

# LEonard

## User Manual

### PREMININARY



L  
ECKY  
ENGINEERING

Lecky Engineering, LLC  
Software Rev 2022.11.7

November 7, 2022

# LLeonard User Manual

---

# LLeonard User Manual

---

<b>OVERVIEW .....</b>	<b>5</b>
<i>An aside: LLeonard File Structure .....</i>	5
<b>ALL ABOUT LEONARD DEVICES .....</b>	<b>5</b>
<b>THE LEONARDSTATEMENT AND LEONARDMESSAGE.....</b>	<b>7</b>
<b>LEONARD DISPLAYS.....</b>	<b>7</b>
<b>LEONARD PROGRAMMING .....</b>	<b>8</b>
AN ASIDE: THE PRINT COMMAND .....	8
LESCRIPT.....	9
JAVA.....	9
PYTHON.....	10
<b>THEORY OF OPERATION .....</b>	<b>11</b>
<b>THE DIAGRAM.....</b>	<b>11</b>
<b>USER MODES.....</b>	<b>12</b>
<b>SYSTEM TABS.....</b>	<b>12</b>
RUN TAB .....	12
<i>Run Tab- No Options.....</i>	12
<i>Run Tab- UR Dashboard Connection Only.....</i>	13
<i>Run - UR Dashboard and Command Connection.....</i>	14
<i>Universal Robot Option Controls.....</i>	15
<i>UR Grinding Option Controls.....</i>	16
<i>Run - UR Dashboard and Command Connection plus LMI Gocator .....</i>	16
<i>Robot Jogging in LLeonard.....</i>	17
CODE TAB .....	19
<i>Code   Positions.....</i>	19
<i>Code   Variables.....</i>	20
<i>Code   Java.....</i>	21
<i>Code   Python.....</i>	21
<i>Code   Manual.....</i>	21
<i>Code   RevHist.....</i>	22
SETUP TAB.....	23
<i>Setup   Devices .....</i>	23
Device Types.....	25
Connect/Disconnect Execution.....	25
Setup   Displays.....	25
<i>Setup   Tools .....</i>	26
<i>Setup   Robot .....</i>	27
<i>Setup   General .....</i>	28
<i>Setup   License .....</i>	29
LOGS TAB .....	29
<b>APPENDIX 1: LINE BY LINE EXECUTION COMMANDS.....</b>	<b>30</b>
STANDARD LIBRARY, ALL LANGUAGES.....	30
<i>General User Output .....</i>	31

# LLeonard User Manual

---

lePrint(string message).....	31
leShowConsole(bool show?) .....	31
leClearConsole() .....	31
lePrompt(string message).....	31
leLogInfo(string message) .....	31
le.LogError(string message) .....	31
<i>Flow Control</i> .....	31
labelName:.....	32
jump(string labelName).....	32
jumpif(bool condition, string labelName) .....	32
Jump_gt_zero(string varName, string label).....	32
call(string labelName) .....	32
return() .....	32
<b>EXAMPLE RECIPES .....</b>	<b>37</b>
REMOVE CURRENT TOOL.....	37
INSTALL A TOOL.....	37
INTEGRATED EXAMPLE.....	38
COMPUTED CONCENTRIC CIRCLES.....	39
LOTS OF GRINDS.....	39

## Overview

Welcome! LLeonard is a work cell control system that maintains communication with all the devices in your industrial work cell and allows you to orchestrate their coordinated operation—just like a good orchestra conductor.

LLeonard allows you to use write programs in custom LLEScript, Java, or Python to write control software and subroutines. The use of Java and Python opens the potential to use millions of lines of pre-existing code, as well as providing you with all the features of these rich programming environments.

LLEScript is simpler and basic and is often more than enough for simple work cell coordination and data collection and is less intimidating to those who may not already have familiarity with Java or Python.

### An aside: LLeonard File Structure

LLeonard is aware of a “LeonardRoot” directory where it is installed. LLeonard expects to be running out of LeonardRoot/LLeonard/bin

For you this could be anywhere. Many customers use c:\LLeonard.

Beneath this root, several directories are maintained:

- Code: Where the programs you create will be stored by default
  - Examples: Many LLeonard example programs
  - Lib: Python and Java code libraries including cusom add-on by Lecky Engineering
- Config: Default location for all of the Devices files \*.ldev
- Data: A good default directory for input and output data files
- DB: LLeonard database for Displays, Positions, Tools, and Variables. These are stored as simple XML.
- LLeonard/bin: The location of the LLeonard executables and support files.
- Logs: Where the logfiles are created.
- 3rdParty: Where example programs for 3<sup>rd</sup> party devices such as the Universal Robot code and code for the LMI Gocator.

## All About Leonard Devices

LLeonard maintains a list of devices that it communicates with. Devices are managed under the **Setup | Devices** tab.

# LLeonard User Manual

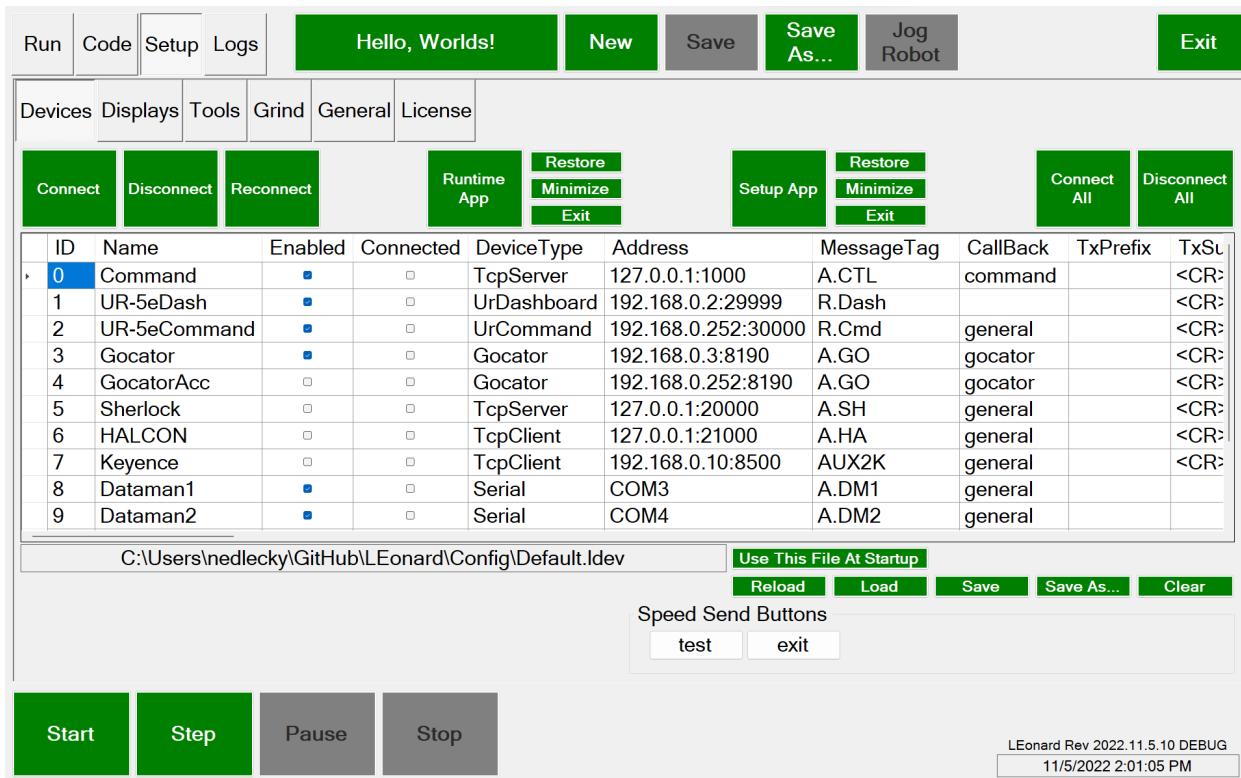


Figure 1 LLeonard Setup | Devices Screen

Each device has many setup parameters that control how LLeonard accesses and communicates with the device.

The device parameters are explained in more detail in [Setup | Devices](#). It is sufficient to know the following:

- 1) The list of connected devices is stored in a device file. You can have many of these, but most users will need only one.
- 2) Each device has a unique ID number and a unique name.
- 3) Devices can be enabled or disabled... this affects which devices get connected by the **Connect All** button.
- 4) There are several types of devices:
  - a. TCP Server
  - b. TCP Client
  - c. Serial
  - d. Null
  - e. Specialty
    - i. Universal Robot Dashboard
    - ii. Universal Robot Command
    - iii. LMI Gocator Interface
- 5) Each device sets up a **CallBack**, which is code to handle messages from the device whenever they arrive.

The “general” callback is most common, and is one of LLeonard’s key features and tricks!

More information on the specifics of the Devices features of LLeonard are available in [Setup | Devices](#).

## The LLeonardStatement and LLeonardMessage

LLeonard relies heavily on the use of a **LLeonardStatement**.

When a **LLeonardStatement** is analyzed, it is expected to look like one of the following. These are checked in sequence and the first match is handled and the other options untested.

- 1) filename.js Execute an entire Java file
- 2) filename.py Execute an entire Python file
- 3) LE:script Execute any LScript statement
- 4) JE:script Execute any Java statement
- 5) PE:script Execute any Python statement
- 6) SET varName value. Identical to varName=value
- 7) GET varName. LLeonard responds with the current value of variable 'varName'
- 8) varName = value. LLeonard stores 'value' into the variable named 'varName' for LScript, Java, and Python

As you can see, external devices can ask LLeonard to do very complicated or special things. This is what makes it possible to use LLeonard to automate so many work cell scenarios.

A **LLeonardMessage** is one or more **LLeonardStatements**.

Single: LLeonardMessage<TERM>

Double: LLeonardMessage<SEP> LLeonardMessage<TERM>

More LeonardMessages can be added on.

The advantage of a multiple statement message is that all of the messages will be executed in rapid sequence as a unit.

## LLeonard Displays

LLeonard was originally designed to work well on standard 10" industrial touch screen tablets. Buttons are large and easy to hit, and special file open and save dialogs, as well as numeric entry, are handled in a touchscreen-friendly way.

However, LLeonard can be used successfully on larger monitors. It is also designed to be resizable and to be usable even on big monitors for setup and other purposes.

LLeonard provides a standard database of various screen sizes, and you can add your own.

# LLeonard User Manual

---

More information on LLeonard displays is provided in the [Setup | Displays](#) section of the manual

## LLeonard Programming

LLeonard provides three common programming languages to enable devices to be controlled and sequenced. So how do we do our Hello, World!

Well, let's first talk about printing to a console screen.

### An Aside: The Print Command

LLeonard provides a unified print console for debugging. Show or hide the console with the F12 key.

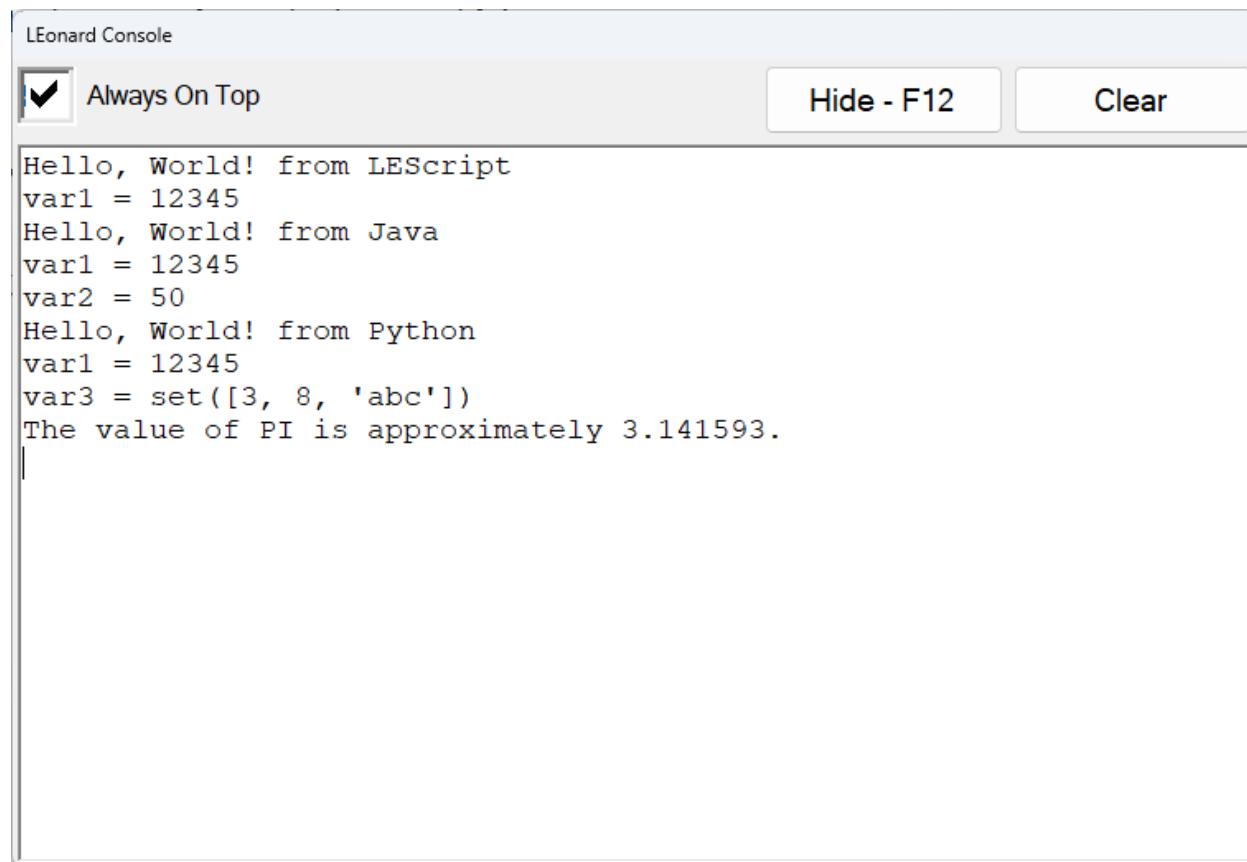


Figure 2 LLeonard Console Window

Pressing F12 repeatedly shows and hides the console. It is always receiving and buffering lePrint command data.

The command `leShowConsole(true|false)` will also show or hide the console and works in any language.

To send data to the print console, use lePrint(message) from any language.

OK, now we can try that Hello, World!

## LEScript

LEScript is a simple scripting language that allows “programming-free” interaction with devices. Most typical work cells can be programmed in LScript. If you need to compute values or get into deeper calculations or manage data structures, LLeonard provides control using either Java or Python, discussed below.

```
# Hello, World! In LScript
UsingLEScript()
lePrint(Hello, World!)
value = 13.25
lePrint(value = {value})

Console:
Hello, World!
value = 13.25
```

An exhaustive list of all the LScript commands is located in **Appendix 1: LScript Commands**

## Java

LLeonard provides a Java interface with full ECMA 5.1 compliance based on Jint.

Specifications:

Jint Version:	2.11.58
Author:	Sebastian Ros
License:	<a href="https://raw.githubusercontent.com/sebastienros/jint/master/LICENSE.txt">https://raw.githubusercontent.com/sebastienros/jint/master/LICENSE.txt</a>
Date Published:	11/27/2017
Project URL:	<a href="https://github.com/sebastienros/jint">https://github.com/sebastienros/jint</a>

There are several ways that Java execution can be triggered in LLeonard

1. In a LLeonard program, commands are interpreted as LScript until UsingJava() is encountered. Subsequent lines will be executed using Jint.
2. The LScript command exec\_java(filename) will run a Java \*.js file as specified.
3. An external device that is using the general callback can return a Java request:
  - a. “Filename.js” The specified file will be executed by LLeonard::Jint
  - b. JE:javascript The specified Java commands will be executed by LLeonard

4. The test area in the **Code | Java** tab. This provides a “sandbox” where Java programs can be created, edited, saved, and retrieved.

```
# Hello, World! In Java
UseJava()
lePrint('Hello, World!')
value = 13.25 * 8.1
lePrint('value = ' + value)
```

```
Console:
Hello, World!
value = 107.32499999999999
```

## Python

LLeonard provides the open-source Iron Python implementation originally developed by Microsoft. This supports Python 2.7 environment and includes the entire standard library. Iron Python support for Python 3 is still in Beta and will be made available upon request if we feel it is stable!

Specification:

IronPython Version: 2.7.12  
Author: Iron Python contributors, Microsoft  
License: <https://licenses.nuget.org/Apache-2.0>  
Date Published: 1/21/2022  
Project URL: <https://ironpython.net/>

There are several ways that Java execution can be triggered in LLeonard

5. In a LLeonard program, commands are interpreted as LScript until UsingPython() is encountered. Subsequent lines will be executed in Python.
6. The LScript command exec\_python(filename) will run a Python \*.py file as specified.
7. An external device that is using the general callback can return a Python request:
  - a. Send “Filename.py” The specified file will be loaded and executed in Python.
  - b. Send “JE:javascript” The specified Python commands will be executed in Python.
8. The test area in the **Code | Python** tab. This provides a “sandbox” where Java programs can be created, edited, saved, and retrieved.

```
# Hello, World! In Python
UsePython()
lePrint('Hello, World!')
value = 13.25 * 8.1
lePrint(str(value))
```

```
import math
var3 = {8, 'abc', 3}
lePrint('var3 = ' + str(var3))
lePrint('The value of PI is approximately {0:.6f}'.format(math.pi))

Console:
Hello, World!
107.325
var3 = set([3, 8, 'abc'])
The value of PI is approximately 3.141593.
```

## Theory of Operation

LLeonard orchestrates the interoperation of a number of devices using a simple program script that can be written in LScript, Java, Python, or a combination of all three.

All three languages are provided to make things easier for you! Use what you like.

LLeonard executes the main program **one line at a time**. This permits monitoring, error recovery, and single-stepping.

This may seem odd to those with a formal Computer Science background, but for reasons of safety and error-recovery, each line of a work cell sequence needs to be handled all on its own.

This also allows single stepping through work cell operations, critical during debug and testing.

Large functions that are safe and which don't make robots fly around can be built into Java or Python procedures that you load at the start of the LLeonard program. This way, you can go fast when you need to, and when the error conditions don't create unsafe machine situations.

Since LLeonard executes Java and Python line-at-a-time in the main run window, creating functions and classes in Java or Python must be done in a separate stand-alone file and loaded in your main program. LLeonard provides Java and Python sandboxes for writing and testing standalone code in the **Code | Java** and **Code | Python** tabs.

## The Diagram

## User Modes

There are three user modes. The user mode is selected using the **User** field in the upper-right corner of the **Run** tab.

1. **Operator** mode only permits loading and running existing programs from the **Run** tab.  
There is no access to the **Code** or **Setup** tabs.
2. **Editor** mode allows access to the **Code** tab to permit editing, but **Setup** is suppressed.
3. **Engineering** mode provides full access to all functions including **Setup**.

Entering Operator or Engineering mode requires a fixed password. By default, these are 9 and 99, respectively.

## System Tabs

In the main LLeonard screen, there are four Operation Tabs: **Run**, **Code**, **Setup**, and **Logs**. These screens are described below.

Many functions can be manually activated with buttons or automatically activated with program commands. The convention in this manual is the used boldface for buttons and Tabs and italics for program commands, as in **This Is a Button** and *this\_is\_a\_code\_function(param1)*.

The four main tabs will be described below. Here are some quick links if you want to jump!:

[Run Tab](#)

[Code Tab](#)

[Setup Tab](#)

[Logs Tab](#)

## Run Tab

The **Run** tab is where the bulk of program execution will typically be observed.

Annunciators and control buttons appear on the Run Tab depending on what options are included and what kinds of devices you are connected to.

## Run Tab- No Options

For example, the raw LLeonard screen looks like this, only showing the main program and providing access to Start, Stop, etc.

# LLeonard User Manual

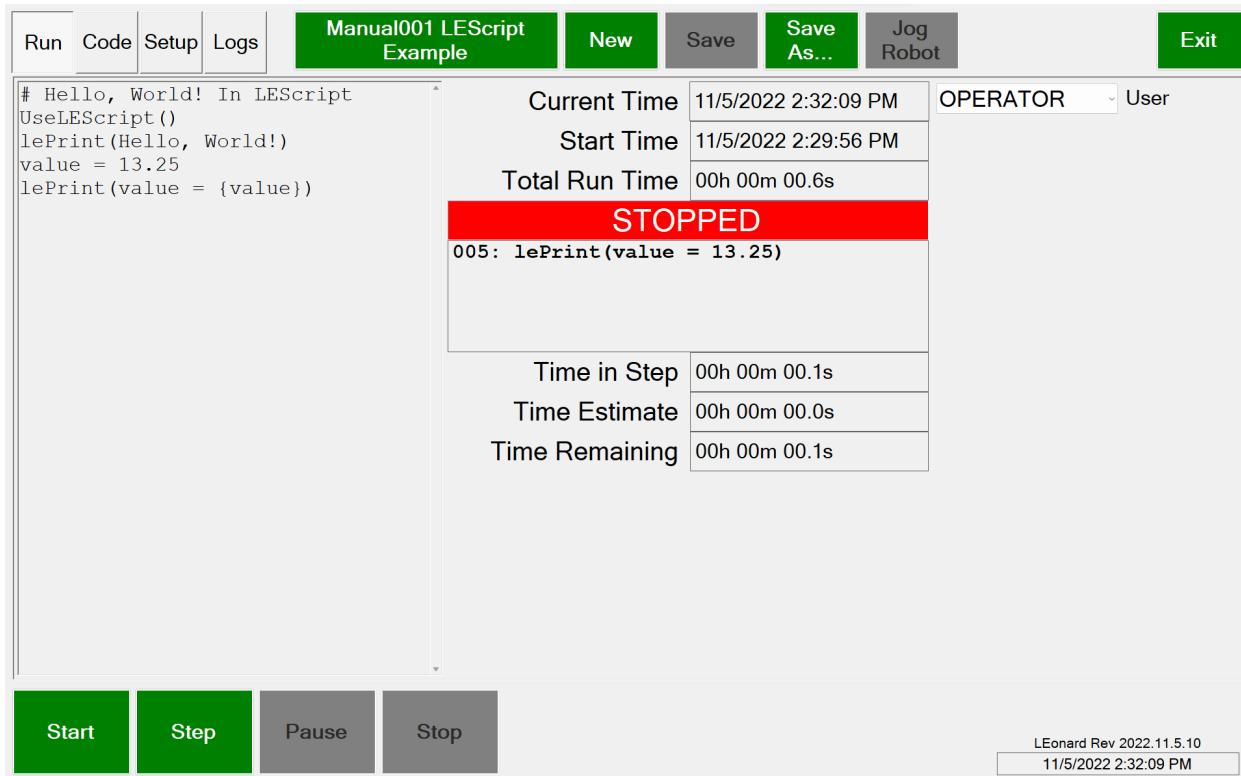


Figure 3 LLeonard Run Screen with No Options Installed

A program is loaded using the **Program Name** button next to the **Logs** tab.

Program file operations in addition to **Load** are **New**, **Save**, **Save As**.

Set the User in the **User Dropdown**.

## Run Tab- UR Dashboard Connection Only

Connecting a UR Robot Dashboard makes the robot control annunciators visible:

Devices	Displays	Tools	Grind	General	License								
Connect	Disconnect	Reconnect	Runtime App		Restore	Minimize	Exit	Setup App	Restore	Minimize	Exit	Connect All	Disconnect All
ID	Name	Enabled	Connected	DeviceType	Address	MessageTag	CallBack	TxPrefix	TxLength	RxLength	RxData	Robot IP	Robot Port
0	Command	<input checked="" type="checkbox"/>	<input type="checkbox"/>	TcpServer	127.0.0.1:1000	A.CTL	command	<<	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	192.168.0.2	299999
1	UR-5eDash	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	UrDashboard	192.168.0.2:29999	R.Dash							

Figure 4 LLeonard Connected to UR Dashboard

# LLeonard User Manual

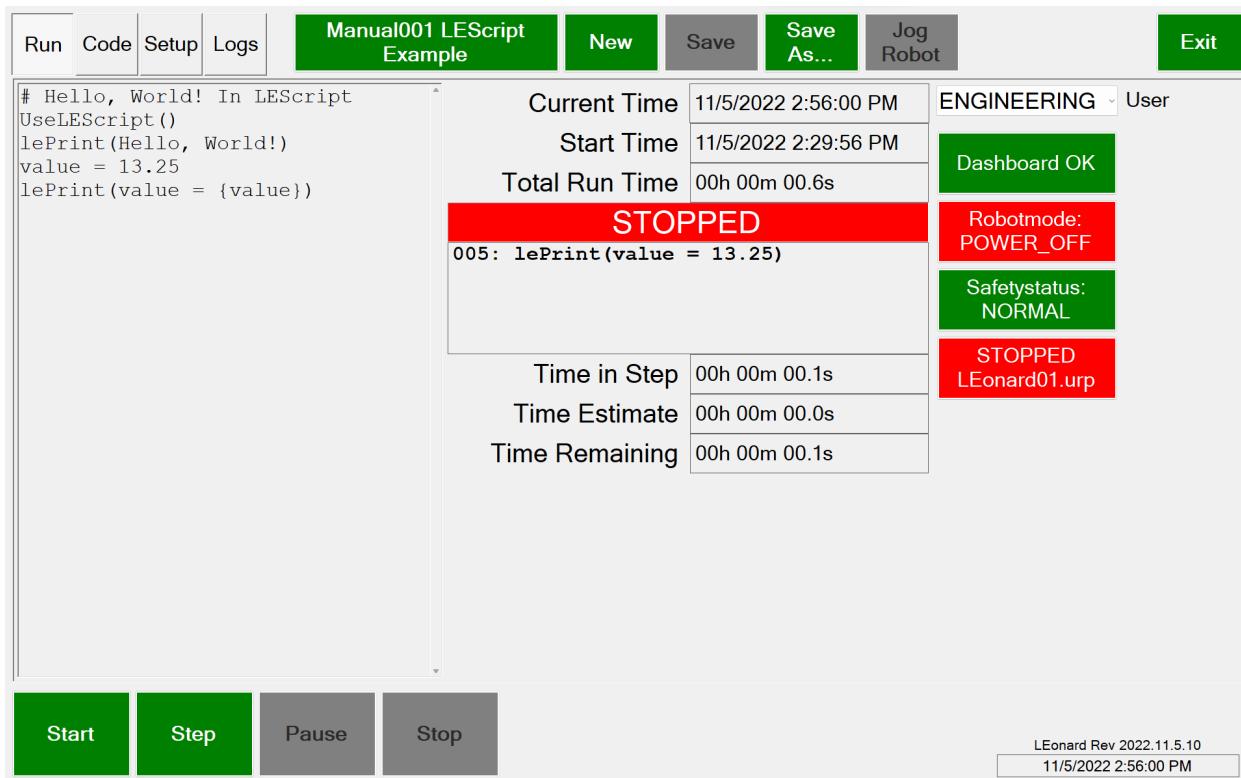


Figure 5 LLeonard Main Screen when Connected to UR Dashboard

**Dashboard OK:** Shows connection status to the Dashboard (this is a TCP client).

**Robotmode:** Press this to cycle from POWER\_OFF to BOOTING to IDLE to RUNNING

**Safetystatus:** Press this to clear robot stop conditions

**STOPPED/PLAYING:** Use this to toggle between a PolyScope program running or not on the robot. The program loaded is specified in the Device entry and is shown in this button.

Most of these options require that the UR is in Remote mode.

## Run - UR Dashboard and Command Connection

With a UR Command interface also connected, the full UR control system is available. This requires that a special PolyScope program be installed and running on the robot. This program is provided by Lecky Engineering and can be modified by the user if you need special functions. Contact Lecky Engineering for assistance if you are new to UR programming!

The images below assume the optional UR Grinding package is also licensed.

# LLeonard User Manual

Devices	Displays	Tools	Grind	General	License			
<b>Connect</b>	<b>Disconnect</b>	<b>Reconnect</b>	<b>Runtime App</b>	<b>Restore</b> <b>Minimize</b> <b>Exit</b>	<b>Setup App</b>	<b>Restore</b> <b>Minimize</b> <b>Exit</b>	<b>Connect All</b>	<b>Discover All</b>
ID	Name	Enabled	Connected	DeviceType	Address	MessageTag	CallBack	TxPrefix
0	Command	<input checked="" type="checkbox"/>	<input type="checkbox"/>	TcpServer	127.0.0.1:1000	A.CTL	command	<
1	UR-5eDash	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	UrDashboard	192.168.0.2:29999	R.Dash		<
2	UR-5eCommand	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	UrCommand	192.168.0.252:30000	R.Cmd	general	<

Figure 6 LLeonard Connected to UR Dashboard and UR Command

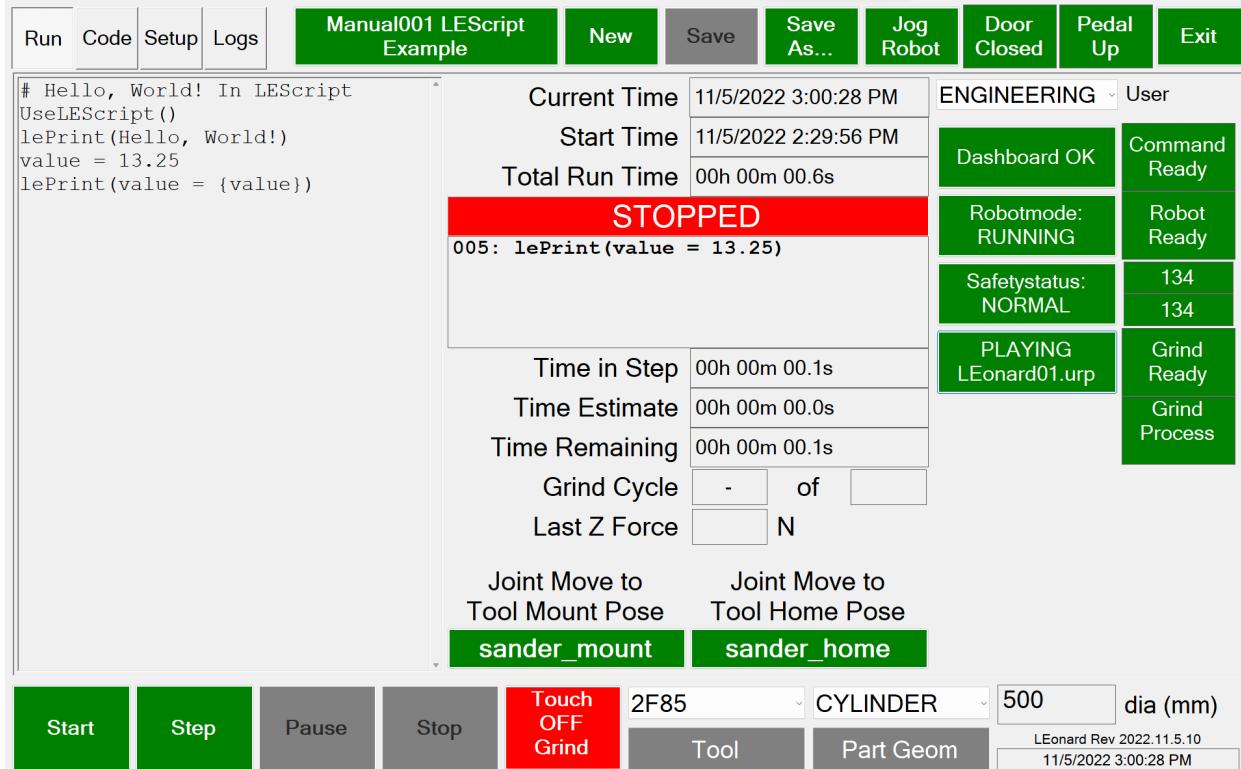


Figure 7 LLeonard Main Screen when Connected to UR Dashboard and UR Command

With the optional Grinding Package enabled we get 2 additional buttons:

**Grind Ready:** Has the current grinding process completed?

**Grind Process:** Are we leaving the tool in contact with the part and expecting the next grind command within a few seconds?

## Universal Robot Option Controls

**Command Ready:** Shows status of UR Command connection (this is a TCP server).

**Robot Ready:** Shows if the robot is currently executing a LLeonard command.

**Command Index Numbers:** Shows index number of last command sent and response ID of last response received. These should stay in lockstep and must be equal to allow the next command to be issued in the program.

Tools are selected in the **Tool Dropdown**. More information on Tools in in [Setup - Tools](#)

# LLeonard User Manual

Part geometry is specified in **Part Geometry Dropdown**: The UR Command program supports surfaces that are FLAT, CYLINDER, or SPHERE

Tools have mount and home positions that can be moved into with the **toolname\_mount** and **toolname\_home** buttons.

**Running a Program** behaves as expected:

**Start, Stop, Step, and Pause / Continue**

**Jog Robot** is used to jog or freedrive to a defect. Jogging is described on the next page.

## UR Grinding Option Controls

Set the grinding mode with the **Touch/Grind button** which cycles through **No contact, Touch Only, and Touch+Grind**

**Protective Stops:** These show up in (and may be cleared with) the SafetyStatus button  
**Door Status** is monitored (IO is configured in the **Setup Tab**). Door Open is treated like **Pause**.  
**Footswitch Status** is monitored (IO is configured in the **Setup Tab**).

## Run - UR Dashboard and Command Connection plus LMI Gocator

Adding an LMI Gocator connection reveals the final (for now!) custom buttons:

Run - UR Dashboard and Command Connection plus LMI Gocator											
Devices		Displays		Tools		Grind		General		License	
Connect			Disconnect			Reconnect			Runtime App		
									<a href="#">Restore</a>	<a href="#">Minimize</a>	<a href="#">Exit</a>
									<a href="#">Setup App</a>	<a href="#">Restore</a>	<a href="#">Minimize</a>
									<a href="#">Exit</a>		
										<a href="#">Connect All</a>	<a href="#">Disconnect All</a>
ID	Name	Enabled	Connected	DeviceType	Address	MessageTag	CallBack	TxPrefix	TxSu		
0	Command	<input checked="" type="checkbox"/>	<input type="checkbox"/>	TcpServer	127.0.0.1:1000	A.CTL	command			<CR>	
1	UR-5eDash	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	UrDashboard	192.168.0.2:29999	R.Dash				<CR>	
2	UR-5eCommand	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	UrCommand	192.168.0.252:30000	R.Cmd	general			<CR>	
3	Gocator	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Gocator	192.168.0.3:8190	A.GO	gocator			<CR>	
4	GocatorAcc	<input type="checkbox"/>	<input type="checkbox"/>	Gocator	192.168.0.252:8190	A.GO	gocator			<CR>	
5	Sherlock	<input type="checkbox"/>	<input type="checkbox"/>	TcpServer	127.0.0.1:20000	A.SH	general			<CR>	
6	HALCON	<input type="checkbox"/>	<input type="checkbox"/>	TcpClient	127.0.0.1:21000	A.HA	general			<CR>	
7	Keyence	<input type="checkbox"/>	<input type="checkbox"/>	TcpClient	192.168.0.10:8500	AUX2K	general			<CR>	
8	Dataman1	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Serial	COM3	A.DM1	general				
9	Dataman2	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Serial	COM4	A.DM2	general				

Figure 8 LLeonard Connected to UR Dashboard/Command and LMI Gocator

# LLeonard User Manual

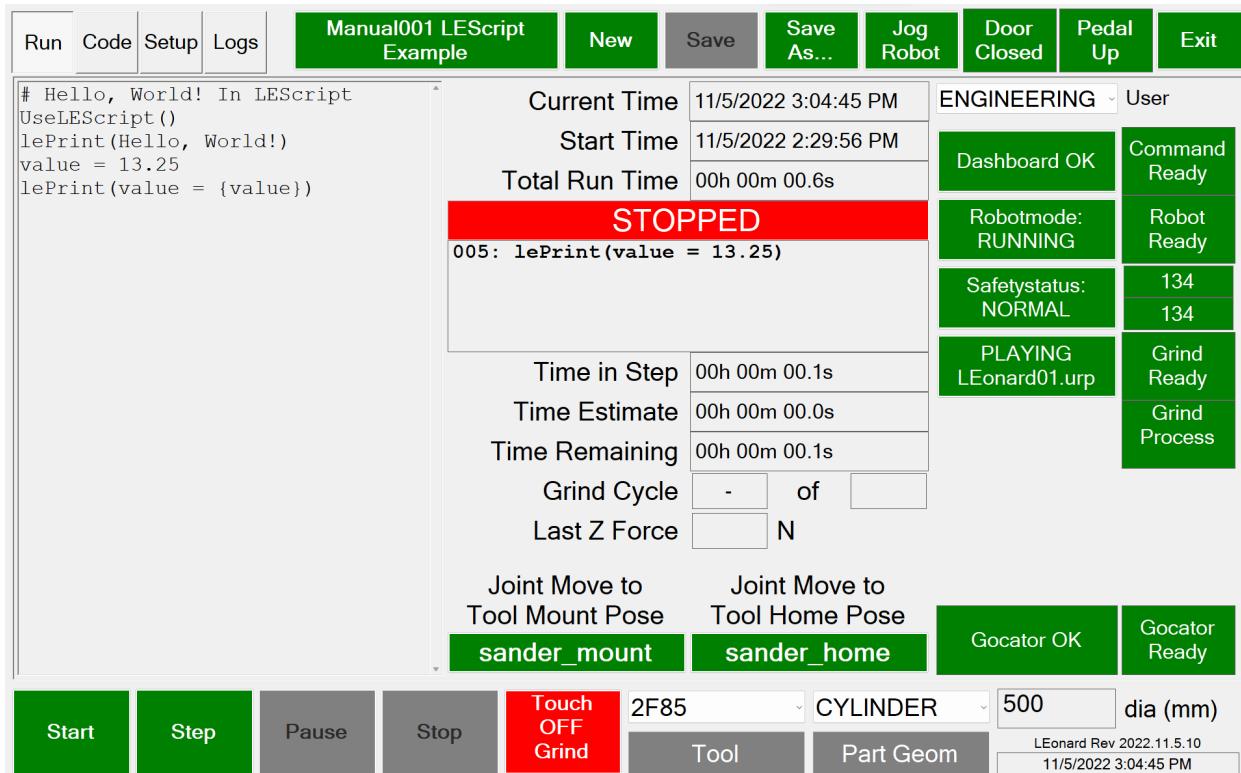


Figure 9 LLeonard Main Screen when Connected to UR Dashboard/Command and LMI Gocator

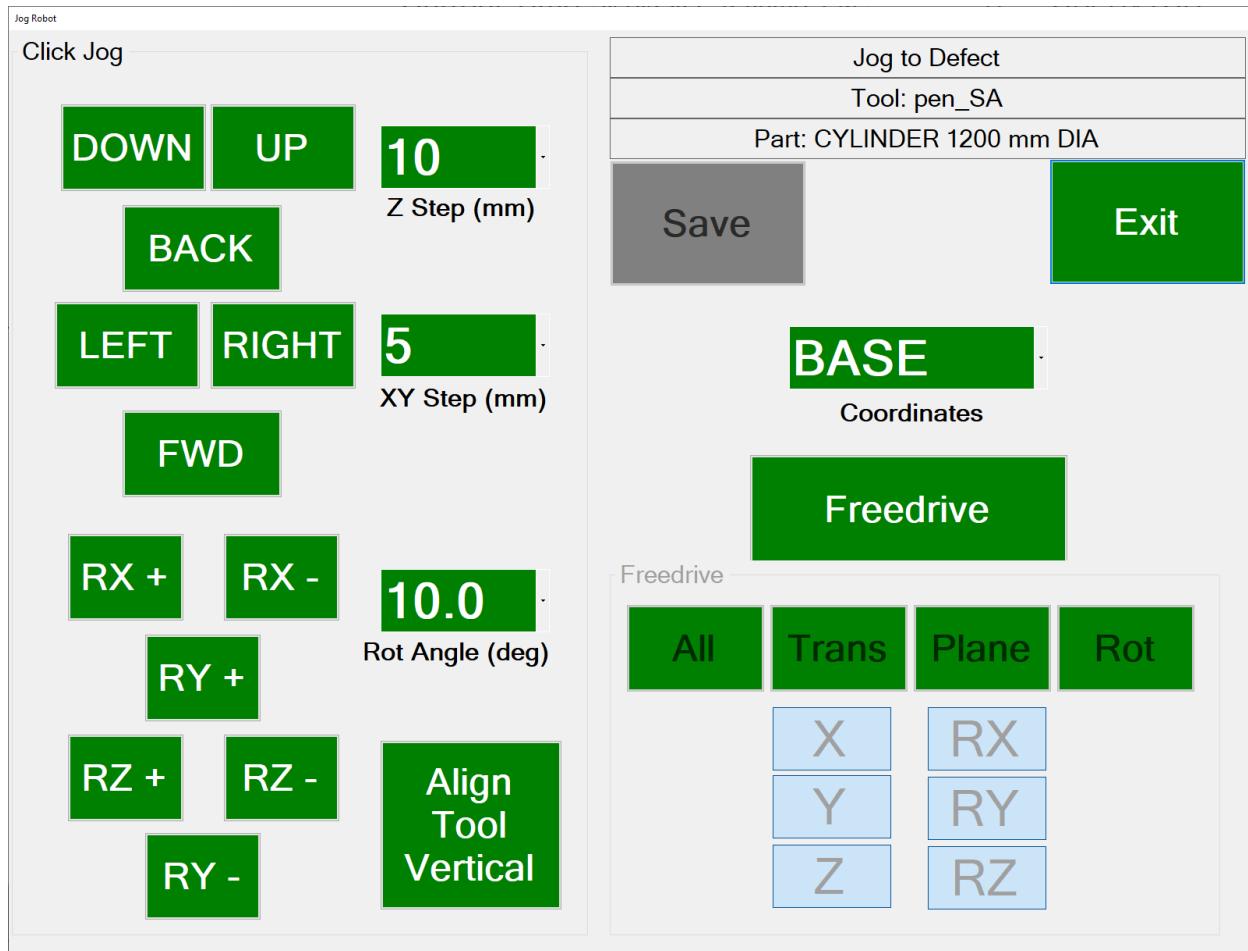
**Gocator OK:** Is the connection with the Gocator (a TCP client) healthy?

**Gocator Ready:** Goes red when the Gocator is processing.

## Robot Jogging in LLeonard

Jogging opens a separate screen. Jogging can be done in **BASE** or **TOOL** coordinates, or relative to a **PART**. The buttons move the robot by the specified increment in Z, XY, or rotation. Holding a button down (mouse) or double-tapping and holding (tablet) makes the move repeat.

# LEonard User Manual



When jogging in **PART** mode, if a cylindrical or spherical geometry is selected, the tool will rotate around the center of the part instead of around the tool tip. This can be convenient for manually jogging to a defect using the touch screen instead of freedrive

Freedrive is supported in a manner identical to on the UR pendant. The X, Y, X, RX, RY and RZ buttons may be used to enable or disable freedrive in any desired axis. All, Trans, Plane, and Rot select pre-defined subsets of axes as on the UR.

Coordinate systems may be changed during freedrive and the tool will allow motion relative to the world, the tool, or the center of the part if part geometry is cylinder or sphere.

Press the Freedrive button again to turn freedrive mode off. Saving or exiting the dialog will also turn off freedrive.

The **Freedrive footswitch** puts all 6 axes in freedrive whether this jog dialog is open or not.

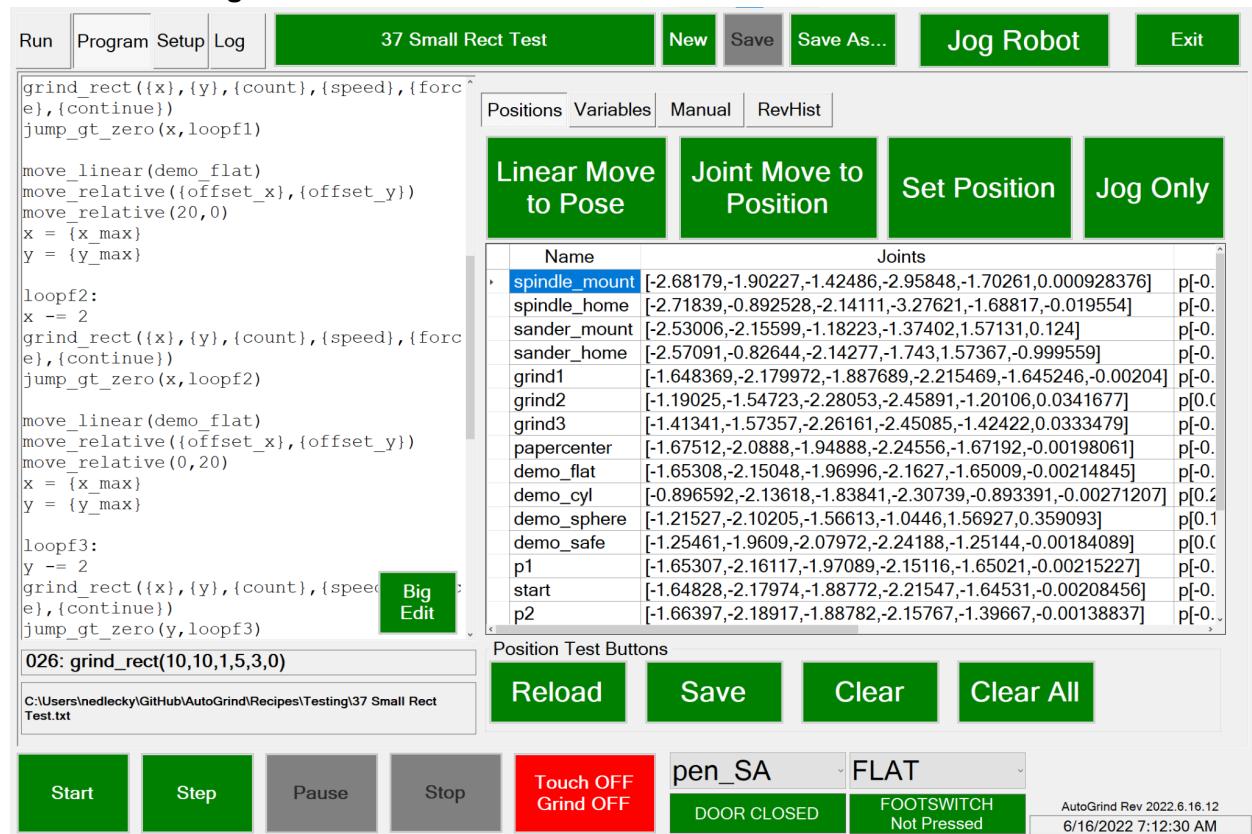
# LEonard User Manual

## Code Tab

The **Code Tab** has six subpages: **Positions**, for teaching and manually moving to fixed positions, **Variables**, for monitoring or changing LEonard variables, **Java**, for writing and testing Java programs, **Python**, for writing and testing Python programs, **Manual**, to provide access to documentation, and **RevHist**, which displays what has been added in each version of the software. The current software version is always displayed in the lower right of the screen.

### Code | Positions

Below is the **Program Tab** when the Positions Subtab is selected.



Positions can be saved manually (**Set Position**) or from the recipe with `save_position(name)`.

You can manually move to Positions in Joint (**Joint Move To Position**) or Linear (**Linear Move To Pose**) paths. These can also be executed from a recipe with `move_linear(position)` or `move_joint(position)`.

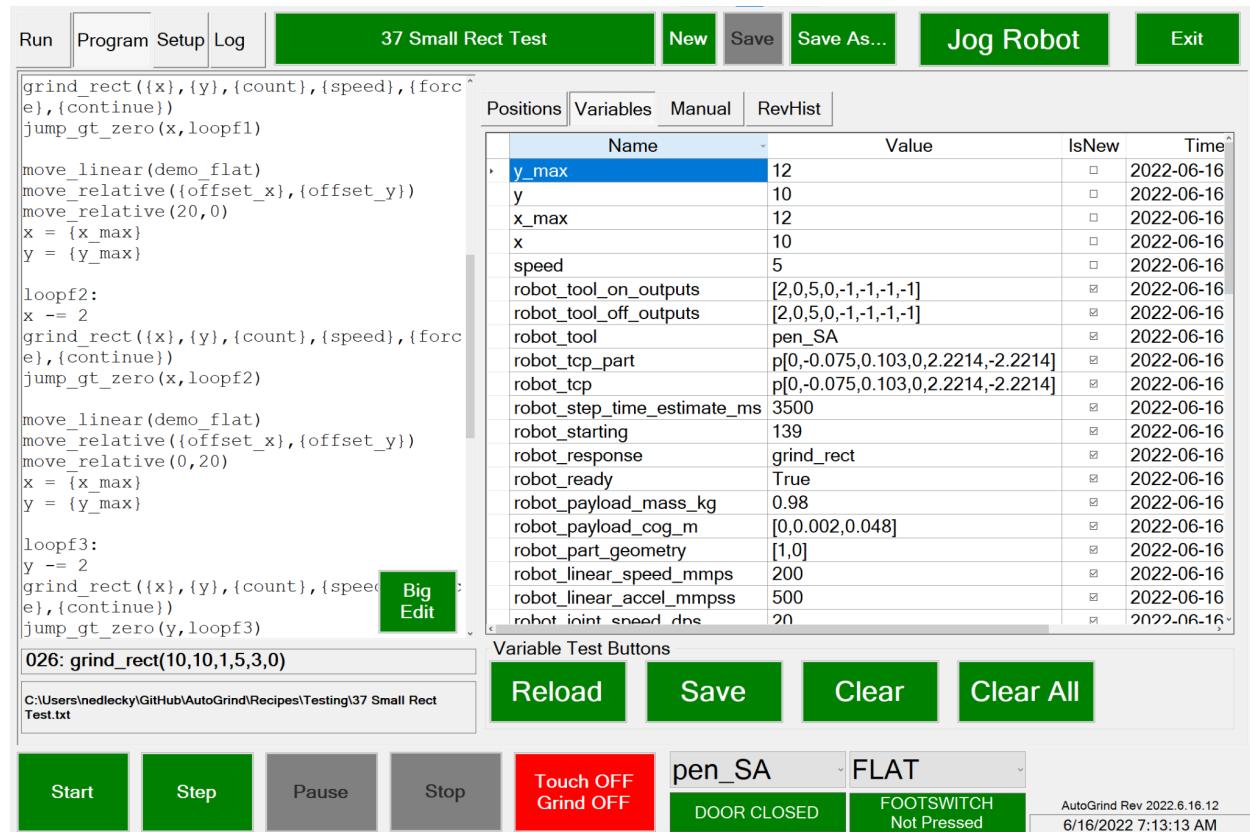
Jogging is used here for setting or updating named positions or just for moving the robot. This uses the standard Jog screen.

**Big Edit** opens up a full-screen editor to make editing complex recipes easier.

# LEonard User Manual

## Code | Variables

Below is the **Program Tab** when the **Variables Subtab** is selected..



This tab shows all of the local variables maintained in LEonard for internal, system, and user purposes. They can be edited here as well.

System variables will not be erased by the **Clear** button, or by the recipe `clear()` command.

The **TimeStamp** shows when the variable was last written.

**IsNew** indicates whether the variable has ever been examined by the program since the last write.

The variables may be saved or reloaded from Recipes/Variables.xml with the **Save** and **Reload** buttons. Variables are automatically saved on program exit and reloaded when the program starts.

# LEonard User Manual

## Code | Java

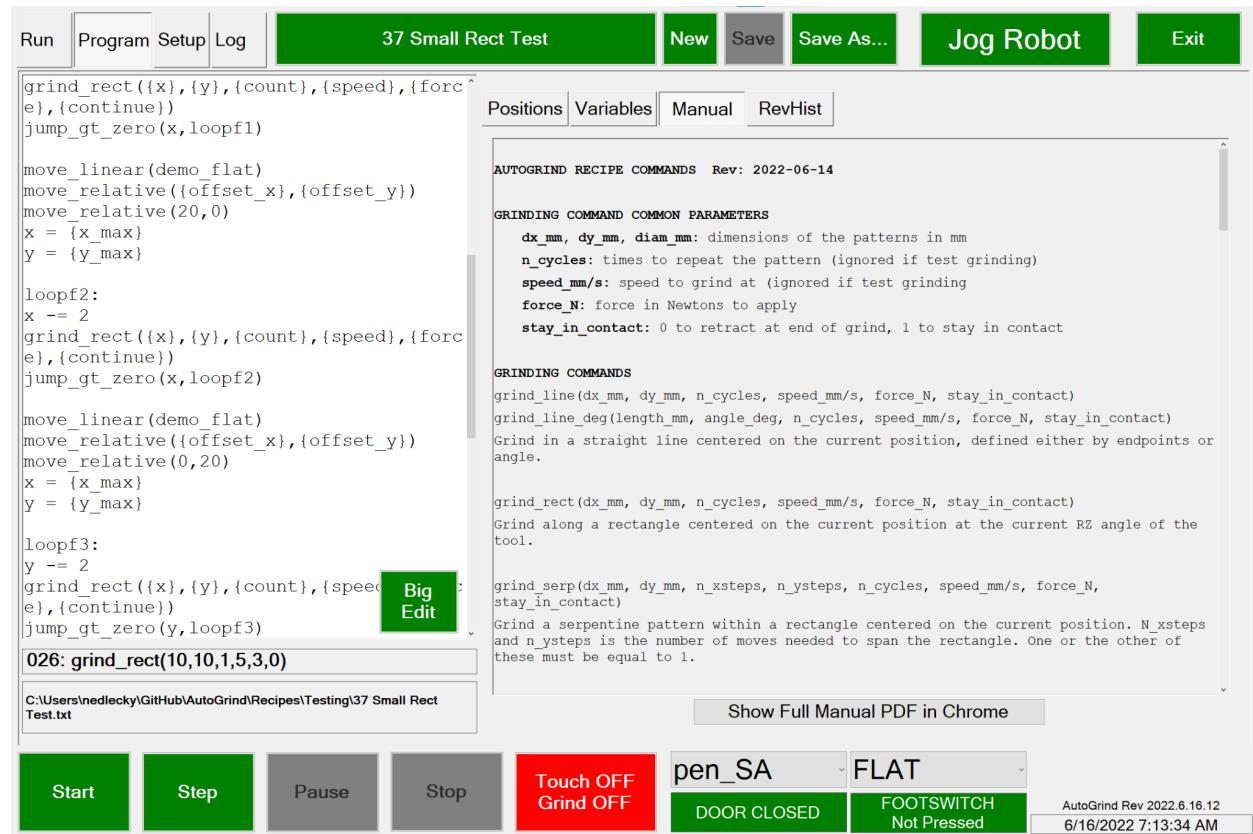
More here TODO

## Code | Python

More here TODO

## Code | Manual

Below is the **Program Tab** when the **Manual Subtab** is selected..



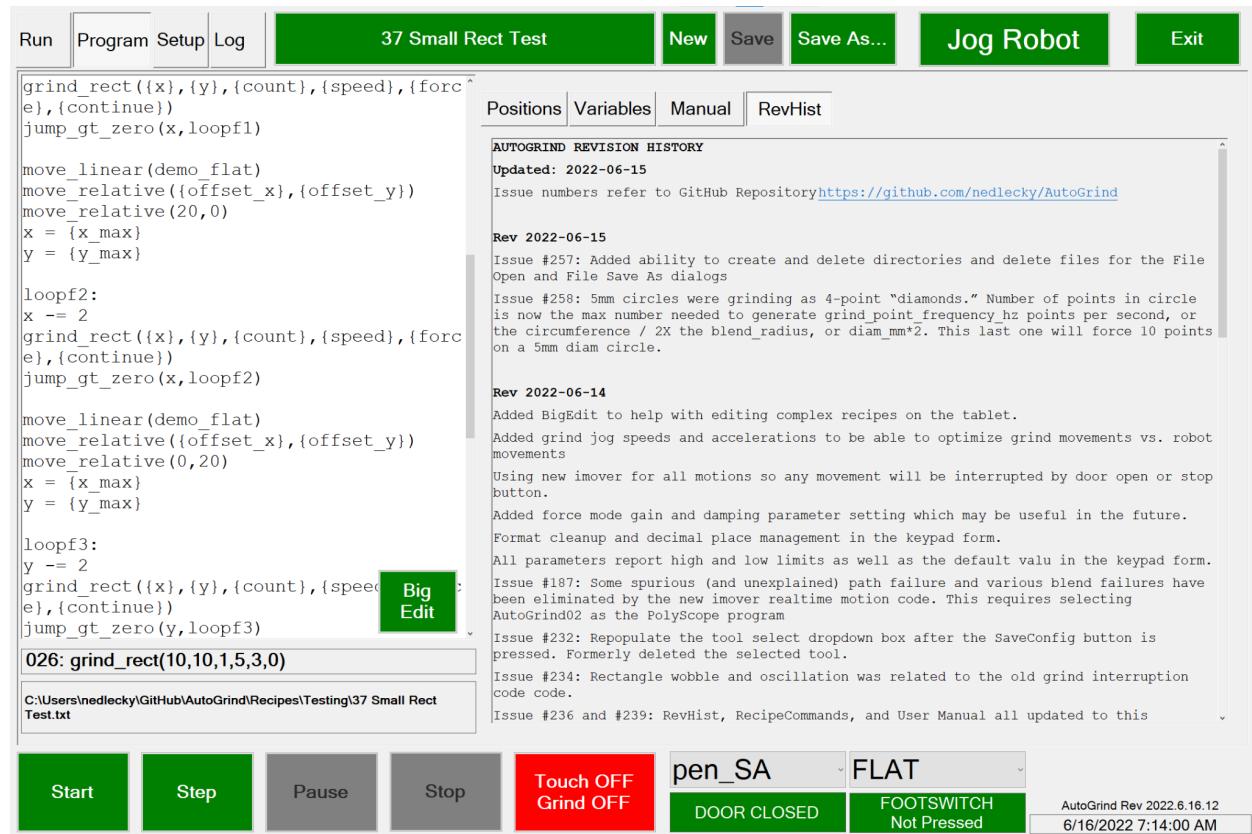
This tab displays the recipe commands to help you remember what each command does and how it is called..

The **Show Full Manual PDF in Chrome** button pulls up a PDF of the manual you are looking at in Chrome. It is assumed that Chrome is installed on the system.

# LLeonard User Manual

## Code | RevHist

Below is the **Program Tab** when the **RevHist Subtab** is selected..



This tab displays the changes incorporated in the current (and prior) versions of the LLeonard program. The issues addressed are described in more detail in the LLeonard GitHub repository.

<https://github.com/nedlecky/LLeonard>

# LLeonard User Manual

## Setup Tab

The **Setup Tab** is where all system configuration takes place.



## Setup | Devices

The LLeonard device list is a datafile create in the LLeonardRoot/Data directory. You can have several device files and load different ones for different testing or operational situations.

LLeonard devices have the following parameters:

Field Name	Type	Description
ID	Integer	A unique ID assigned to the device
Name	String	A name to help you remember what the device does
Enabled	Boolean	Specifies whether the device should be automatically connected by the Connect All button. Devices are also automatically connected when the device file is loaded if <b>Auto Connect On Load</b> is enabled in <b>Setup   General</b>
Connected	Boolean	A boolean value
DeviceType	String	One of several classes of devices, discussed below
Address	String	Either IP:Port or COMn

# LLeonard User Manual

---

MessageTag	String	A string to be prepended to log messages to help identify messages from a particular device
CallBack	String	A callback function, described below.
TxPrefix <PREFIX>	String	A prefix of characters to be sent before each transmission. May include <CR> <LF>, and <CRLF>
TxSuffix <SUFFIX>	String	Characters sent at the end of each transmission. May include <CR> <LF>, and <CRLF>
RxTerminator <TERM>	String	Characters to be waited for to signify end of received message. May include <CR> <LF>, and <CRLF>
RxSeparator <SEP>	String	LLeonard will parse (and execute atomically) multiple commands using command<SEP>command<TERM>
OnConnectExec	String	When the external device connects, any LLeonardMessage specified here will be executed.
OnDisconnectExec	String	When LLeonard initiates a disconnect, any LLeonardMessage specified here will be executed.
RuntimeAutostart	Boolean	If true, Runtime Program will be started before connection is attempted. This can be used to start a background server needed by the device for operation.
RuntimeWorkingDirectory	String	The Runtime Program will be executed from this directory
RuntimeFilename	String	Specifies the filename of the Runtime Program
RuntimeArguments	String	Specifies arguments for the Runtime Program
SetupWorkingDirectory	String	Some devices have a Setup Program used to configure them, They can be specified here for directory, filename, and arguments
SetupFilename	String	Filename of the Setup Program
SetupArguments	String	Arguments for the Setup Program
SpeedSendButtons	String	If you'd like to be able to send simple commands for testing to the device, they may be entered here as command1 command2 command3 ... A button will be created for each string between vertical bars!
JobFile	String	If a device needs to load a specific program to run it can be specified here. This is currently used by UrDashboard and Gocator to start the specified programs at connect time.
Model	String	If a device returns a model number, it will be entered here

# LLeonard User Manual

---

Serial	String	If a device returns a serial number, it will be entered here
Version	String	If a device returns a software version number, it will be entered here

## Device Types

LLeonard device types must be one of the following items:

1. TcpServer Set up a TCP Server on Address:Port and wait for a connection.
2. TcpClient Immediately connect to a device on Address:Port.
3. Serial Connect to a device with Address = COMn using serial protocol over either a hard serial or a USB serial connection
4. UrDashboard: Setup a TCP Client connection to a Universal Robot dashboard server.
5. UrCommand: Setup a TCP Server for a Universal Robot PolyScope program to attach to
6. Gocator: Setup a TYCP Client appropriate for handling commands with an LMI Gocator.
7. Null Connect to nothing... but perhaps use the other features of a device!

## Connect/Disconnect Execution

Just after a device connects or just before it is disconnected, LLeonard can perform an operation. These operations are encoded in the OnConnectExec and OnDisconnectExec fields in the device.

Any LeonardMessage can be specified. Recall a multi-statement LLeonard message can be encoded as:

LeonardMessage = LeonardStatement<SEP>LeonardStatement

## Setup | Displays

LLeonard maintains a database of standard display sizes in the Setup | Displays tab, shown below.

# LLeonard User Manual

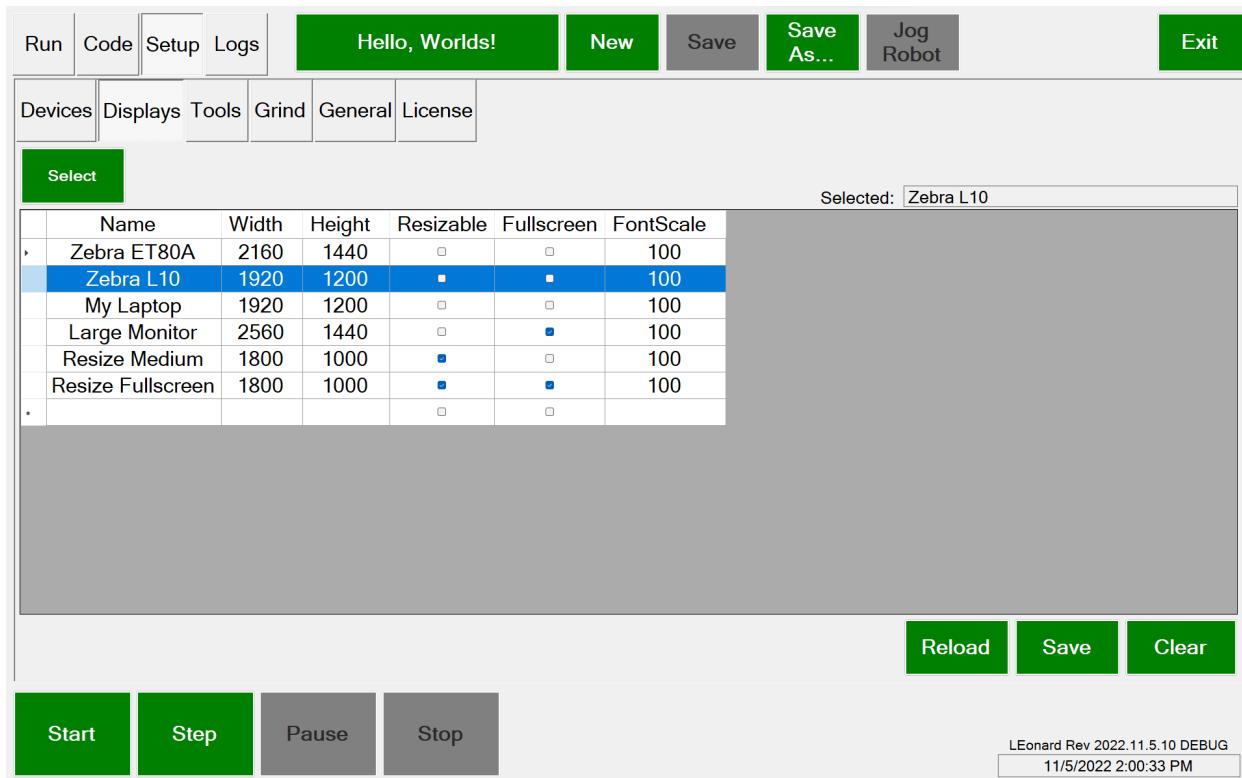


Figure 10 LLeonard Setup | Displays Screen

All LLeonard windows and dialogs are designed to be resizable. Fonts get larger or smaller as necessary. However you arrange the dialog to use it, that is how it will appear the next time you see it.

You can create your own display setting, complete with Width, Height, whether the window is fixed size or resizable, whether it should expand to full screen, and an optional additional Font Scale parameter.

The display database is automatically saved by LLeonard, but you can reload the old one if you've made a mistake. Clearing the displays will also offer an option to restore these defaults.

Whatever display is selected when LLeonard exits will be restored at application startup.

## Setup | Tools

Tools are defined in the Tool Table. Each contains the following information. These are saved in the Tools.xml file in LLeonardRoot/Recipes and are loaded and saved automatically.

1. **Tool TCP:** This is a copy of what we would teach for the tool on the UR including x, y, z offset and rx, ry, rz orientation. Teaching these is best done on the UR and then the values simply copied to the entry in LLeonard
2. **Mass and Center of Gravity:** Set these as you would on the UR. Accurate settings improves behavior when in freedrive mode.

3. **ToolOnOuts, ToolOffOuts:** This is a list of up to 4 digital IOs that need to be turned on or off to enable the tool. This is only done during a grind in **Touch ON Grind ON** mode. Examples: "1,1,3,1" implies that output 1 should be set to 1 and output 3 should be set to 1. "3,1" implies that output 3 should be set to 1
4. **CoolantOnOuts, CoolantOffOuts:** Similarly, these are digital output commands to be executed when grinding in **TouchOn Grind ON** mode.
5. **MountPosition:** This is a position recommended for installing/removing this tool. The system will use joint moves to approach the position with **Joint Move To Mount** or *move\_tool\_mount()*. This must be a position that has been defined in the **Positions Table**.
6. **HomePosition:** This is a position recommended for homefor this tool. The system will use joint moves to approach the position with **Joint Move To Home** or *move\_tool\_home()*. This must be a position that has been defined in the **Positions Table**.
7. **Tool Test, Tool Off and Cool Test, Cool Off:** These allow manually verifying the outputs for the currently selected tool.
8. **Set Footswitch Pressed Input:** This is defaulted to 7,1 meaning that input 7 goes high when the footswitch is pressed.
9. **Set Door Closed Input:** This is defaulted to 1,1 meaning that input 1 goes high when the door is closed.

## Setup | Robot

First, there are settings governing grind operations. These are saved in the Variables.xml file in LLeonardRoot/Recipes and are sent to the robot whenever the software starts. New values are saved automatically.

**Grind Trial Speed:** When not in **Touch On Grind On** mode, the grind patterns are limited to one cycle and are performed at this speed.

**Grind Acceleration:** Linear acceleration used during grinding

**Grind Max Blend Radius:** Maximum blend radius used during grinding. Recommended 2 mm

**Grind Touch Speed:** Speed robot advances toward part for touchoff. Recommended 5-10 mm/s

**Grind Touch Retract:** Distance robot retracts from part after touchoff.

**Grind Force Dwell Time:** How long robot waits after turning force-on to allow time for tool to settle against part

**Grind Max Wait Time:** Maximum time system will wait for the next grind command if a grind command ends with 1 (stay in contact with part)

**Grind Jog Speed:** Linear speed used for all grinding motions that are not in contact with the part other than the actual simulated grind which runs at **Grind Trial Speed**.

**Grind Jog Accel:** Linear acceleration used for all grinding motions that are not in contact with the part.

**Grind Point Frequency:** Used as a minimum frequency for points generated for circles and spirals.

**Force Damping:** May be useful for force mode tuning in the future. Calls the URScript function `force_mode_set_damping()` with a value between 0 and 1. The default is 0 and that is the only value that has been tested.

**Force Gain Scaling:** May be useful for force mode tuning in the future. Calls the URScript function `force_mode_set_gain_scaling()` with a value between 0 and 2. The default is 1.0 and that is the only value that has been tested.

Second, there are generally self-explanatory settings for speeds and accelerations used in jogging and non-grinding motion. These are saved in the `Variables.xml` file in `LLeonardRoot/Recipes` and are sent to the robot whenever the software starts. New values are saved automatically. **Restore Defaults** sets all to standard values used in all testing.

## Setup | General

1. **LLeonard Root Directory:** Location where subdirectories Recipes and Logs will be created. Tools, Variables, and Positions are also stored here in the Recipes subfolder.
2. **Robot Program To Load:** Specifies the program on the UR that the UR will load and run when this software starts. Starting in version 2022-06-07, this should be **AutoGrind/AutoGrind02.urp**
3. **Local IP for Server:** This should be the IP address of the port on the host computer that is connected to the UR
4. **UR Robot IP:** The IP address that the UR is set to
5. **Allow Running Offline:** Testing only... will allow recipes to run with no UR attached

# LLeonard User Manual

---

## Setup | License

LLeonard uses an encrypted license file that includes information about the CPU, Windows Version it is licensed on, and options that are included in the installation.

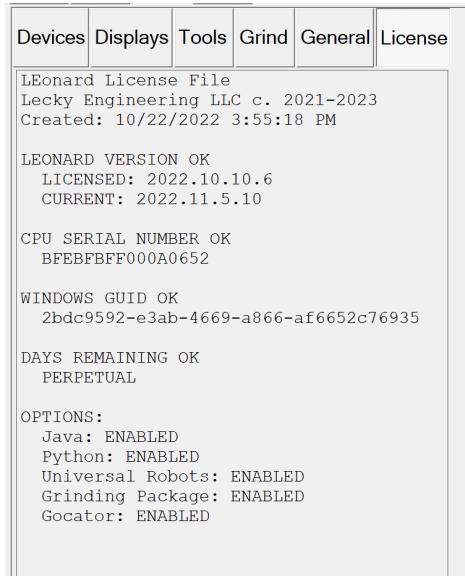


Figure 11 LLeonard Setup | License Viewer

Access your license on the **Setup | License** tab.  
Licenses can be perpetual or have a time limit for trial purposes.

A given license file will only work on one CPU and one installation of Windows. Contact Lecky Engineering if you have other needs. We're very flexible.

Current options are:

- Java
- Python
- Universal Robots Support Package
- Grinding Package for Universal Robots
- LMI Gocator Support

Contact Lecky Engineering to enable features and troubleshoot licensing issues on your system.

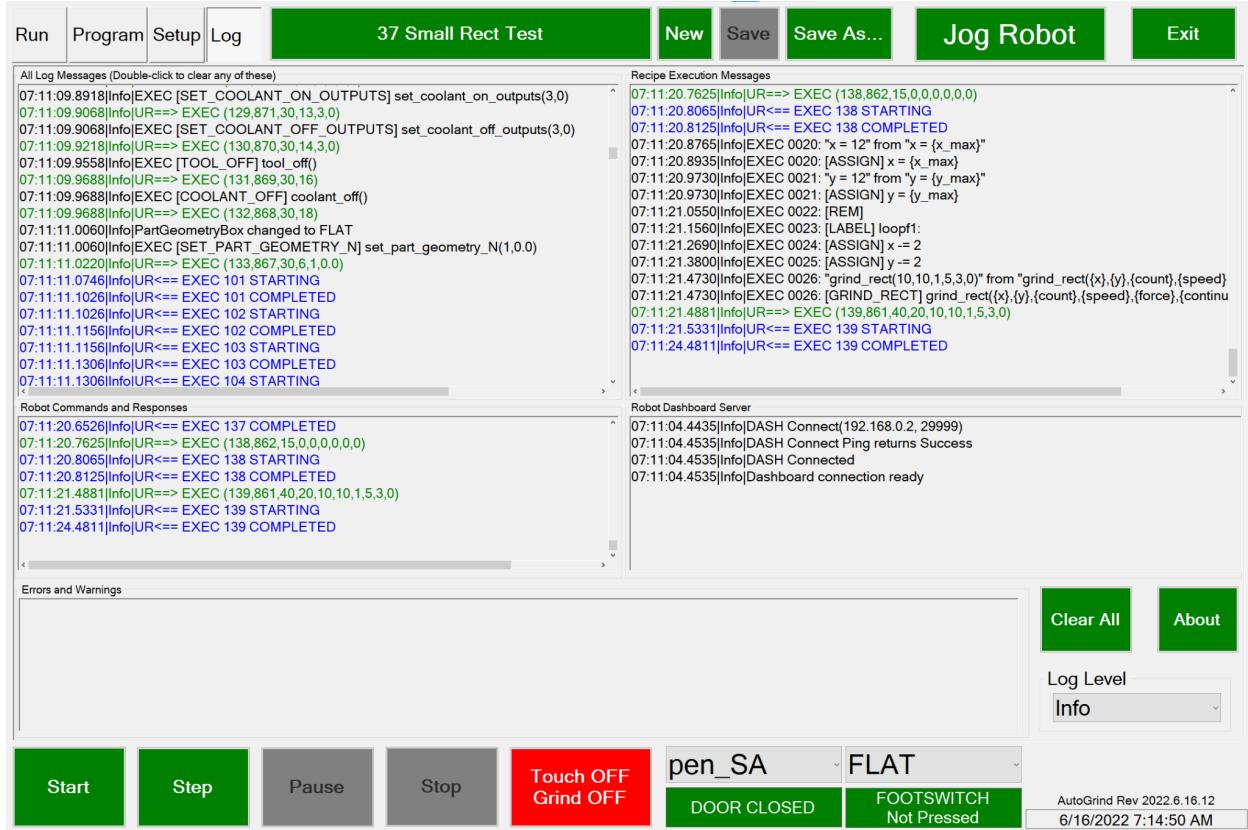
## Logs Tab

The Log Tab provides five windows where log messages are displayed. The level of detail in the messages is controlled by the Log Level setting:

- Error: only error messages are shown
- Warn: Error messages and Warnings are shown
- Info: All of the above, plus informational messages about execution. Default setting
- Debug: All of the above plus additional information that may be useful for debugging
- Trace: All of the above plus extremely verbose execution tracing

The All Log Messages box gets 100% of the generated messages. These messages are also written to log files in the LLeonardRoot/Logs directory, where up to forty 25MB files are archived. Information older than this 2GB total is automatically and silently deleted over time.

# LEonard User Manual



## TODO THIS NEEDS UPDATING

In addition, some messages are copied for clarity to other boxes. The boxes are labeled with their respective data: Recipe Execution Messages, Robot Commands and Responses, Robot Dashboard (Control and Monitoring) Messages, and Errors/Warnings.

Any of the boxes can be cleared by double-clicking on them. All boxes can be cleared with **Clear All**. All of the messages are appended to the log files and are archived as described above.

## Appendix 1: Line By Line Execution Commands

### Standard Library, All Languages

LEScript	Java	Python
x	x	x

## General User Output

```
lePrint(string message)
```

Prints message to the Console Window. For Java and Python, it is also sent to the console test areas in **Code | Java** or **Code | Python** as appropriate. The messages are also logged to the logfile as:

LEScript:	LE** message
Java:	JV** message
Python:	PY** message

```
leShowConsole( bool show?)
```

Hides or shows the Console Window. The console is always open and accumulating any lePrint messages from any language.

```
leClearConsole()
```

Clears all the text from the Console Window, just like the Clear button in the window itself.

```
lePrompt(string message)
```

Puts up a dialog box containing message and pauses execution until the operator presses Continue or Abort.

```
leLogInfo (string message)
```

Send the message to the logging system as Info. The message will appear in the **Logs** tab and in the logfile.

```
le.LogError (string message)
```

Send the message to the logging system as Error. The message will appear in the **Logs** tab in red in the Error buffer and will also be sent to the logfile.

## Flow Control

Flow control in line-by-line execution is implemented differently than it is in the Java or Python standards.

LLeonard follows a convention in which lines can be given label names and then jumped to or called. This is consistent with line-by-line execution and is why LLeonard is different in the main execution window.

Java and Python functions created in text files operate the way Java and Python always do!

# LEonard User Manual

---

labelName:

Defines a named label in the program. These are found prior to execution and can be used as targets for `jump(labelName)`, `jumpif(condition, labelName)`. LEScript provides the `jump_gt_zero(variable, labelName)` function since it does not have the ability to evaluate comparison conditions

`jump(string labelName)`

Causes execution to pass to the line containing `labelName`:

`jumpif(bool condition, string labelName)`

Performs a jump to the line containing `labelName`: if `condition` is true.

`Jump_gt_zero(string varName, string label)`

Only available in LEScript since comparisons aren't available there. Equivalent to `jumpif(varName > 0, labelName)`.

`call(string labelName)`

Causes execution to pass to the line containing `labelName`: Use `return` to return from the call. Call maintains a return stack (which is cleared when execution begins!) and can nest.

`return()`

Causes execution to go back to the line that most recently executed call.

This is a copy of the recipe commands document that is available from within the software.

**LEONARD RECIPE COMMANDS Rev: 2022-06-14**

## GRINDING COMMAND COMMON PARAMETERS

`dx_mm`, `dy_mm`, `diam_mm`: dimensions of the patterns in mm  
`n_cycles`: times to repeat the pattern (ignored if test grinding)  
`speed_mm/s`: speed to grind at (ignored if test grinding)  
`force_N`: force in Newtons to apply  
`stay_in_contact`: 0 to retract at end of grind, 1 to stay in contact

## GRINDING COMMANDS

`grind_line(dx_mm, dy_mm, n_cycles, speed_mm/s, force_N, stay_in_contact)`  
`grind_line_deg(length_mm, angle_deg, n_cycles, speed_mm/s, force_N, stay_in_contact)`

# LLeonard User Manual

---

Grind in a straight line centered on the current position, defined either by endpoints or angle.

```
grind_rect(dx_mm, dy_mm, n_cycles, speed_mm/s, force_N,  
stay_in_contact)
```

Grind along a rectangle centered on the current position at the current RZ angle of the tool.

```
grind_serp(dx_mm, dy_mm, n_xsteps, n_ysteps, n_cycles, speed_mm/s,  
force_N, stay_in_contact)
```

Grind a serpentine pattern within a rectangle centered on the current position. N\_xsteps and n\_ysteps is the number of moves needed to span the rectangle. One or the other of these must be equal to 1.

```
grind_poly(circle_diam_mm, n_sides, n_cycles, speed_mm/s, force_N,  
stay_in_contact)
```

Grind along a polygon of n\_sides inscribed in circle\_diam\_mm centered on the current position.

```
grind_circle(circle_diam_mm, n_cycles, speed_mm/s, force_N,  
stay_in_contact)
```

Grind along a circle centered on the current position.

```
grind_spiral(circle1_diam_mm, grind_circle2_diam_mm, n_spirals,  
n_cycles, speed_mm/s, force_N, stay_in_contact)
```

Grind along a variable diameter circle centered on the current position. The circle goes from the first diameter to the second in n\_spirals full revolutions.

# LEonard User Manual

---

```
grind_retract()
```

Ensure not in contact with the part. Happens automatically if a non-grind command is sent, if stop or pause is selected, or if grind\_max\_wait timer expires.

```
grind_contact_enable(0=Touch OFF,Grind OFF|1=Touch ON,Grind OFF|  
2=Touch ON,Grind ON)
```

Set the grinding mode programmatically as shown.

```
select_tool(tool_name)
```

Setup all of the necessary environment to be able to use tool\_name. No motion is performed. Future grinds and position moves will assume this tool is attached.

```
set_part_geometry(FLAT|CYLINDER|SPHERE, part_diam_mm)
```

Future grinds will assume the specified geometry.

The commands below provide a programmatic way to set the grinding parameters.

```
grind_touch_retract(touch_retract_mm)  
grind_touch_speed(touch_retract_speed_mm/s)  
grind_force_dwell(dwell_time_ms)  
grind_max_wait(max_time_before_retract_ms)  
grind_max_blend_radius(grind_blend_radius_mm)  
grind_trial_speed(trial_speed_mm/s)  
grind_linear_accel(accel_mm/s^2)  
grind_point_frequency(point_frequency_hz)  
grind_jog_speed(point_frequency_hz)  
grind_jog_accel(point_frequency_hz)  
grind_force_mode_damping(damping: 0.0 - 1.0)  
grind_force_mode_gain_scaling(scaling: 0.0 - 2.0)
```

## VARIABLE MANIPULATION

```
import(filename)
```

Open up a file and perform any variable assignment (name = value) lines found in it.

```
clear or clear()
```

Delete all variables except ones that are marked in the Variables Table as system variables. (Variables named robot\_\* and grind\_\* are automatically system variables.)

Update variables using any of these basic operations. Variables can be inserted in any command using the syntax {var\_name}.

```
var_name = 12.3      var_name = {other_var_name}  
var_name++          var_name--  
var_name -= 17.5     var_name += 18
```

# LLeonard User Manual

---

## FLOW CONTROL

label:

Labels a line in a program with a name. Label names are alphanumeric, case-sensitive, and may include the '\_' character.

jump(label)

Jumps to the line after the label specified.

jump\_gt\_zero(var\_name, label)

Jumps to the line after the label specified if the var\_name is numeric and greater than 0.

end or end()

Terminate execution of a recipe.

prompt(message)

Prompt the operator with a message and pause execution until the dialog is acknowledged.

sleep(seconds)

Pause execution for the specified time. Fractional seconds may be used.

assert(var\_name, value)

Testing support. Checks to see if var\_name==value and generates an error message if not.

Comments

Blank lines are ignored.

Anything on a line after a '#' character is ignored.

## MOTION

save\_position(position\_name)

The current robot position is stored in the Positions Table as position\_name.

move\_linear(position\_name)

The robot moves along a linear path to Position position\_name.

move\_joint(position\_name)

The robot performs a joint move to Position position\_name.

move\_relative(dx\_mm, dy\_mm)

Move (dx\_mm, dy\_mm) relative to current tool position. If the part geometry selected is CYLINDER or SPHERE, robot moves along the part.

move\_tool\_home()

Perform a joint move to the home position associated with the current tool.

# LEonard User Manual

---

```
move_tool_mount()  
Perform a joint move to the mounting position associated with the  
current tool.
```

```
free_drive(0=OFF|1=ON)  
Turn robot free drive mode on or off.
```

The commands below provide a programmatic way to set the default motion parameters.

```
set_linear_speed(speed_mm/s)  
set_linear_accel(accel_mm/s^2)  
set_joint_speed(speed_deg/s)  
set_joint_accel(accel_deg/s^2)  
set_blend_radius(blend_radius_mm)
```

## ENGINEERING USE ONLY

### User Timers

Enabling these will time each grind operation and place it in a circular buffer of user\_timers that can be returned to the variable list with *return\_user\_timers()*. Used for internal testing.

```
enable_user_timers(0=disable|1=enable)  
zero_user_timers()  
return_user_timers()
```

### Low-level Setup Calls

These are all called automatically by *select\_tool(...)*, *set\_part\_geometry(..)*, and the *grind\_...()* commands. They should be used that way through the high level interface except during testing.

```
set_part_geometry_N(1=FLAT|2=CYLINDER|3=SPHERE, diam_mm)  
set_tcp(x,y,x,rx,ry,rz)  
set_payload(mass_kg,cog_x_m, cog_y_m, cog_z_m)  
set_door_closed_input(dig_in, value)  
set_footswitch_pressed_input(dig_in, value)  
set_tool_on_outputs(dig_out, value, ...) Can have up to 4 listed  
set_tool_off_outputs(dig_out, value, ...) Can have up to 4 listed  
set_coolant_on_outputs(dig_out, value, ...) Can have up to 4 listed  
set_coolant_off_outputs(dig_out, value, ...) Can have up to 4 listed  
set_output(DOUT, 0|1)  
tool_on()  
tool_off()  
coolant_on()  
coolant_off()  
send_robot(param1, param2, ...)
```

## Example Recipes

Here are a few recipes that show the kinds of things that can be done in a recipe. The **Testing** subdirectory in the Recipes folder has many more complicated examples that you can examine (and run!)

### Remove Current Tool

Just remove the current tool from the robot. As long as the one actually mounted is selected, this goes to the tool home followed by the mount/demount position and prompts the operator when it is time to remove.

```
# Remove Current Tool
# Go through demount procedure
# Assumes you have selected whatever tool is actually mounted!

prompt(Please confirm: you wish to demount {robot_tool}?)

move_tool_home()
move_tool_mount()
prompt(Please demount tool {robot_tool})

select_tool(None)
```

### Install A Tool

This goes through prompting to mount a specific tool.

```
# Install 2F85
# Example to install a tool when none is currently installed
# We just select the new tool, move to the mount position, prompt the
operator, and move to move_tool_home

# Change to whatever tool you like
tool=2F85

# Operator confirmation
prompt(About to mount {tool})

# Mounting process
select_tool({tool}) # This only informs the robot what is mounted

# This does the physical swap
move_tool_mount()
prompt(Please mount tool {tool})
move_tool_home()
```

## Integrated Example

Here we start with the 2F85 tool ready to grind and swap tools and continue from the same location mid-recipe.

```
# Integrated Example
# Assumes we're where we want to grind initially but need to do a tool
swap mid-way

tool1=2F85
tool2=vertest

# Program assumes we are starting with tool1- verify internally and
with operator!
assert(robot_tool,{tool1})
prompt(Confirming tool {tool1} is currently mounted and you are
grinding on {robot_geometry})

# This will always be our grind_start position
save_position(grind_start)

# Do some grinding with tool1
move_linear(grind_start)
grind_rect(30,30,3,10,10,1)
grind_rect(20,20,3,10,10,1)

prompt(Ready to swap {tool1} to {tool2}?)
# Remove {tool1}
move_tool_home()
move_tool_mount()
prompt(Please remove {tool1})

# Install {tool2}
select_tool({tool2})
move_tool_mount()
prompt(Please install {tool2})
move_tool_home()

# Do some grinding with tool2
move_linear(grind_start) # Returns us to the starting position
grind_rect(30,30,3,10,10,1)
grind_rect(20,20,3,10,10,1)
```

## Computed Concentric Circles

Here's a test recipe that grinds 3 concentric circles explicitly and in a loop, not lifting until the final one.

```
# 26 Concentric Circle Test

# Old school
grind_circle(30,2,0.9,10,1)
grind_circle(20,2,0.9,10,1)
grind_circle(10,2,0.9,10,0)

# Do it with a loop
size = 30
count = 2
speed = 0.9
force = 10

repeat:
    grind_circle({size},{count},{speed},{force},1)
    size -= 10
    jump_gt_zero(size,repeat)
```

## Lots of Grinds

By pre-teaching points and swapping geometries, a whole day's work could be done (other than tool swaps!)

```
# Test all the patterns on all the geometries

size1=40
size2=10
count=3
speed=5
force=10

select_tool(2F85)
cycleCount=0

redo:
move_linear(demo_flat)

set_part_geometry(FLAT,0)
grind_line({size1},{size2},{count},{speed},{force},1)
grind_line(-{size2},{size1},{count},{speed},{force},1)
grind_rect({size1},{size2},{count},{speed},{force},1)
grind_rect({size2},{size1},{count},{speed},{force},1)
grind_serp({size1},{size1},1,3,{count},{speed},{force},1)
grind_serp({size1},{size1},3,1,{count},{speed},{force},1)
grind_circle({size1},{count},{speed},{force},1)
grind_circle({size2},{count},{speed},{force},1)
```

# LLeonard User Manual

---

```
grind_spiral({size1},{size2},3,{count},{speed},{force},1)

set_part_geometry(CYLINDER,400.1)
grind_line({size1},{size2},{count},{speed},{force},1)
grind_line(-{size2},{size1},{count},{speed},{force},1)
grind_rect({size1},{size2},{count},{speed},{force},1)
grind_rect({size2},{size1},{count},{speed},{force},1)
grind_serp({size1},{size1},1,3,{count},{speed},{force},1)
grind_serp({size1},{size1},3,1,{count},{speed},{force},1)
grind_circle({size1},{count},{speed},{force},1)
grind_circle({size2},{count},{speed},{force},1)
grind_spiral({size1},{size2},3,{count},{speed},{force},1)

set_part_geometry(CYLINDER,600.1)
grind_line({size1},{size2},{count},{speed},{force},1)
grind_line(-{size2},{size1},{count},{speed},{force},1)
grind_rect({size1},{size2},{count},{speed},{force},1)
grind_rect({size2},{size1},{count},{speed},{force},1)
grind_serp({size1},{size1},1,3,{count},{speed},{force},1)
grind_serp({size1},{size1},3,1,{count},{speed},{force},1)
grind_circle({size1},{count},{speed},{force},1)
grind_circle({size2},{count},{speed},{force},1)
grind_spiral({size1},{size2},3,{count},{speed},{force},1)

set_part_geometry(CYLINDER,800.1)
grind_line({size1},{size2},{count},{speed},{force},1)
grind_line(-{size2},{size1},{count},{speed},{force},1)
grind_rect({size1},{size2},{count},{speed},{force},1)
grind_rect({size2},{size1},{count},{speed},{force},1)
grind_serp({size1},{size1},1,3,{count},{speed},{force},1)
grind_serp({size1},{size1},3,1,{count},{speed},{force},1)
grind_circle({size1},{count},{speed},{force},1)
grind_circle({size2},{count},{speed},{force},1)
grind_spiral({size1},{size2},3,{count},{speed},{force},1)

set_part_geometry(CYLINDER,1000.1)
grind_line({size1},{size2},{count},{speed},{force},1)
grind_line(-{size2},{size1},{count},{speed},{force},1)
grind_rect({size1},{size2},{count},{speed},{force},1)
grind_rect({size2},{size1},{count},{speed},{force},1)
grind_serp({size1},{size1},1,3,{count},{speed},{force},1)
grind_serp({size1},{size1},3,1,{count},{speed},{force},1)
grind_circle({size1},{count},{speed},{force},1)
grind_circle({size2},{count},{speed},{force},1)
grind_spiral({size1},{size2},3,{count},{speed},{force},1)

set_part_geometry(SPHERE,400.2)
grind_line({size1},{size2},{count},{speed},{force},1)
grind_line(-{size2},{size1},{count},{speed},{force},1)
```

# LEonard User Manual

---

```
grind_rect({size1},{size2},{count},{speed},{force},1)
grind_rect({size2},{size1},{count},{speed},{force},1)
grind_serp({size1},{size1},1,3,{count},{speed},{force},1)
grind_serp({size1},{size1},3,1,{count},{speed},{force},1)
grind_circle({size1},{count},{speed},{force},1)
grind_circle({size2},{count},{speed},{force},1)
grind_spiral({size1},{size2},3,{count},{speed},{force},1)

set_part_geometry(SPHERE,600.2)
grind_line({size1},{size2},{count},{speed},{force},1)
grind_line(-{size2},{size1},{count},{speed},{force},1)
grind_rect({size1},{size2},{count},{speed},{force},1)
grind_rect({size2},{size1},{count},{speed},{force},1)
grind_serp({size1},{size1},1,3,{count},{speed},{force},1)
grind_serp({size1},{size1},3,1,{count},{speed},{force},1)
grind_circle({size1},{count},{speed},{force},1)
grind_circle({size2},{count},{speed},{force},1)
grind_spiral({size1},{size2},3,{count},{speed},{force},1)

set_part_geometry(SPHERE,800.2)
grind_line({size1},{size2},{count},{speed},{force},1)
grind_line(-{size2},{size1},{count},{speed},{force},1)
grind_rect({size1},{size2},{count},{speed},{force},1)
grind_rect({size2},{size1},{count},{speed},{force},1)
grind_serp({size1},{size1},1,3,{count},{speed},{force},1)
grind_serp({size1},{size1},3,1,{count},{speed},{force},1)
grind_circle({size1},{count},{speed},{force},1)
grind_circle({size2},{count},{speed},{force},1)
grind_spiral({size1},{size2},3,{count},{speed},{force},1)

set_part_geometry(SPHERE,1000.2)
grind_line({size1},{size2},{count},{speed},{force},1)
grind_line(-{size2},{size1},{count},{speed},{force},1)
grind_rect({size1},{size2},{count},{speed},{force},1)
grind_rect({size2},{size1},{count},{speed},{force},1)
grind_serp({size1},{size1},1,3,{count},{speed},{force},1)
grind_serp({size1},{size1},3,1,{count},{speed},{force},1)
grind_circle({size1},{count},{speed},{force},1)
grind_circle({size2},{count},{speed},{force},1)
grind_spiral({size1},{size2},3,{count},{speed},{force},1)

cycleCount++
jump(redo)
```

# LLeonard User Manual

---

## Index

### Standard Library Commands

`leClearConsole`, 31

`leLogInfo`, 31

`lePrint`, 31

`lePrompt`, 31

`leShowConsole`, 31