to effect
V	Integer variable or literal	Setting (must be in range)

Intent:

Change the intensity of bulb in the light source by a given amount.

SetLampShutter

Syntax: SetLampShutter(L,V)

Arguments

Name	Type	Special Considerations
L	Integer variable or literal	Lamp unit's number
V	Integer variable or literal	

Intent:

Set the current light source shutter setting.

ShowGraph

Syntax: `ShowGraph(S)`

Arguments

Name	Type	Special Considerations
S	Spectral	

Intent:

To display a graph of spectrum S on Jaz.

ShowMenu

Syntax: `ShowMenu(s1{,s2,s3,s4})`

Where: s = a string or a text variable.

s2 through s4 are optional and may be omitted for a 1 line menu.

Each string constitutes a line on the display and is associated automatically with the navigation buttons on the Jaz unit.

Arguments

Name	Type	Special Considerations
s1	TEXT	Mandatory field
s2 ... s4	TEXT	Optional arguments

Intent:

To display user input prompts on the Jaz screen. *OnButtonClick* function is usually used with this function to retrieve the user's selection.

SIN

Syntax: `Sin(A,B)`

Arguments

Name	Type	Special Considerations
A	INT_16 or REAL variable or literal	
B	REAL variable	B = sin(A)

Sub

Syntax: `Sub(A,B,C)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	
C	Spectral	A -B Pixel values

Intent:

Subtract the pixel values of spectra A and B and store in C.

Trim

Syntax: `Trim(A,B,From,To)`

Arguments

Name	Type	Special Considerations
A	Spectral	
B	Spectral	B is all values of A from FROM to TO
From	Low wavelength value	
To	High wavelength value	

WaveLength

Syntax: WaveLength(A,V,B)

Arguments

Name	Type	Special Considerations
A	Spectral	
V	Integer value or variable	
B	Real	Wavelength of pixel at location V

Intent:

To get the wavelength corresponding to a pixel location.

WriteFile

Syntax:

`WriteFile(F,X)`

`WriteFile(F,A,M,N)`

Arguments

Name	Type	Special Considerations
F	FILE	
X	Variable	Any type supported
A	Spectral	
M	Low index	Literal or variable
N	High Index	Literal or variable

Intent:

Send data to user-defined file.

WriteSpectrum

Syntax: WriteSpectrum(F,A)

Arguments

Name	Type	Special Considerations
F	FILE	Type determines the format of the output (wavelength,pixel value)
A	Spectral	

Intent:

Write spectral data to user-defined file.

Appendix A

Example Scripts

Bare Script Template

```
SCRIPT Testing1
VERSION 1.0.0
VARIABLES ////////////////////////////////////////////////
//
//////////////////////////////////////////////////////////

END //variables

CONSTANTS ////////////////////////////////////////////////
//
//////////////////////////////////////////////////////////

END //constants

START ////////////////////////////////////////////////
//
//////////////////////////////////////////////////////////

END
//////////////////////////////////////////////////////////
STOP
```

Complete Example Script -- (Demonstrates syntax)

This example demonstrates syntax.

A script must have a name and a 3-digit version number. As the product evolves, new script engines will be produced, so it is important that a script targeted for a later version is not sent to an earlier version engine. The intention is that all script engines be backward compatible with version 1.0.0. The script writer adjusts this version tag to guarantee a match between script engines and the script.

```
SCRIPT Simple_test_of_scriptor

Version 0.0.1

VARIABLES

//
// File to store the Spectrum that we get
//
testfile1 FILE "testfile1.txt" ASCII


//
// SPECTRAL variable for the spectrum that we get from the Spectrometer
//
testspectrum SPECTRAL

//
// Channel that we are getting the spectrum from
//
channel INT_16

//
// Button choice variable
//
choice INT_16

//
// Variable to hold the Integration Time
//
integrationtime REAL

END

//
//  Start of the Script
//
START

//
// Set variables
//
channel := 0
integrationtime := 0.0004

//
//  Set the display precision
//
SetDisplayPrecision(8,6)
```

```
//
//  Set the integration time
//
Display("Setting$Integration$time to$",integrationtime,"secs")
pause(2)
SetIntegrationTime(integrationtime)

//
//  set a label so that we can return to this location.
//
LABEL TOP

//
//Display a message and get a spectrum
//
DisplayMsg("Collecting$Spectrum")
pause(2)
GetSpectrum(channel,testspectrum)

//
//  Open a file and save the Spectrum
//
OpenFile(testfile1,ForWrite)
WriteSpectrum(testfile1,testspectrum)
CloseFile(testfile1)

//
// Display the Spectrum that we just got and
// Pause so that it can be seen
//
ShowGraph(testspectrum)
Pause(3)

//
//  Put up a menu for the user to see if they
//  want to get another Spectrum or exit
ShowMenu("Get another?","Exit")
OnButtonClick(choice, 30)

//
//  If the user wants another spectrum goto the
//  TOP Label otherwise exit
//
If (choice = 0) GOTO TOP

DisplayMsg("Exiting")
Pause(5)

END
STOP
```

```
    ////////////////////////////////////////////////////////
    //
    //
    ////////////////////////////////////////////////////////

    [PROCESS MYGETSPECTRUM]
    MySendPacket[10] := -1956.0
    RealDumb[19] := MySendPacket[10]
    myname := "SPECTRUM"
    LampIndex := 6
    GetLampShutter(LampIndex,LampShutter)
    SetLampShutter(LampIndex, 20)
    END
    STOP
```

pH Measurement Script

SCRIPT pH_Module_Pittcon

Version 0.3.1

Variables

```
MaxFile FILE "MaxFile.txt" csv
DarkFile FILE "DarkFile.txt" csv
LowFile FILE "LowFile.txt" csv
pH_Save FILE "pH_Save.txt" csv
UserSelection int_16
DarkSpectrum SPECTRAL
LowpHSpectrum SPECTRAL
MaxpHSpectrum SPECTRAL
SamplepHSpectrum SPECTRAL
QuickSpectrum SPECTRAL
FiveSpectrum SPECTRAL
SixSpectrum SPECTRAL
SevenSpectrum SPECTRAL
EightSpectrum SPECTRAL
LowpHSpectrumDark SPECTRAL
MaxpHSpectrumDark SPECTRAL
SamplepHSpectrumDark SPECTRAL
QuickSpectrumDark SPECTRAL
```

```
FiveSpectrumDark SPECTRAL
SixSpectrumDark SPECTRAL
SevenSpectrumDark SPECTRAL
EightSpectrumDark SPECTRAL
pH REAL
LowPeak REAL
LowIntensityPeak INT_16
LowIntensityBase INT_16
SampleIntensityPeak INT_16
SampleIntensityBase INT_16
SamplePeakRatio REAL
SampleBaseRatio REAL
SamplePeakAbsorbance REAL
SampleBaseAbsorbance REAL
SampleAbsorbance REAL
MaxIntensityPeak INT_16
MaxIntensityBase INT_16
MaxPeakRatio REAL
MaxBaseRatio REAL
MaxPeakAbsorbance REAL
MaxBaseAbsorbance REAL
MaxAbsorbance REAL
QuickIntensityPeak INT_16
QuickIntensityBase INT_16
QuickPeakRatio REAL
QuickBaseRatio REAL
QuickPeakAbsorbance REAL
QuickBaseAbsorbance REAL
QuickAbsorbance REAL
FiveIntensityPeak INT_16
FiveIntensityBase INT_16
FivePeakRatio REAL
FiveBaseRatio REAL
FivePeakAbsorbance REAL
FiveBaseAbsorbance REAL
FiveAbsorbance REAL
SixIntensityPeak INT_16
SixIntensityBase INT_16
SixPeakRatio REAL
```

```
SixBaseRatio REAL
SixPeakAbsorbance REAL
SixBaseAbsorbance REAL
SixAbsorbance REAL
SevenIntensityPeak INT_16
SevenIntensityBase INT_16
SevenPeakRatio REAL
SevenBaseRatio REAL
SevenPeakAbsorbance REAL
SevenBaseAbsorbance REAL
SevenAbsorbance REAL
EightIntensityPeak INT_16
EightIntensityBase INT_16
EightPeakRatio REAL
EightBaseRatio REAL
EightPeakAbsorbance REAL
EightBaseAbsorbance REAL
EightAbsorbance REAL
LogArgument REAL
LogTerm REAL
LogArgumentFive REAL
LogTermFive REAL
LogArgumentSix REAL
LogTermSix REAL
LogArgumentSeven REAL
LogTermSeven REAL
LogArgumentEight REAL
LogTermEight REAL
LogArgumentRefresh REAL
LogTermRefresh REAL
PassCount int_16
UserChoice INT_16
pK REAL
pK1 REAL
pK2 REAL
pK3 REAL
pK4 REAL
pKSum REAL
Slope REAL
```

```
Slope1 REAL
Slope2 REAL
Slope3 REAL
Slope4 REAL
Slope5 REAL
Slope6 REAL
SlopeSum REAL
imax1 INT_16
MaxIntensityPeak1 INT_16
LowIntensityPeak1 INT_16
MaxPeakRatio1 REAL
MaxPeakAbsorbance1 REAL
QuickIntensityPeak1 INT_16
QuickPeakRatio1 REAL
QuickPeakAbsorbance1 REAL
QuickAbsorbance1 REAL
FiveIntensityPeak1 INT_16
FivePeakRatio1 REAL
FivePeakAbsorbance1 REAL
FiveAbsorbance1 REAL
SixIntensityPeak1 INT_16
SixPeakRatio1 REAL
SixPeakAbsorbance1 REAL
SixAbsorbance1 REAL
SevenIntensityPeak1 INT_16
SevenPeakRatio1 REAL
SevenPeakAbsorbance1 REAL
SevenAbsorbance1 REAL
EightIntensityPeak1 INT_16
EightPeakRatio1 REAL
EightPeakAbsorbance1 REAL
EightAbsorbance1 REAL
SampleIntensityPeak1 INT_16
SamplePeakRatio1 REAL
SamplePeakAbsorbance1 REAL
SampleAbsorbance1 REAL
i int_16
Light1 INT_32
LightSpectrum SPECTRAL
```

```
LightPeak INT_16
MaxPeak INT_16
QuickPeak INT_16
FivePeak INT_16
SixPeak INT_16
SevenPeak INT_16
EightPeak INT_16
SamplePeak INT_16

END

CONSTANTS

TimeOutSeconds = 60
MaxPasses = 100

END

START

PassCount := 0
SetDisplayPrecision(6,2)

// Set Integration Time, Averaging, BoxCar, and default pK and Slope
SPECTROMETER_CHANNEL_NUMBER := 0
INTEGRATION_TIME_SECONDS := 0
INTEGRATION_TIME_NANOSEC := 3000
AVERAGES := 15
BOX_CAR := 15
pK := 6.0
Slope := 1.5

// Power Lamp

DisplayMsg("Light adjustment$in progress")

Pause(2)

SetLampShutter(0,1)
```

```
Light1 := 4000

LABEL LightLoop
SetLampIntensity(0,ALLBULBS,Light1)
GetSpectrum(0,LightSpectrum)
LocateWavelength(LightSpectrum,566,LightPeak)
IF(LightSpectrum[LightPeak] > 32000) GOTO Again
GOTO EndLight

LABEL Again
Light1 := Light1 - 100
SetLampIntensity(0,ALLBULBS,Light1)
GOTO LightLoop

LABEL EndLight
DisplayMsg("Light adjustment$complete")
Pause(2)
Display("Max Int = ",LightSpectrum[LightPeak],"")
Pause(2)

// Main User Interface
Label Top
CALL MainMenu

If(UserChoice = 0) GOTO Standardize
If(UserChoice = 1) GOTO VIEW
If(UserChoice = 2) GOTO RefreshpK
GOTO QUIT

LABEL Standardize
CALL GetStandards
GOTO TOP

LABEL View
Call ViewpH
GOTO TOP

LABEL RefreshpK
```

```
Call Refresh
GOTO TOP

LABEL QUIT
DisplayMsg("Ending$Session")
Pause(2)

END

//Menu to standardize

[Process  GetStandards]
LABEL TOP
ShowMenu("Take Dark Ref","Take Low pH Ref","Take Max pH Ref","Back")
OnButtonClick(UserSelection,TimeOutSeconds)
If(UserSelection = 0) GOTO Dark
If(UserSelection = 1) GOTO Low
If(UserSelection = 2) GOTO Max
If(UserSelection = 3) GOTO EXIT

LABEL Dark
Call GetTheDarkRef
GOTO TOP

LABEL Low
Call GetTheLowRef
GOTO TOP

LABEL Max
Call GetTheMaxRef
GOTO TOP

LABEL EXIT
END

[Process MainMenu]

PassCount := 0
```

```
LABEL TOP
ShowMenu("Standardize","View pH","Refresh pK","Exit")
OnButtonClick(UserSelection,TimeOutSeconds)
If(UserSelection <> 0) GOTO Check1
UserChoice := 0
GOTO EXIT

LABEL Check1
If(UserSelection <> 1) GOTO Check2
UserChoice := 1
GOTO EXIT

LABEL Check2
If(UserSelection <> 2) GOTO Check3
UserChoice := 2
GOTO EXIT

LABEL Check3
If(UserSelection <> 3) GOTO Bummer
UserChoice := 3
GOTO EXIT

LABEL bummer
PassCount := PassCount + 1
If(PassCount > MaxPasses) GOTO QUIT
DisplayMsg("Please select$something")
Pause(10)
GOTO TOP

LABEL QUIT
UserSelection := -1
LABEL EXIT
END

[PROCESS GetTheDarkRef]

// Turns off lamp, takes dark reference, repowers lamp, adjusts IntTime
SetLampIntensity(0,ALLBULBS,0)
SetLampShutter(0,0)
```

```
Pause(1)

GetSpectrum(0,DarkSpectrum)

DisplayMsg("Taking Dark$Reference")
Pause(2)
SetLampShutter(0,1)
SetLampIntensity(0,ALLBULBS,Light1)

END

[PROCESS GetTheLowRef]

// Prompts user to insert buffer sample, takes low reference
DisplayMsg("pH = 1 Buffer$Should Be$Present")
Pause(3)
GetSpectrum(0,LowpHSpectrum)

Sub(LowpHSpectrum, DarkSpectrum, LowpHSpectrumDark)
Pause(3)

END

[PROCESS GetTheMaxRef]

/// Prompts user to insert buffer sample, takes max reference
DisplayMsg("pH = 11 Buffer$Should Be$Present")
Pause(3)
GetSpectrum(0,MaxpHSpectrum)

Sub(MaxpHSpectrum, DarkSpectrum, MaxpHSpectrumDark)

OpenFile(MaxFile, ForWrite)
WriteSpectrum(MaxFile,MaxpHSpectrumDark)
CloseFile(MaxFile)

OpenFile(DarkFile, ForWrite)
WriteSpectrum(DarkFile,DarkSpectrum)
CloseFile(DarkFile)
```

```
OpenFile(LowFile, ForWrite)
WriteSpectrum(LowFile,LowpHSpectrumDark)
CloseFile(LowFile)

LocateWavelength(MaxpHSpectrumDark, 750, MaxIntensityBase)
LocateWavelength(LowpHSpectrumDark, 750, LowIntensityBase)

MaxBaseRatio := MaxpHSpectrumDark[MaxIntensityBase] /
LowpHSpectrumDark[LowIntensityBase]
LOG10(MaxBaseRatio, MaxBaseAbsorbance)
MaxBaseAbsorbance := -(MaxBaseAbsorbance)

LocateWavelength(MaxpHSpectrumDark, 618, MaxIntensityPeak1)
LocateWavelength(LowpHSpectrumDark, 618, LowIntensityPeak1)

MaxPeakRatio1 := MaxpHSpectrumDark[MaxIntensityPeak1] /
LowpHSpectrumDark[LowIntensityPeak1]
LOG10(MaxPeakRatio1, MaxPeakAbsorbance1)
MaxPeakAbsorbance1 := -(MaxPeakAbsorbance1)

MaxAbsorbance := MaxPeakAbsorbance1 - MaxBaseAbsorbance

Display("Max Abs = ",MaxAbsorbance,"")

Pause(2)

END

//Reset pK and Slope Menu

[PROCESS Refresh]

LABEL TOP
ShowMenu("Manual Entry","Quick Reset","Full Reset","Back")
OnButtonClick(UserSelection,TimeoutSeconds)
If(UserSelection = 0) GOTO Manual
If(UserSelection = 1) GOTO Quick
If(UserSelection = 2) GOTO Full
If(UserSelection = 3) GOTO EXIT
```

```
GOTO EXIT

LABEL Manual
//Not yet a feature in Scriptor
DisplayMsg("Not yet a$usable feature")
Pause(2)
GOTO TOP

//Quick single buffer calibration

LABEL Quick
DisplayMsg("Make sure Dark,$Low, and Max Refs$have been taken")
Pause(3)
DisplayMsg("pH = 7 Buffer$Should Be$Present")
Pause(2)
GetSpectrum(SPECTROMETER_CHANNEL_NUMBER, QuickSpectrum)
Sub(QuickSpectrum, DarkSpectrum, QuickSpectrumDark)

iMax1 := 596
LocateWavelength(QuickSpectrumDark, 750, QuickIntensityBase)
LocateWavelength(LowpHSpectrumDark, 750, LowIntensityBase)

QuickBaseRatio := QuickSpectrumDark[QuickIntensityBase] /
LowpHSpectrumDark[LowIntensityBase]
LOG10(QuickBaseRatio, QuickBaseAbsorbance)
QuickBaseAbsorbance := -(QuickBaseAbsorbance)

QuickAbsorbance := 0

LABEL Loop2
iMax1 := iMax1 + 1

LocateWavelength(QuickSpectrumDark, iMax1, QuickIntensityPeak1)
LocateWavelength(LowpHSpectrumDark, iMax1, LowIntensityPeak1)

QuickPeakRatio1 := QuickSpectrumDark[QuickIntensityPeak1] /
LowpHSpectrumDark[LowIntensityPeak1]
LOG10(QuickPeakRatio1, QuickPeakAbsorbance1)
QuickPeakAbsorbance1 := -(QuickPeakAbsorbance1)
```

```
QuickAbsorbance1 := QuickPeakAbsorbance1 - QuickBaseAbsorbance

if(QuickAbsorbance1 >= QuickAbsorbance) GOTO AssignQuick
If(iMax1 < 645) GOTO Loop2
GOTO Calculate2

LABEL AssignQuick
QuickAbsorbance := QuickAbsorbance1
QuickPeak := QuickIntensityPeak1
GOTO Loop2

LABEL Calculate2

LogArgumentRefresh := QuickAbsorbance/(MaxAbsorbance - QuickAbsorbance)
if(LogArgumentRefresh> 0.0) GOTO OK6
GOTO ABORT

Label OK6
LOG10(LogArgumentRefresh, LogTermRefresh)
pK := -(1.9287 * LogTermRefresh) + 7.069
Slope := 11.19225 / pK
DisplayMsg("Quick Reset$Successful")
Pause(2)
Display("pK = ",pK,"")
Pause(2)
Display("Slope = ",Slope,"")
Pause(2)
GOTO TOP

Label ABORT
DisplayMsg("Internal$Math$ERROR")
Pause(2)

GOTO TOP

//Full four buffer calibration

LABEL Full
```

```
DisplayMsg("Make sure Dark,$Low, and Max Refs$have been taken")
Pause(3)
DisplayMsg("You will need$pH 6 and 8$buffers")
Pause(3)
LABEL FullMenu
ShowMenu("pH = 6","pH = 8")
OnButtonClick(UserSelection,TimeOutSeconds)
If(UserSelection = 0) GOTO Six
If(UserSelection = 1) GOTO Eight

LABEL Six
DisplayMsg("pH = 6 Buffer$Should Be$Present")
Pause(2)
GetSpectrum(SPECTROMETER_CHANNEL_NUMBER, SixSpectrum)
Sub(SixSpectrum, DarkSpectrum, SixSpectrumDark)

LocateWavelength(SixSpectrumDark, 750, SixIntensityBase)
LocateWavelength(LowpHSpectrumDark, 750, LowIntensityBase)

SixBaseRatio := SixSpectrumDark[SixIntensityBase] /
LowpHSpectrumDark[LowIntensityBase]
LOG10(SixBaseRatio, SixBaseAbsorbance)
SixBaseAbsorbance := -(SixBaseAbsorbance)

LocateWavelength(SixSpectrumDark, 618, SixIntensityPeak1)
LocateWavelength(LowpHSpectrumDark, 618, LowIntensityPeak1)

SixPeakRatio1 := SixSpectrumDark[SixIntensityPeak1] /
LowpHSpectrumDark[LowIntensityPeak1]
LOG10(SixPeakRatio1, SixPeakAbsorbance1)
SixPeakAbsorbance1 := -(SixPeakAbsorbance1)

SixAbsorbance := SixPeakAbsorbance1 - SixBaseAbsorbance

LogArgumentSix := SixAbsorbance/(MaxAbsorbance - SixAbsorbance)
if(LogArgumentSix > 0.0) GOTO OK16
GOTO ABORT

LABEL OK16
LOG10(LogArgumentSix, LogTermSix)
```

```
GOTO FullMenu

LABEL Eight
DisplayMsg("pH = 8 Buffer$Should Be$Present")
Pause(2)
GetSpectrum(SPECTROMETER_CHANNEL_NUMBER, EightSpectrum)
Sub(EightSpectrum, DarkSpectrum, EightSpectrumDark)
LocateWavelength(EightSpectrumDark, 750, EightIntensityBase)
LocateWavelength(LowpHSpectrumDark, 750, LowIntensityBase)

EightBaseRatio := EightSpectrumDark[EightIntensityBase] /
LowpHSpectrumDark[LowIntensityBase]
LOG10(EightBaseRatio, EightBaseAbsorbance)
EightBaseAbsorbance := -(EightBaseAbsorbance)

LocateWavelength(EightSpectrumDark, 618, EightIntensityPeak1)
LocateWavelength(LowpHSpectrumDark, 618, LowIntensityPeak1)

EightPeakRatio1 := EightSpectrumDark[EightIntensityPeak1] /
LowpHSpectrumDark[LowIntensityPeak1]
LOG10(EightPeakRatio1, EightPeakAbsorbance1)
EightPeakAbsorbance1 := -(EightPeakAbsorbance1)

EightAbsorbance := EightPeakAbsorbance1 - EightBaseAbsorbance

LogArgumentEight := EightAbsorbance/(MaxAbsorbance - EightAbsorbance)
if(LogArgumentEight > 0.0) GOTO OK26
GOTO ABORT

LABEL OK26
LOG10(LogArgumentEight, LogTermEight)

Slope := 2/(LogTermEight - LogTermSix)
pK2 := 6 - (Slope*LogTermSix)
pK4 := 8 - (Slope*LogTermEight)
pKSum := pK2 + pK4
pK := pKSum / 2

DisplayMsg("Calibration$successful")
Pause(2)
```

```
Display("pK = ",pK,"")
Pause(2)
Display("Slope = ",Slope,"")
Pause(2)

GOTO Top

LABEL EXIT

END

[PROCESS ViewpH]

// View pH Interface
Label TOP
ShowMenu("Measure Sample","Save pH","Back")
OnButtonClick(UserSelection,TimeOutSeconds)
If(UserSelection = 0) GOTO CALCULATE_IT
If(UserSelection = 1) GOTO Save_pH
If(UserSelection = 2) CALL MainMenu
GOTO EXIT

LABEL CALCULATE_IT
CALL CalculatepH
GOTO TOP

LABEL Save_pH

OpenFile(pH_Save,ForWrite)
SaveReading(pH_Save,"pH = ",pH,"")
CloseFile(pH_Save)
DisplayMsg("pH Saved")
Pause(2)
GOTO TOP

LABEL EXIT

END
```

```
[PROCESS CalculatepH]

//Gets sample spectrum, calculates and displays pH, returns to previous screen
GetSpectrum(SPECTROMETER_CHANNEL_NUMBER, SamplepHSpectrum)
Sub(SamplepHSpectrum, DarkSpectrum, SamplepHSpectrumDark)

LocateWavelength(SamplepHSpectrumDark, 750, SampleIntensityBase)
LocateWavelength(LowpHSpectrumDark, 750, LowIntensityBase)

SampleBaseRatio := SamplepHSpectrumDark[SampleIntensityBase] /
LowpHSpectrumDark[LowIntensityBase]
LOG10(SampleBaseRatio, SampleBaseAbsorbance)
SampleBaseAbsorbance := -(SampleBaseAbsorbance)

LocateWavelength(SamplepHSpectrumDark, 618, SampleIntensityPeak1)
LocateWavelength(LowpHSpectrumDark, 618, LowIntensityPeak1)

SamplePeakRatio1 := SamplepHSpectrumDark[SampleIntensityPeak1] /
LowpHSpectrumDark[LowIntensityPeak1]
LOG10(SamplePeakRatio1, SamplePeakAbsorbance1)
SamplePeakAbsorbance1 := -(SamplePeakAbsorbance1)

SampleAbsorbance := SamplePeakAbsorbance1 - SampleBaseAbsorbance

LogArgument := SampleAbsorbance/(MaxAbsorbance - SampleAbsorbance)
if(LogArgument> 0.0) GOTO OK6
GOTO ABORT

Label OK6
LOG10(LogArgument, LogTerm)
pH := (Slope * LogTerm) + pK

Display("pH = ", pH, "")
Pause(5)
CALL ViewpH
GOTO EXIT
Label ABORT
DisplayMsg("Internal$Math$ERROR")
Pause(2)
LABEL EXIT
END
STOP
```

Index

G

I

L

M

N

O

P

R

S

T

U

V

W