SoftLinx 5.2 Guide

Hudson ROBOTICS, INC.

10 Stern Avenue, Springfield, NJ, USA 07081
Tel: +1-973-376-7400 ♦ Fax: 973-376-8265

Table of contents

SoftLinx 5.2

SoftLinx 5.2 is Hudson Robotics' latest laboratory automation application suite designed to program and operate laboratory work cells.

SoftLinx 5.2 is made of 5 applications:

- **SoftLinx Plug-in Manager**: Allows for plug-in installation and manipulation. Plug-ins are scripts used by SoftLinx to command software or instruments.

- **SoftLinx Protocol Editor**: Allows for protocol development. Users can create, test, and execute protocols with this application.

- **SoftLinx Player**: Allows for simplified protocol execution without exposure to editing the protocol. Protocol wizards can be implemented here.

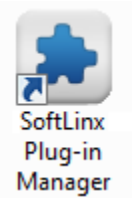
- **GO Wizard Editor**: Allows for the creation of a guided operation (GO) wizard which can be attached to and run by protocols when executed in the SoftLinx Player.

- **SoftLinx Log Viewer**: Allows users to see how a protocol ran on the system. Logs can be exported and sent to Hudson for further analysis and support.

This help file will aid the user in running the SoftLinx 5.2 application suite.

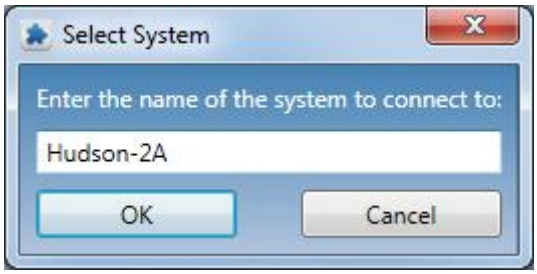
Please note that Windows 7 or above is required for SoftLinx 5.2.

SoftLinx Plug-In Manager

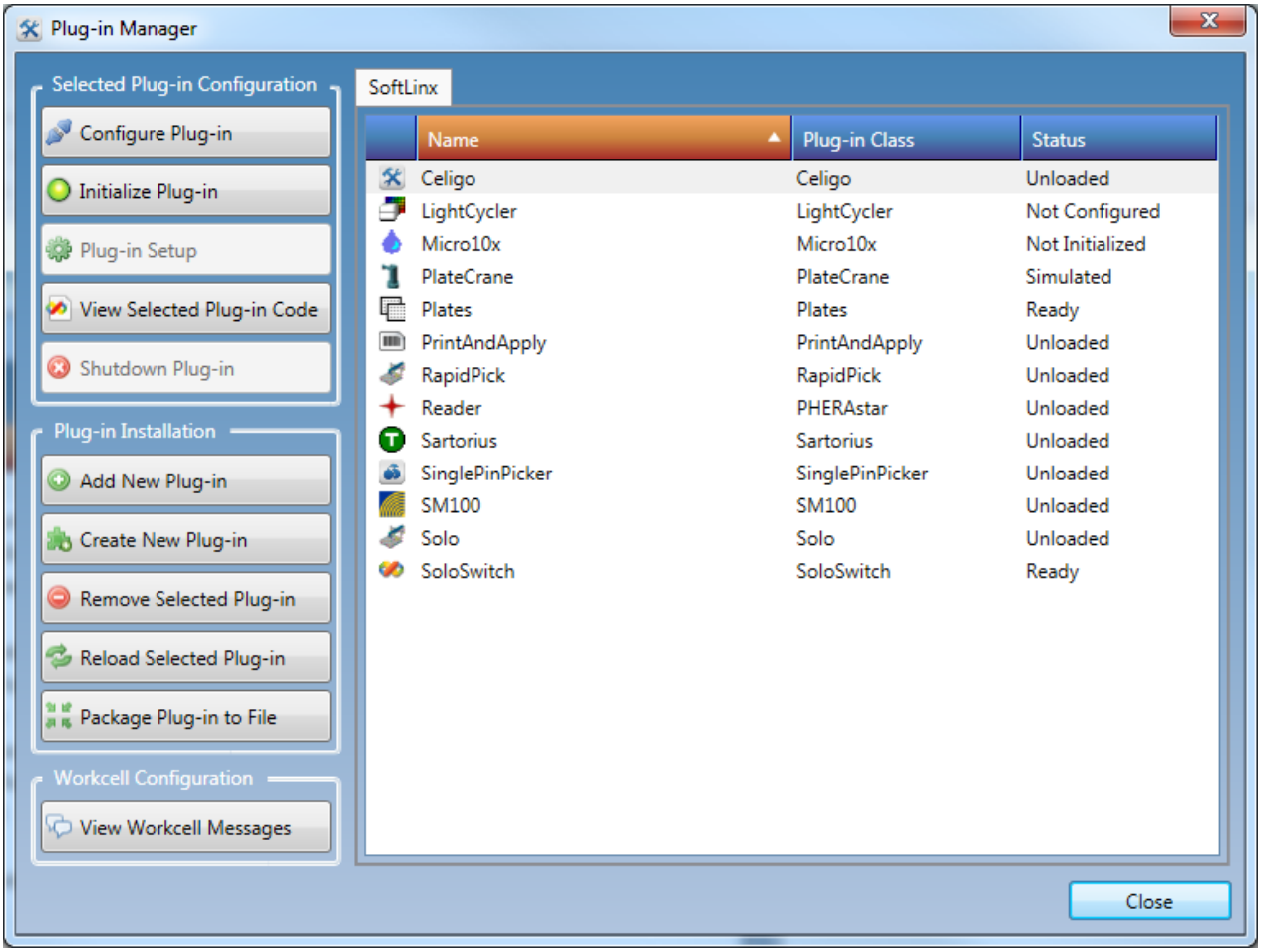


The plug-in manager hosts a variety of options to set up the user's workcell in preparation for protocol development. This program can be accessed via Programs - > Hudson Robotics - > SoftLinx or through the Protocol Builder by clicking "Manage Plug-Ins".

When opened externally, a taskbar notification icon will appear for the manager, allowing for easy access. Right click the icon to open the main window. In addition, if another system is on the same local network and a user wishes to connect remotely, select "Change System" and enter the computer name in the window that appears. This gives access to the workcell on that system.



Main Window and Statuses



The plug-in manager hosts a variety of options to set up the user's workcell in preparation for protocol development. The above list shows all instruments available to the user within SoftLinx. Each plug-in will list its unique name, an identification icon, its instrument class (or specific type of instrument), and its current status. Although each plug-in's name must be unique, SoftLinx can simultaneously control multiple plug-ins of the same class.

Plug-ins can have one of many statuses:

Unloaded - The plug-in is fully shut down. Configure or Initialize the plug-in to activate it. Opening a protocol which uses this plug-in will tell SoftLinx to initialize the plug-in.

Not Configured - The plug-in is operating, but connection settings have not been established. Use the configure button to specify connection settings to the plug-in, which will allow SoftLinx to initialize.

Not Initialized - Connection settings have been established, but the plug-in has not been initialized. Initialization requires that the instrument must test its connection settings, to see if it is to be considered Ready (Fully Initialized) or Simulated.

Simulated - The plug-in is active and can be used, but it does not have a connection to the instrument. Running the plug-in in a SoftLinx protocol while simulated will not run any commands that require the use of the instrument.

Ready - The plug-in has been connected to an active instrument, and can be used in protocols.

Each plug-in has configuration options available, which are required to set up the unit.

Configure Plug-in - Allows the user to modify communication parameters between the instrument or software and the plug-in on the computer.

Initialize Plug-in - Applies the configuration settings to the plug-in, and activates it for use. If successful, the plug-in will be placed in a "Ready" state. Plug-ins may also be put in a "Simulated" state if the instrument is unavailable.

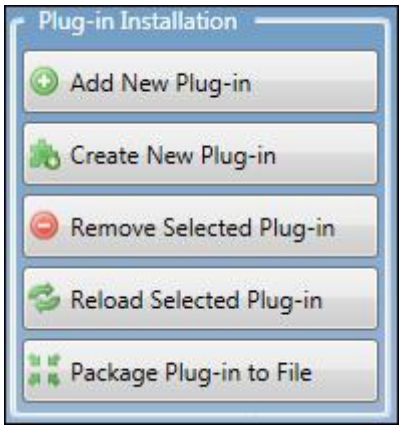
Plug-in Setup - Calls the "Setup" command, opening the selected instrument's setup window to manipulate runtime settings, such as position setup, operation speed, and other runtime modifications.

View Selected Plug-In Code - Opens the control script for the selected instrument within its own IDE (Integrated Development Environment). Users can debug and edit code of each plug-in.

Shut Down Plug-in - Deactivates the plug-in. Communication between the plug-in and instrument or outside software is severed.

Plug-In Installation

Plug-in installation is quick and easy.



The following commands are available:

Add New Plug-in - Installs instrument plug-ins onto the system. One plug-in must be installed per instrument. Multiple instances of the same plug-in can be installed in a workcell. See Plug-in Installation for details.

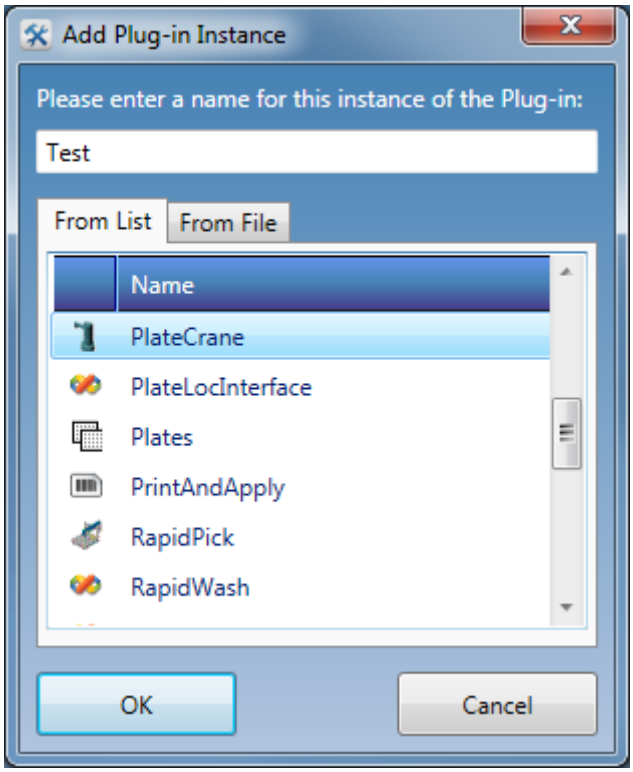
Create New Plug-in - Starts a new plug-in template, which can be used to create new plug-ins. See Plug-in Installation for details.

Remove Selected Plug-in - Remove the selected plug-in from the list. In addition, if the plug-in to be removed is the last instance of its plug-in class within the list, it will be fully uninstalled from the system. Protocols which require this removed plug-in will still include the steps of this plug-in, however the plug-in within the toolbox will be marked as "Unavailable". All protocol steps which originate from the plug-in will be treated as inactive until it is reinstalled.

Reload Selected Plug-in - Refresh the files of the plug-in from its SIP. This can be used to remove modifications done to the plug-in or fix corrupted files.

Package Plug-in to File - Compresses the files of the plug-in into a single SIP, which can be moved to other workcells for reinstallation.

To add a new plug-in, simply click on the Add New Plug-in button. The following window will appear:



The list of plug-ins available are pre-installed with SoftLinx. Additionally, plug-ins can be side loaded if the *.SIP (SoftLinx Plug-in) file is available on the machine. Use the "From File" tab to locate *.SIP files.

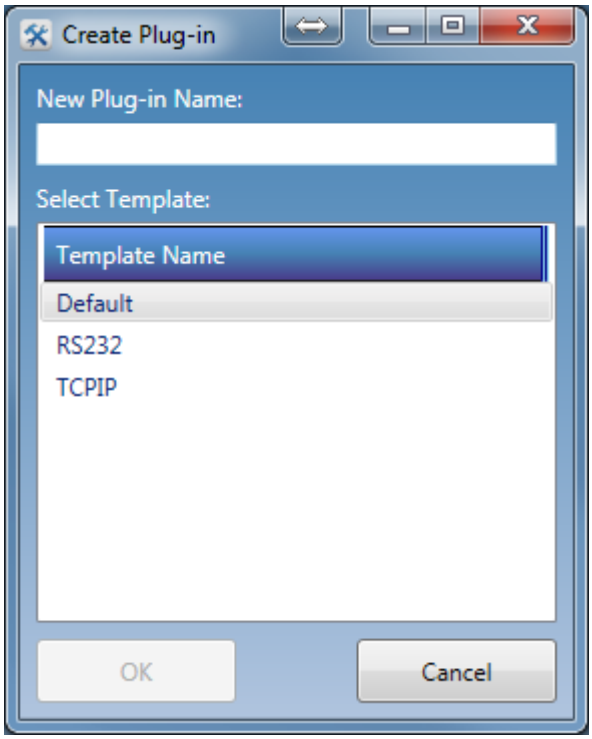
The instance name is used to identify the specific unit in this workcell. It is filled in by the name of the plug-in by default, however there can only be one unique name per plug-in per workcell. If a workcell has 2 or more of the same unit, they must be named differently.

Interfaces which are already installed on the machine have their proper icon listed with their name.

Once added, it will be in the list of available plug-ins in the main manager window, but will have an unloaded status.

In addition to installing interfaces, users may do the following:

Create New Plug-In - Allows the user to create a new plug-in, based off of an available interface template. Clicking on the button will open the following:



Users name the plug-in in this window by class, not by instance name. Templates are available to use as starting points for plug-in development. Once named, the newly created plug-in will immediately open and be available for development. The RS232 and TCPIP templates install controls with the plug-in that allow for RS232 and TCPIP communication, respectively.

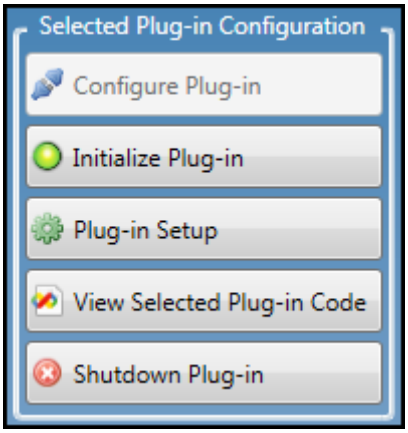
Note that if the user removes (or un-installs) the plug-in without packaging it to a SIP, the plug-in cannot be recovered.

Remove Selected Plug-in - Removes all plug-in files permanently from the computer. SIPs are not removed.

Reload Selected Plug-in - Reinstalls a plug-in from the location it was first installed, and overwrites all files of the plug-in with those from the associated SIP file.

Package Plug-in to file - Packages a plug-in folder to an SIP file, and saves it to a location defined by the user.

Plug-In Configuration



The following buttons will aid in plug-in manipulation:

Configure Plug-in - Allows the user to modify communication parameters between the instrument, plug-in, and computer. This button is active only when the plug-in is in an "Unloaded" or "Not Configured" state.

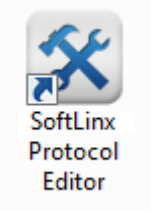
Initialize Plug-in - Applies the configuration settings to the plug-in, and activates it for use. If successful, the instrument will be placed in an "Initialized" state. Instruments may also be put in a "Simulated" state if the instrument is unavailable.

Plug-in Setup - Calls the "Setup" command, opening the selected plug-in's setup window to manipulate runtime settings, such as position setup, operation speed, and other runtime modifications. This is not available if a plug-in is running within a protocol.

View Selected Plug-in Code - Opens the script for the selected plug-in, typically within a VBA IDE. Users have the ability to edit code of each plug-in when enabled.

Shutdown Plug-in - Forces the plug-in to close and cease all operation, putting it back into the "Unloaded" state.

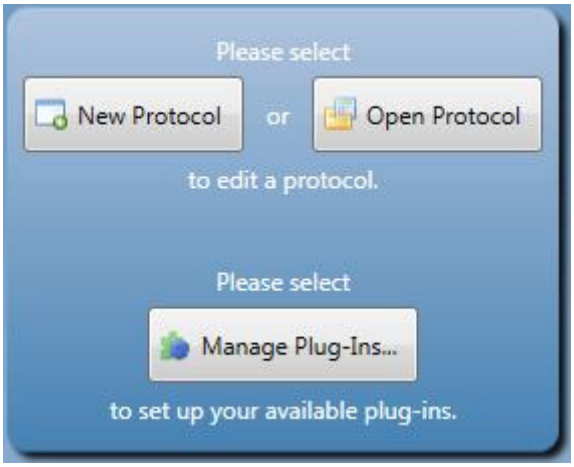
SoftLinx V Protocol Editor



The Protocol Editor is the main program of SoftLinx. It is where the processes which control instruments, known as protocols, are made. Once built, they can be run in this program or the SoftLinx Player to perform processes.

By default, all users can access the Protocol Editor. However, some administrators can lock access to the program and force end users to use the Player only for protocol execution.

Main Menu



Upon starting the Protocol Builder, or whenever no protocols are open, the window will display the screen above. Users may do one of the following:

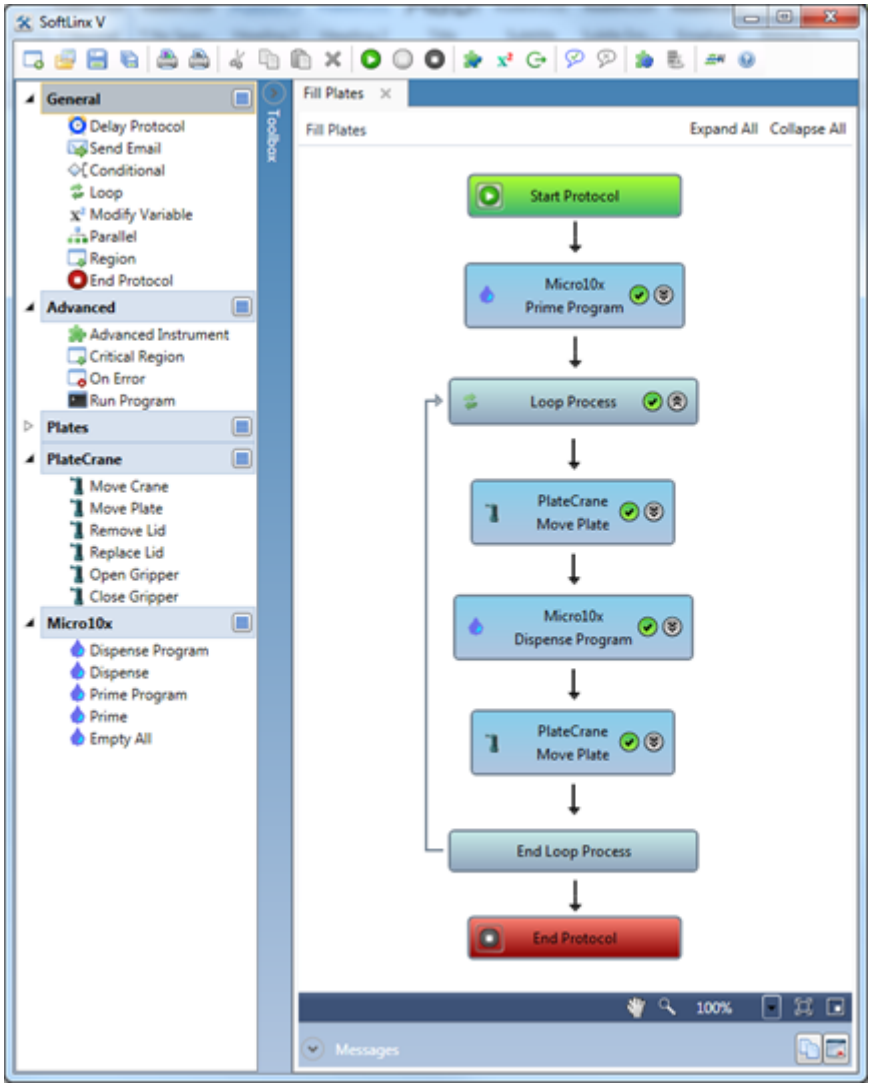
New Protocol - Creates a new protocol. Opens a new tab with an empty protocol. Activates default plug-ins, if any are specified as default.

Open Protocol - Opens a previously-saved protocol. Loads the protocol into a new tab. Activates all associated plug-ins.

Manage Plug-ins - Opens the Plug-in Manager.

When a protocol is opening, plug-ins associated with that protocol will initialize.

Protocol Editor Main Window



Above is the main protocol editor screen. Protocols can be viewed, modified, and executed here. Multiple protocols can be opened and run simultaneously.

All available protocol steps are found in the toolbox on the left, and specific messages regarding the current protocol are reported in the Messages frame. Both the toolbox and the messages frames can be minimized.

Menu Bar:

The menu bar is located at the top of the window. This provides basic functions of the protocol editor window.

The following commands, with keyboard shortcuts, are accessible to users:

New Protocol - Opens a new tab with an empty protocol. (Ctrl + N)

Open Protocol - Opens a tab with a previously saved protocol. (Ctrl + O)

Save Protocol - Saves the currently viewed protocol to a .SLVP file. If it is a new protocol, the user must specify a save file name. (Ctrl + S)

Save Protocol As - Saves the currently viewed protocol to a new .SLVP file. The user must specify a save name. (Ctrl + Shift + S)

Print to Image - Specify a printer and print the current protocol as an image.

Print to Text - Specify a printer and print the current protocol to a text file. The protocol will be printed in an outline format. Example:

- Protocol: Prepare Plate Part 1
    - Loop steps 15 times.
        - Crane.MovePlate - MovePlate from 'Stack1' to 'Micro10x.Nest'
        - Micro10x.Dispense - Dispense 50uL to Plate.
        - Crane.MovePlate - MovePlate from 'Micro10x.Nest' to 'StackLink.Pos1'
        - StackLink.ReturnPlate - Return plate from 'StackLink.Pos1' to StackLink.Stack2
    - End Loop
- End Protocol

Cut - Copies the selected steps to the clipboard. Removes the selected steps from the protocol, if such steps can be removed. (Ctrl + X)

Copy - Copies the selected steps to the clipboard. Does not remove the steps. (Ctrl + C)

Paste - Places the copied steps onto a selected location. (Ctrl + V)

Delete - Removes the selected steps from the protocol. (Delete key)

Run Protocol - Starts execution of the currently visible protocol. (F5)

Pause Protocol - Pauses the currently visible protocol only. Pausing does not pause all protocols. Pressing Play resumes the protocol. (Space Bar)

Stop protocol - Stops and terminates the currently visible protocol, if it is running.

Select Plug-ins - Choose plug-ins that the protocol will use.

Manage Protocol Variables - Opens the Protocol Variable Manager.

GO Wizard Editor - Opens the GO wizard for the current protocol.

Manage Plug-ins - Opens the Plug-in Manager.

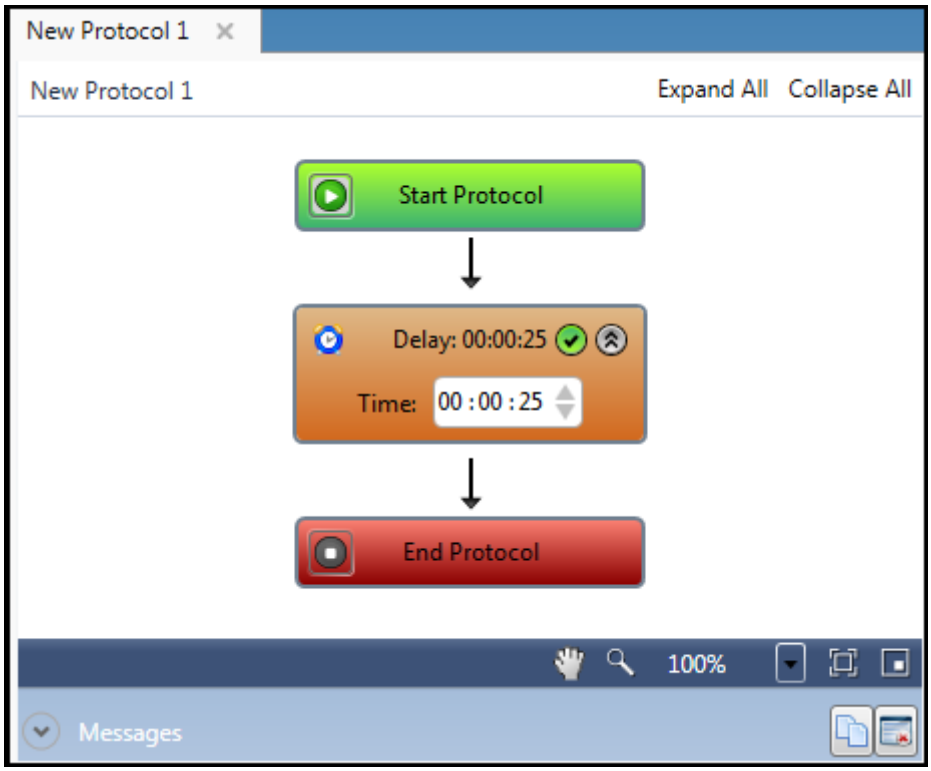
Show/Hide Protocol Comments - Expands/Hides all steps' comment boxes, which are located on all steps. Comment boxes allow users to place notes on each step.

About - Shows information about this program, including version numbers of SoftLinx and all installed interfaces.

Help - Opens this help file. (F1)

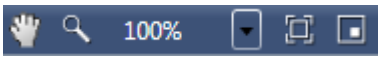
Protocol Frame

The protocol designer frame shows the protocol currently being edited. Multiple protocols can be opened, but only one protocol can be shown at a time. Protocols are managed by a tab system, located at the top of the frame.



The designer can expand or condense all protocol steps, by clicking on the Expand All or Collapse All links in the upper right corner of the frame. To restore protocol steps and allow for custom expand and collapse actions, click on the Restore link, which will replace the Expand or Collapse links when clicked.

In the lower right corner of the frame are the protocol navigation tools. These are used to change the viewing area of the protocol.



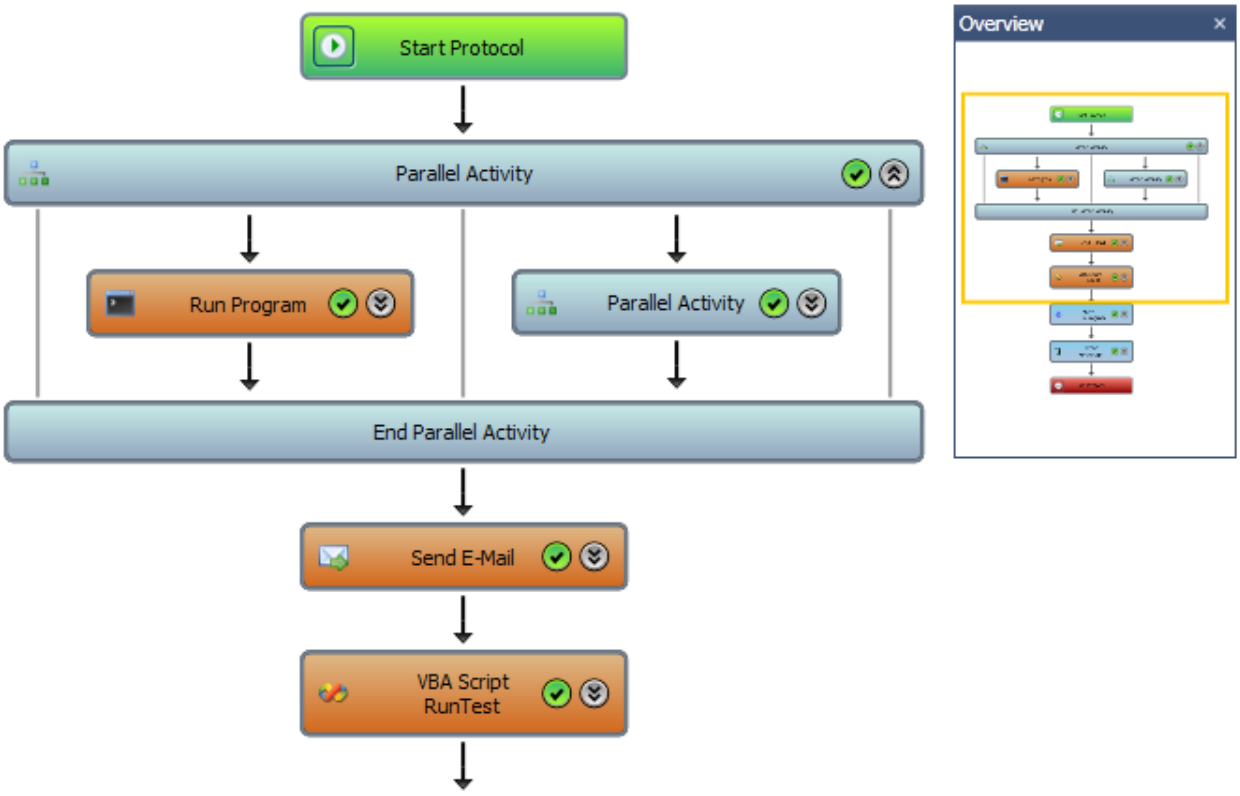
 The Pan icon allows a user to scroll across a protocol if it does not fit within the viewing area.

The zoom value is customizable by the user, ranging from 25% to 400%.

The magnifying glass will restore the zoom value to a default of 100%.

The fit all button  will attempt to fit the entire protocol into the view window.

The mini map button will create an overview window, or a mini map, of the entire protocol in a separate window. Users can drag the yellow box in the overview window to change the visible area of the protocol, as shown on the image below.



In addition, the messages tab has two buttons. The will copy all messages to the Clipboard. The will clear all messages within the messages tab.

Testing the Protocol

Within the builder, protocol execution is controlled by using the three buttons in the task bar:

These buttons will only control the active protocol. Any other protocols running from other unselected tabs, or from other programs, will not be affected by these buttons.

The play button will only be active when the current protocol is stopped or paused. The pause button will only be active when the current protocol is running. The stop button will be active when the current protocol is running or paused. SoftLinx protocols may run simultaneously, from the builder, player, other external programs, or any combination of the above.

Protocol Execution:

Once a protocol is saved and verified, users may execute the protocol. Users may either click Play on the task bar or press F5.

Hitting the Run command will do the following, in order:

- Ask the user to save the protocol before execution, if the protocol has not been saved since the last change. If it has, it will confirm execution of this protocol. This is the only confirmation before execution.

- Any variables that have the parameter "Prompt for Value" marked true will pop a window open prior to the start of the protocol asking to validate the value of said variable.

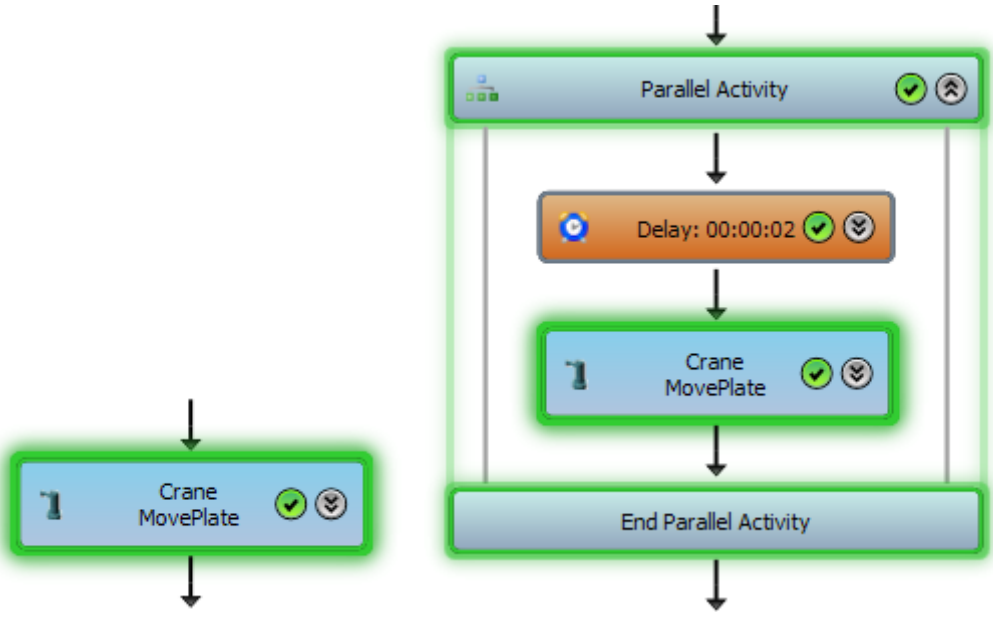
- Causes all active instruments, when not currently engaged in a protocol, to activate their startup procedures. Instruments do this simultaneously. Status messages will be displayed in the "Messages" box, located at the bottom of the main SoftLinx V window.

- The protocol will then begin execution. Steps will execute sequentially, from top to bottom.

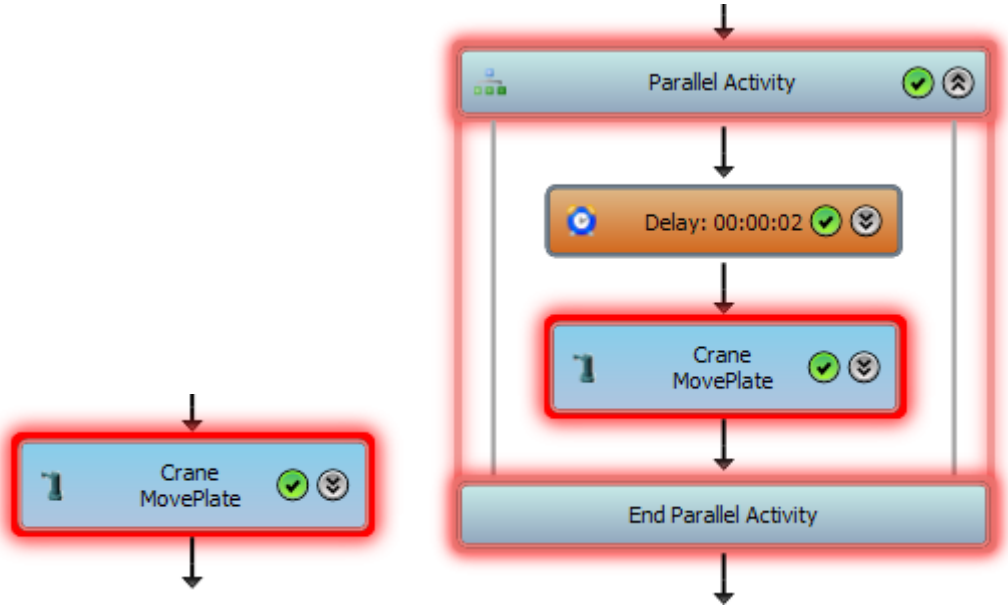
Runtime Options:

While a protocol is running, the user is able to monitor the runtime status of the overall protocol. Protocols cannot be changed while executing. In addition, all protocol messages will be posted to the message log, located below the protocol frame. Execution of protocols can be interrupted with the pause or stop buttons.

Steps that are currently executing will be marked with a green aura. Expanded steps with children steps that are executing will be marked with a light green aura.



In addition, if a user manually stops a protocol run by SoftLinx, the last running steps which was executing will be marked with a red aura. Red auras will remain until the user restarts the protocol.

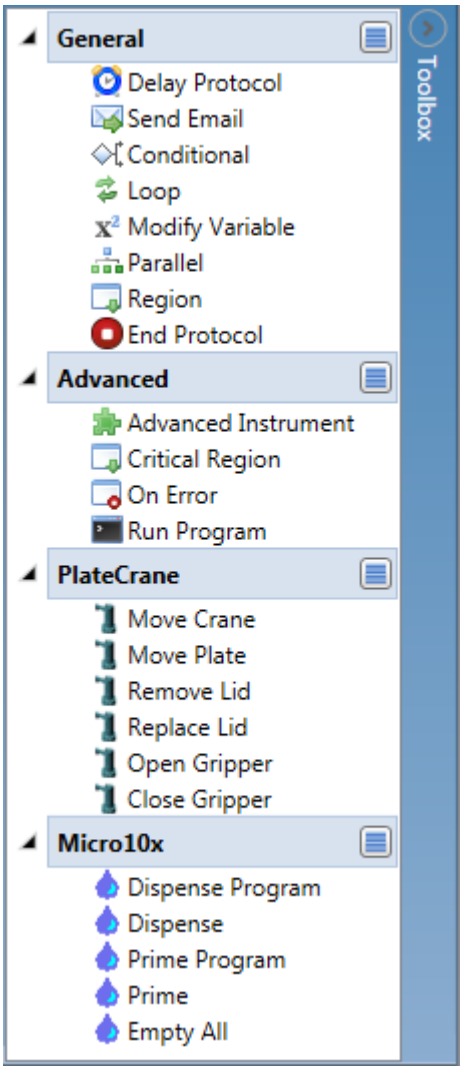


Protocol Steps

Steps are commands that can be executed by SoftLinx. There are several different types of steps used in protocol development. SoftLinx has a flexible set of steps the user can place in protocols.

The Toolbox:

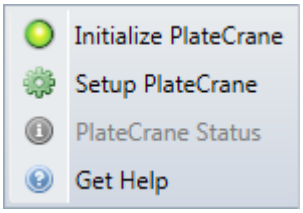
The Toolbox holds every step that can be utilized by users within a protocol. General steps are listed, as well as steps from every active plug-in.



Only plug-ins which are marked as active for the selected protocol will show their steps in the toolbox.

The toolbox is located on the left side of the protocol editor window. In addition, the toolbox can be minimized by clicking on the blue bar between the protocol tabs and the toolbox.

By clicking on the icon, or right clicking on a plug-in name, the user may access additional options for that instrument, similar to those found in the Plug-in Manager window:



Initialize - Refreshes the plug-in's connection to the instrument.

Setup - Opens the Setup for that plug-in. This is only available when a protocol with this plug-in active is not running.

Status - Opens the Status window for that plug-in, if available. This is only available while a protocol is running.

Get Help - Opens the Help file for the selected plug-in, if available.

Additional info about Steps:

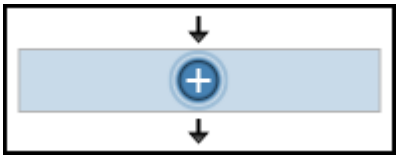
Steps are specific actions which will be executed in protocols. There are different types of steps. Most perform specific actions, while some control the flow of the protocol. Built-in steps are separated by General and Advanced markers.

All steps have an activate button and an expand button.

The Activate button can be toggled to activate or deactivate steps, and will define whether or not a step will run. The expand button can be clicked to show additional details for that specific step. For composite steps (steps which have activities housed within themselves), the expand button can be used to show all of the child steps within it, or compress the step.

In addition, the labels of composite steps can be edited by placing the mouse over the label and clicking it.

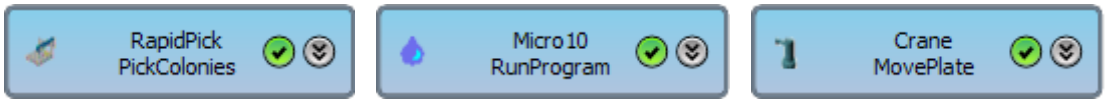
To utilize the steps, users can simply drag and drop items from the toolbox onto the protocol. Users can drag steps onto visible arrows between steps. If a step can be placed, a drop target will be shown.



Upon dropping the step, the user will be presented with options to modify that step, if applicable. Users may also double click the step on the protocol, or right click it to access those options.

Each protocol step type has its own options associated with it. See each step's help entry for details.

Plug-in Steps



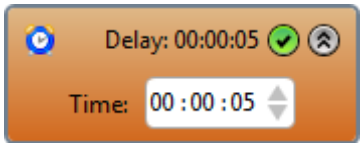
All plug-ins have a unique set of commands which the protocol editor will present to the user, also known as plug-in steps. Each command is placed in the toolbox for the user to access. Examples are above.

When dropped onto the protocol, the plug-in step will open its own step parameters window, if applicable. In addition, each plug-in step will be identified by a steel blue background, as well as its plug-in name, command, specific icon, and when expanded, details about the command issued.

See each plug-in's help file for details about each command.

Delay Protocol

The Delay Protocol step allows users to pause the flow of protocol execution at any point within the protocol.



Users can directly edit the time fields on the step. Users can add hours, minutes, or seconds as needed.

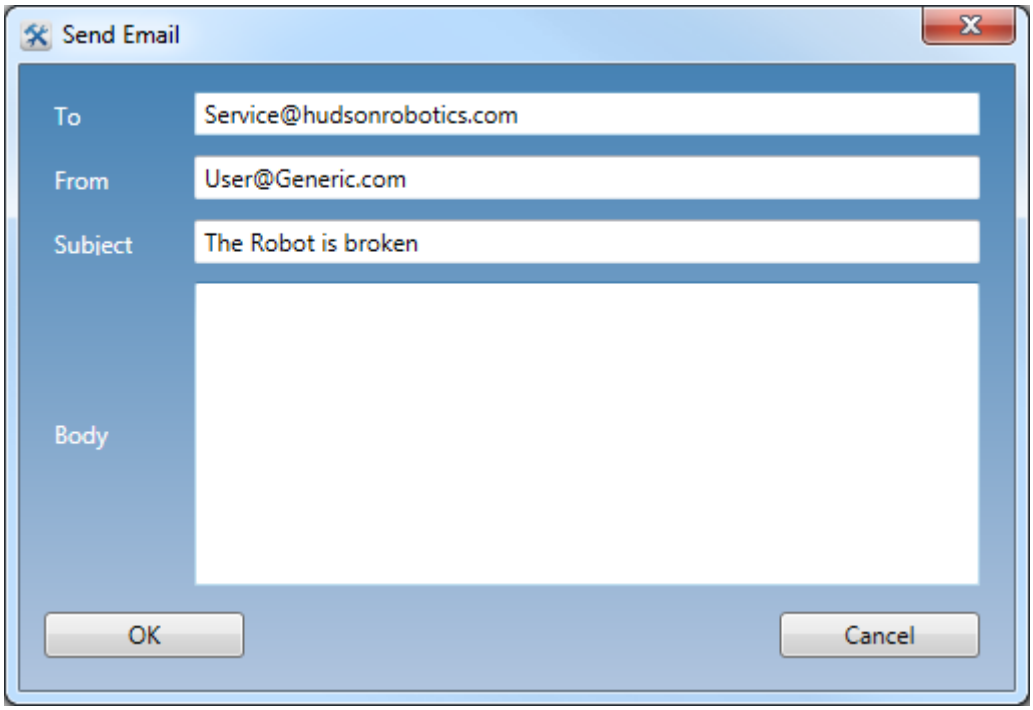
The Delay Protocol only delays execution of the next step in line, not the full protocol. In the case of a parallel activity, it will only delay the branch in which it is placed.

In addition, if the protocol is manually paused by the user, the delay is NOT paused internally. It will continue to count down as normal. If additional time is desired, simply keep the protocol paused until ready.

Send E-Mail

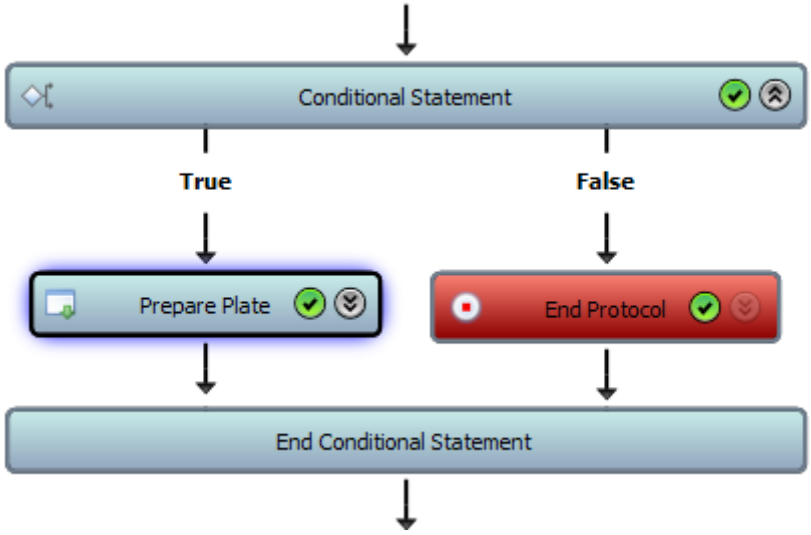
The Send E-mail step will allow a user to send out an e-mail message at a certain location within the protocol. The user simply adds the basic fields to an e-mail message: To/From addresses, subject, and body paragraphs.

The “To” and “From” addresses are required, and must be valid for e-mails to be sent. Subject and Body fields are not required.

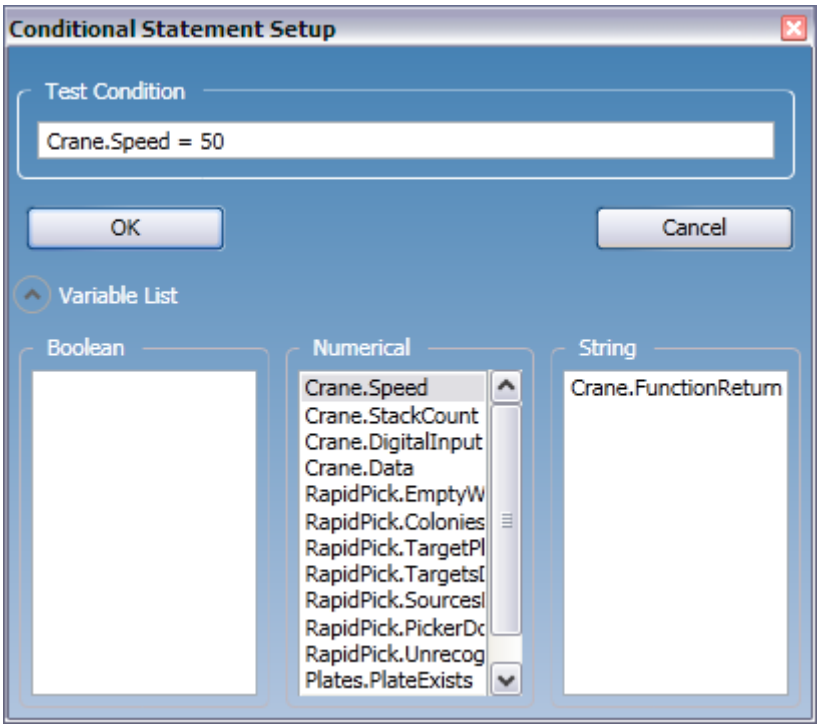


Conditional Statement

Conditional statements are used when a decision to run only one set of specific steps is necessary.



Based on the result of the current condition defined, one and only one of the two branches will execute its set of steps. Users can define a condition using the setup screen below:

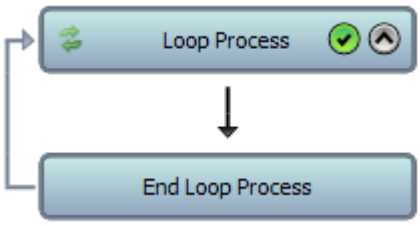


Users must write a function or statement which must evaluate to a "True" or "False" response, such as the example above, Crane.Speed = 50. If the conditional statement the user selects turns out to be true (Ex: If the value of Crane.Speed is actually 50, then the statement is true), then the left branch will execute its steps. If the statement is false (Ex: If the value of Crane.Speed is actually 40, then the statement is false), the right branch will execute.

Note that any protocol-defined or valid interface variable can be used in the statement. SoftLinx will detect if the function or statement is able to evaluate to a True or False response.

Loop Process

Loops are composite steps, which will execute a set of activities within itself repeatedly for a set number of iterations. Steps must be placed between the loop and end loop boxes to be considered as part of the loop, and repeat as defined.



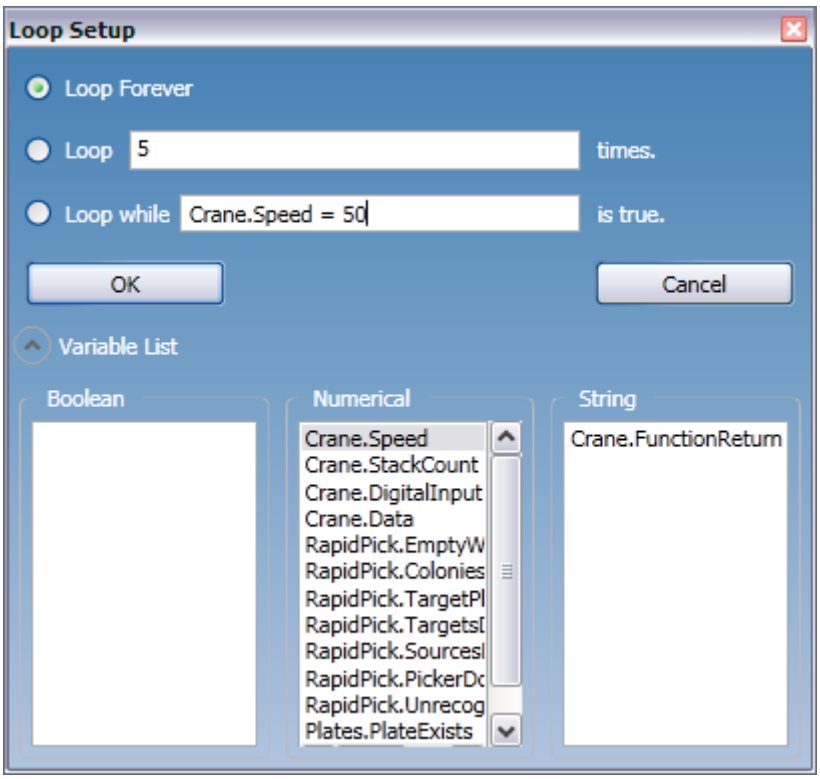
Users can define the number of times a loop runs by one of three ways:

- "Forever". This means the system will repeat the steps until an aborting condition is met (such as if the Platecrane can't retrieve a plate from a stack) or the user hits the stop button.

- Finite number, either predefined or variables.

- Loop by condition.

Loops are defined using the setup window shown below:



Finite loops are predefined, and will always run the loops for that number of iterations unless stopped by the stop button or an internal step. Loops that run forever can only be stopped by an internal step or the stop button.

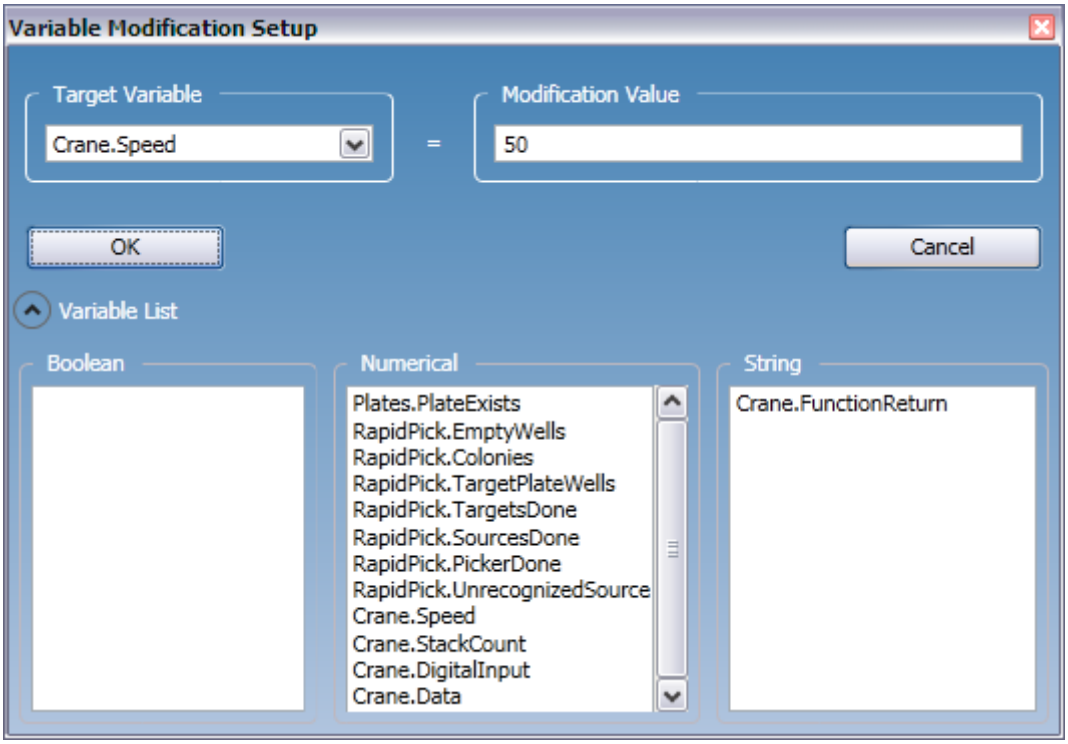
Conditional loops will run depending on whether or not a condition is satisfied. Users define a conditional statement, which if evaluated to true, will allow the loop will run. As long as that condition is true at the beginning of the loop's execution, the loop will continue to repeat itself. If the condition becomes false by the time the loop executes, or repeats execution, the loop will no longer execute, and the protocol will continue on to the next step following the End Loop box.

Conditions are checked before a loop executes its process, and will re-evaluate the condition before repeating the process each time to determine if the loop should run.

Note that any protocol-defined or valid plug-in variable can be used in the conditional defined for the loop. Invalid variables will not be accepted.

Modify Variable

The Modify Variable step will change a plug-in or protocol variable to a different value. Users can input direct values or functions to define the new value of a variable.



SoftLinx will ensure that the result of the modification value is valid before the user can confirm the step. Invalid entries will not be accepted.

Some of the instruments have special variables that can be used. Commonly used special variables are listed below.

SoftLinx.Platecrane.Speed - % of maximum speed in which the crane will run. This value can be modified in a protocol.

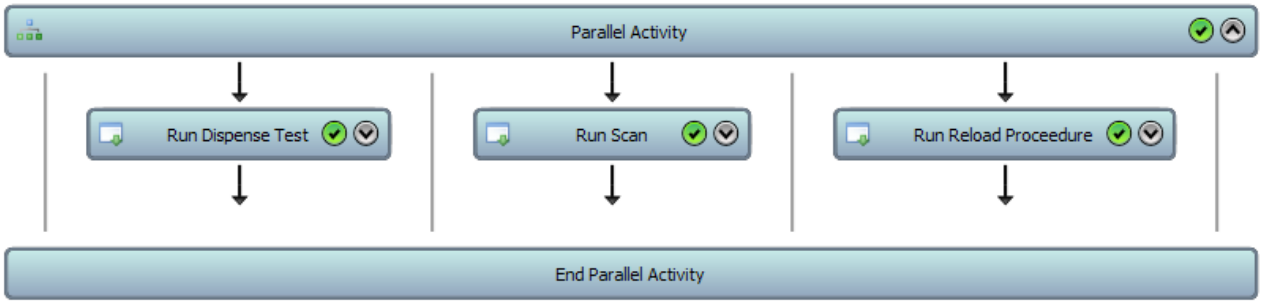
SoftLinx.Platecrane.StackCount(X) - Returns the number of plates that the Platecrane knows about in stack X. Replace X with the number of the stack in question. This can be used in conditionals. This value is read only.

SoftLinx.Plates.PlateExists("NAME") - Returns a 1 if a plate exists at location NAME, and a 0 if it does not exist. Replace NAME with a position name, such as SoftLinx.Micro10x.Nest. This can be used in conditionals. This value is read only.

SoftLinx.RapidPick.Colonies - Returns the number of colonies last scanned by the RapidPick. This can be used in conditionals. This value is read only.

Parallel process

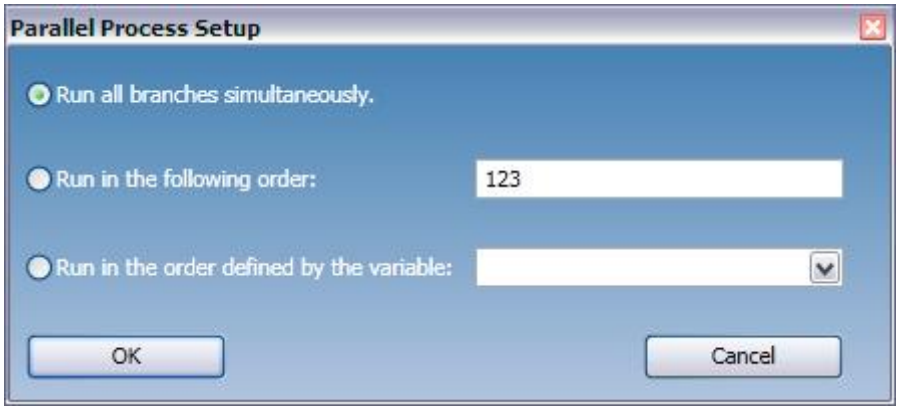
Parallel processes can run multiple sets of steps, either simultaneously or in a predefined order.



Each branch holds its own set of steps. When this process begins, it will begin execution of each branch, either simultaneously or in a specified order. This process will not complete until all branches have been executed, and all child steps have been completed.

Users may add branches by dragging steps onto the silver lines shown between steps. Only one silver line is shown if no steps exist within the parallel step. Users may then add steps to branches by placing items on the arrows.

Users can define the way this step will execute its branches through the following window:



By default, all branches will run simultaneously, and only one time.

When selecting "Run in the following order", the user can type in a series of characters that denote an order in which to run branches, or to run multiple branches multiple times. For example, if a user selects this option and the input string is 123, the parallel process will run branch 1, then branch 2, then branch 3, sequentially. Alternatively, if the user enters the string 321, the parallel process will run branch 3, then branch 2, then branch 1, sequentially. To run branches simultaneously, use (). If a user enters the string (12), then the parallel process will run branches 1 and 2 simultaneously, and not run branch 3 at all. This can lead to more complicated strings and execution of branches. Advanced Example: If a user enters the string (12)3(12)11(123), then the parallel process will run branches 1 and 2 simultaneously. Once the set is done, it will run branch 3 by itself. Then, it will run branches 1 and 2 simultaneously. Then, it will run branch 1 by itself, twice, in sequence. Finally, it will run all 3 branches simultaneously before it is completed.

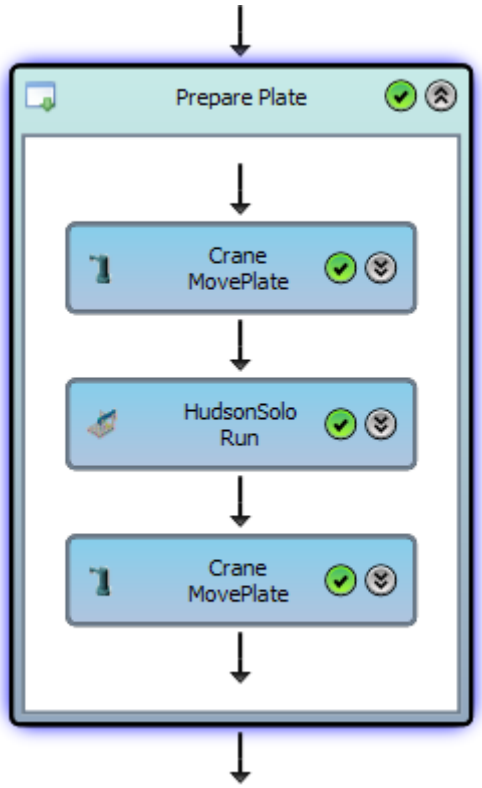
Should a user have more than 9 branches, the user must use A-Z to denote branches 10-35. Branches 36 and above cannot be run using this method, however it is very uncommon to have 36 or more branches under one sequential protocol step.

In addition, users can also select a protocol string variable to define the string used to control the execution of branches. The string must match the same format as specified above. Invalid characters will be ignored.

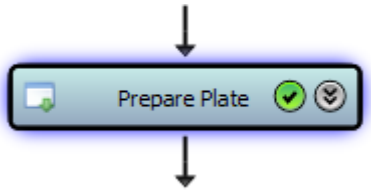
Region

Regions are collapsible actions which can hold any number of protocol steps. Users may place steps within a region, label the region, condense the region, and even deactivate it so that no steps within the region will execute at runtime. Region steps are used for organizational purposes, and do not have any special execution rules associated with them.

Expanded:

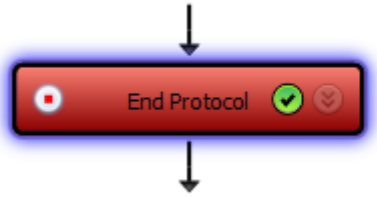


Condensed:



Regions do not have special execution rules, and will run the steps in order, as if they were never placed within the region.

End Protocol

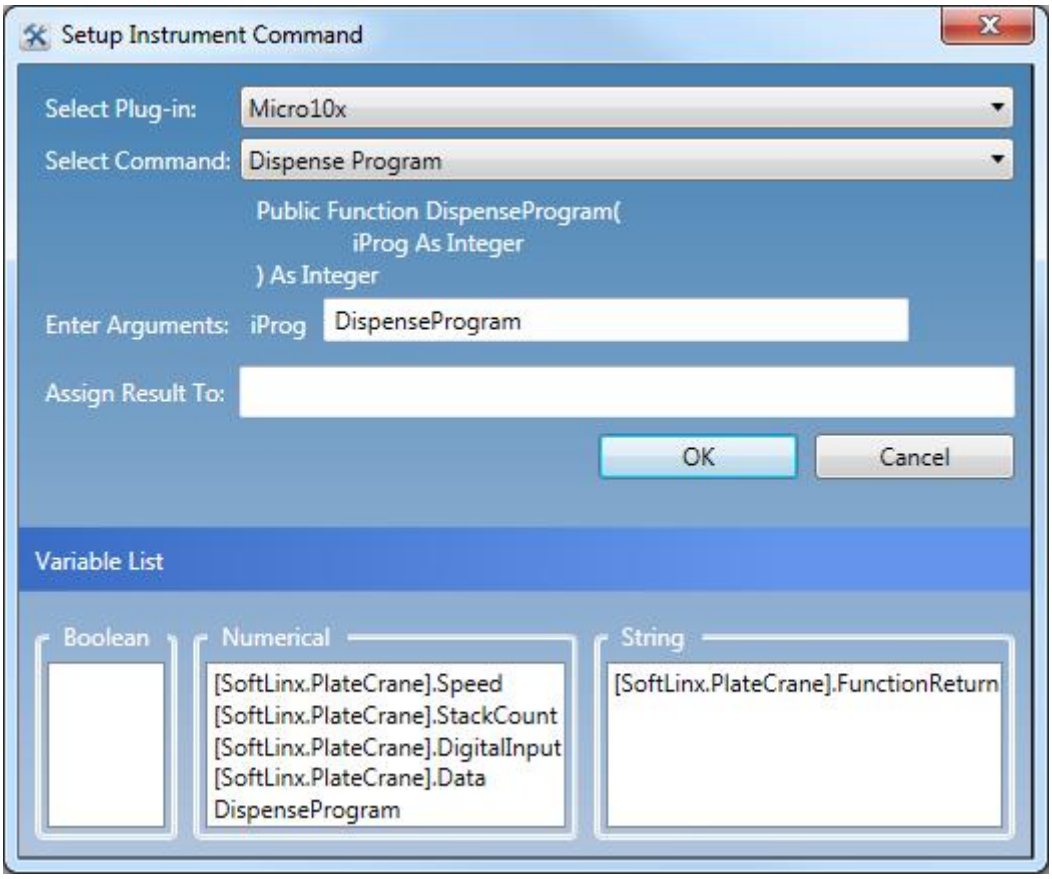


Stops execution of the protocol. This can be useful in conditional statements, or used as stop points if a user wants to only run a portion of a protocol without making major modifications to an existing protocol.

Advanced Instrument

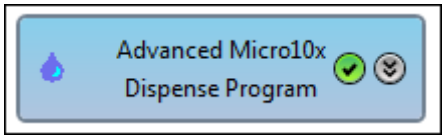
Advanced instrument steps allow users to directly call plug-in functions and assign the arguments themselves. Arguments assigned can be hard-coded values or variables. This command is intended for power users.

The advanced instrument step will show the following when placed:



This allows users to assign variables to arguments of plug-in functions, as well as retain the result of functions to variables after completion.

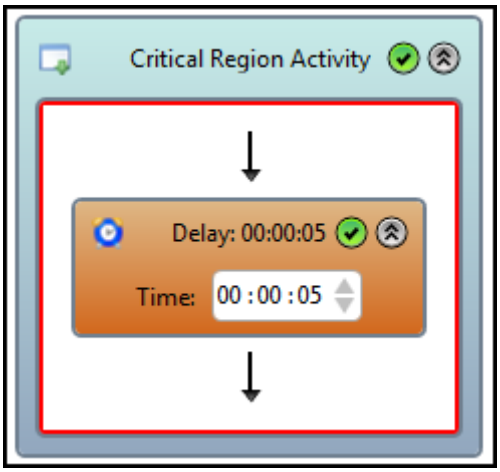
Advanced instrument steps that are placed will look like plug-in steps, but are identifiable with the "Advanced" tag.



Critical Region

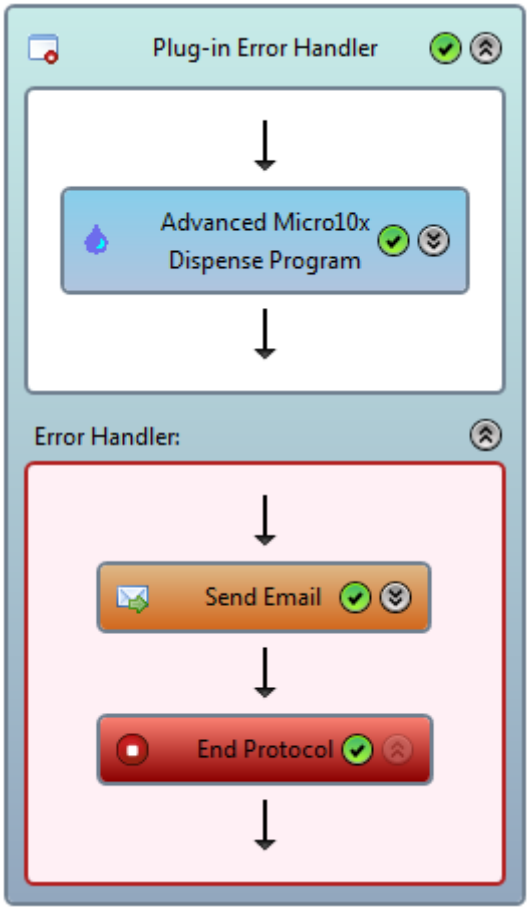
Critical regions are collapsible actions which can hold any number of protocol steps. Users may place steps within a critical region, label the critical region, condense the critical region, and even deactivate it so that no steps within the critical region will execute at runtime. Critical region steps are used for organizational purposes - however, unlike normal regions, critical regions have one special rule associated with them.

While a critical region is executing during a protocol, any other critical region that begins execution with the same caption as the first-executing region will be effectively paused and must wait for the first-executing critical region to complete before executing. In other words, if a critical region with a caption of A1 runs, any other critical region that attempts to start with a caption of A1 must wait until the actively running critical region with a caption of A1 is complete before it can execute. Any other critical regions that do not have a matching caption can still run, however.



On Error

"On error" steps resemble region steps with two sections. The top section will always execute. The bottom "error handler" section will execute only if an error is detected.



If the plug-in step above does not produce an error, the steps within the error handler section will not be executed. However, if the step does produce an error, anything within the error handler section will execute. This allows users to recover from errors successfully.

If the same step was outside the plug-in error handler and produced an error, the protocol may end.

Run Program

The Run Program step will run one command line string or open an executable with optional arguments.

Users will enter the path of the executable and the arguments in the proper text boxes. Users may also test the command by clicking on the "Test Command" button.

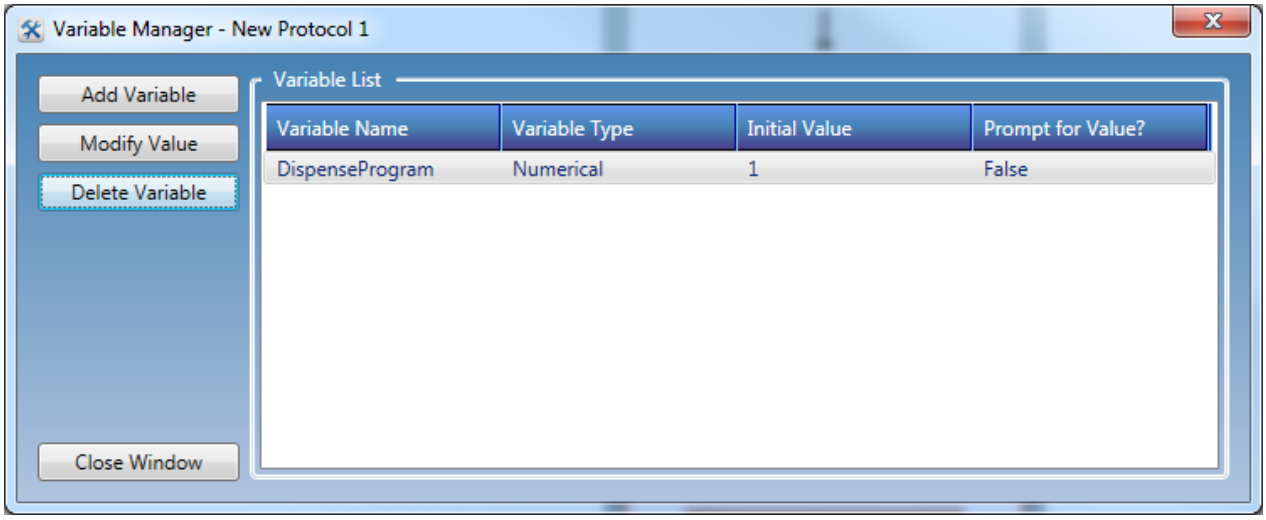
The "Hide Command Prompt" option will run the command without the command prompt window showing up at runtime, if applicable. The "Wait for Completion" option will force the protocol to wait for the complete execution of the command before continuing on to the next step.

Protocol Variables

Protocol variables can be defined to help facilitate the execution of the protocol's steps. Variables are often used in controlling Loop Process steps, Conditional Statements, or even directly setting values within plug-ins themselves through the use of the Modify Variable step.

By clicking on the variable icon, the user can define variables for that specific protocol. Variables defined in this manner do not transfer to other protocols outside of the one in which they are defined.

Protocols support True/False (Boolean) variables, numerical variables, and string variables.



Variables must be named using alphanumeric characters, and underscores only. Variable names must also begin with a letter.

Variables can be added, modified, or deleted from any protocol before execution.

Protocol Build Examples

Protocols can be made in any number of ways. There are no correct answers when making protocols, assuming that the protocol completes the task in question.

It is recommended to follow these guidelines when creating protocols.

1. Define the task that the protocol is to complete.

One needs to do their best to understand the problem at hand and how to solve it manually before applying robots to complete the task. Write out the steps to see what it takes to complete the task.

2. Solve the task for one plate.

Create a linear protocol that handles one plate. This will help the person creating the protocol to understand how to complete the process in an automated fashion. It may help to build the protocol in pieces.

3. Test the protocol for one plate

This can be done at any point in the process. The person must watch the protocol run to ensure that there are no errors in execution that cannot be seen by software, such as collisions, timing errors, etc.

If the protocol is large (over 10 steps), it is recommended to test the protocol in segments of 5-10 steps at a time.

4. Modify the protocol for multiple plates.

If the protocol requires more than 1-2 plates, loops and parallel processes may be required. Single plate protocols may be easier to build than multiple plate protocols. Once the protocol has been developed for one plate, develop a way to chain multiple plates for processing.

5. Continue to test for every change made

Any changes made to the protocol may have consequences for plates if not tested properly. No matter how small the change, always test to ensure that the protocol is functioning as intended.

Protocol Tutorial - Simple

Protocols are sets of instructions given to the instruments to perform in a specified manner.

The following instructions will advise on how to create a simple protocol. Please note that any protocol can be created in multiple ways.

1. Define your task.

It is very important to clearly understand the goal of the protocol before creating it. Let's define a sample problem.

Problem: We wish to fill 20 micro plates with solution, using the Micro 10 (liquid dispenser) and the Plate Crane (micro plate handler).

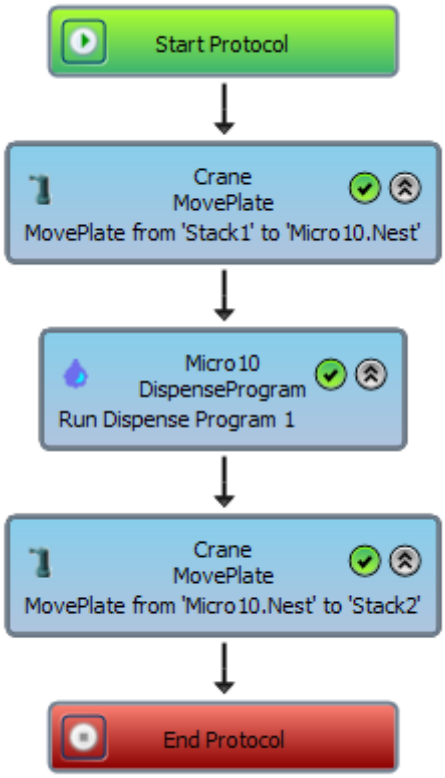
A sample outline to solve the problem can be seen below:

- Take a plate from a stack on the Crane, and place it in the Micro 10.
- Run Dispense Program 3 on the Micro 10.
- Move the plate from the Micro 10 to a different stack on the Crane.
- Repeat the process 20 times.

The following example is a short protocol, and can be completed by using 4 steps.

2. Create the protocol in a linear fashion.

It would be best to create a protocol for one plate, based on the outline above. Split the protocol into steps.



All steps can be dropped from the toolbox to the canvas to create the pattern above. Protocols run from start to finish, top to bottom.

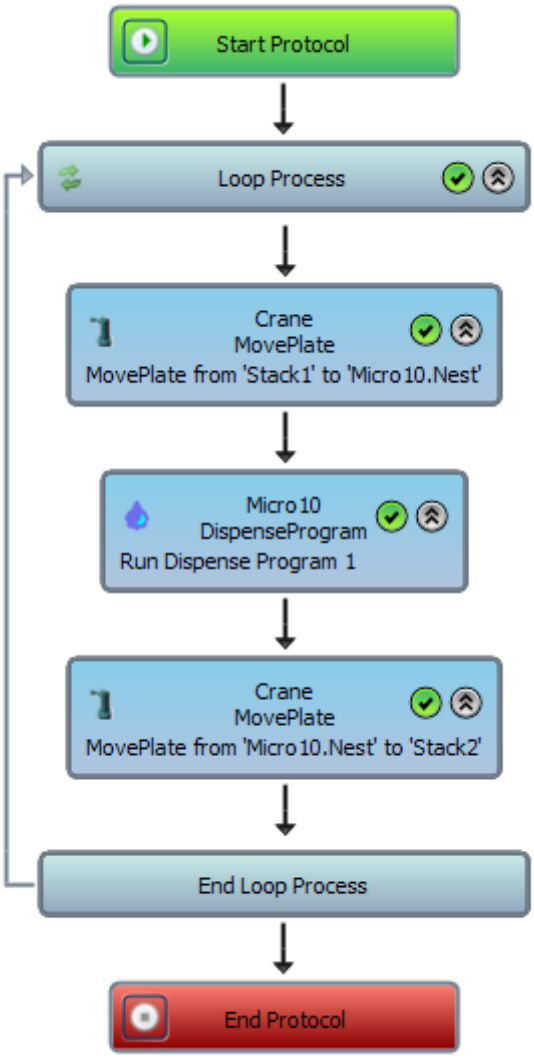
3. Test the protocol.

This should complete the goal for a single plate. It is recommended to run this protocol now, to ensure that it acts the way you want it to run, and that all instruments are running properly.

Ensure that the correct plate types and initial plate locations are defined for this protocol. A 96 well plate, with initial plate location at stack 1, is recommended.

4. Modify the protocol for multiple plates.

However, we are not done. This protocol is supposed to run 20 times. To repeat parts of a protocol, use a loop step, and place the items you wish to repeat within the loop.



There are multiple ways to set up this loop. Users can either set the loop up to run forever and empty stack 1, or run the loop a specific amount of times, or use variables to determine how many times a loop should run. In the example below, we can use the instrument variable "Crane.StackCount(1) > 0" to tell the loop to run until the number of plates in the Crane's stack 1, as detected by the Crane, is 0.

This concludes the basic protocol tutorial.

Protocol Tutorial - RapidPick

Advanced protocols require multiple steps of various types to complete. A sample advanced protocol which utilizes the Rapid Pick is shown below.

1. Define your task.

Situation: We have a bunch of bacteria colonies in agar plates that we would like to pick, using the Rapid Pick colony picker, and place into 96 well micro plates. The micro plates must be pre-filled with solution from the Micro 10, and later sealed by a plate sealer. We also have a plate crane to move plates between all of the instruments.

It is highly recommended to break up this task into parts. With SoftLinx V, the user can use region activities to label each part of the protocol separately.

This protocol has 3 parts.

A. Initial Loading of the colony picker. This portion of the protocol will only be done once.

B. Actual Colony Pick. The picker will be reloaded every time plates are needed.

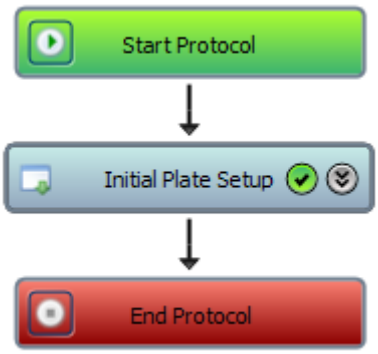
C. Final unloading of the picker. Will only be done once.

For this example, the protocol requires the following interfaces:

Crane - Micro Plate Handler
Micro10 - Liquid Dispenser
RapidPick - Colony Picker
Brandel - Plate Sealer

2A. Build the Initial Plate Setup region.

We can focus on one part at a time. First, the initial loading of the colony picker. Place a region activity on the canvas, and label it "Initial Plate Setup".



Now, we must define what we need to do to set up the Rapid Pick for picking. Generally, preparation should go like this:

- Move a lidded colony source plate from a stack to the RapidPick.

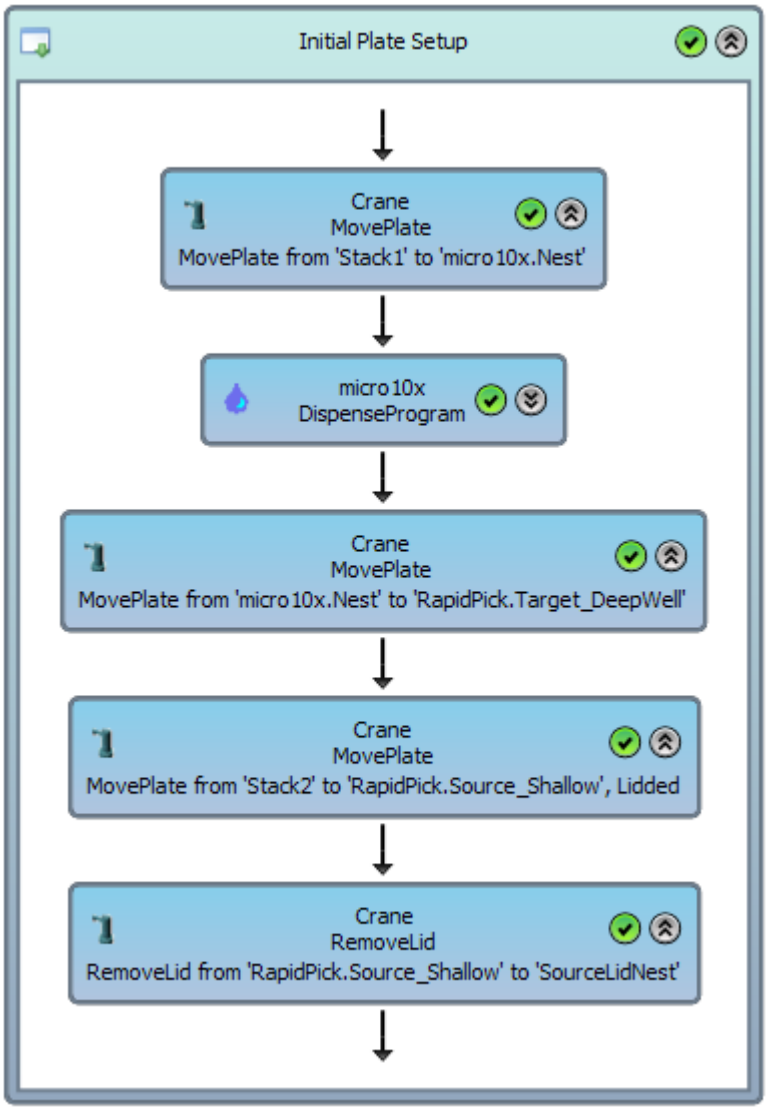
- Remove the lid from the source plate.

- Move a new target plate from a stack to the micro 10.

- Dispense liquid to the target plate.

- Move the prepared target plate to the RapidPick.

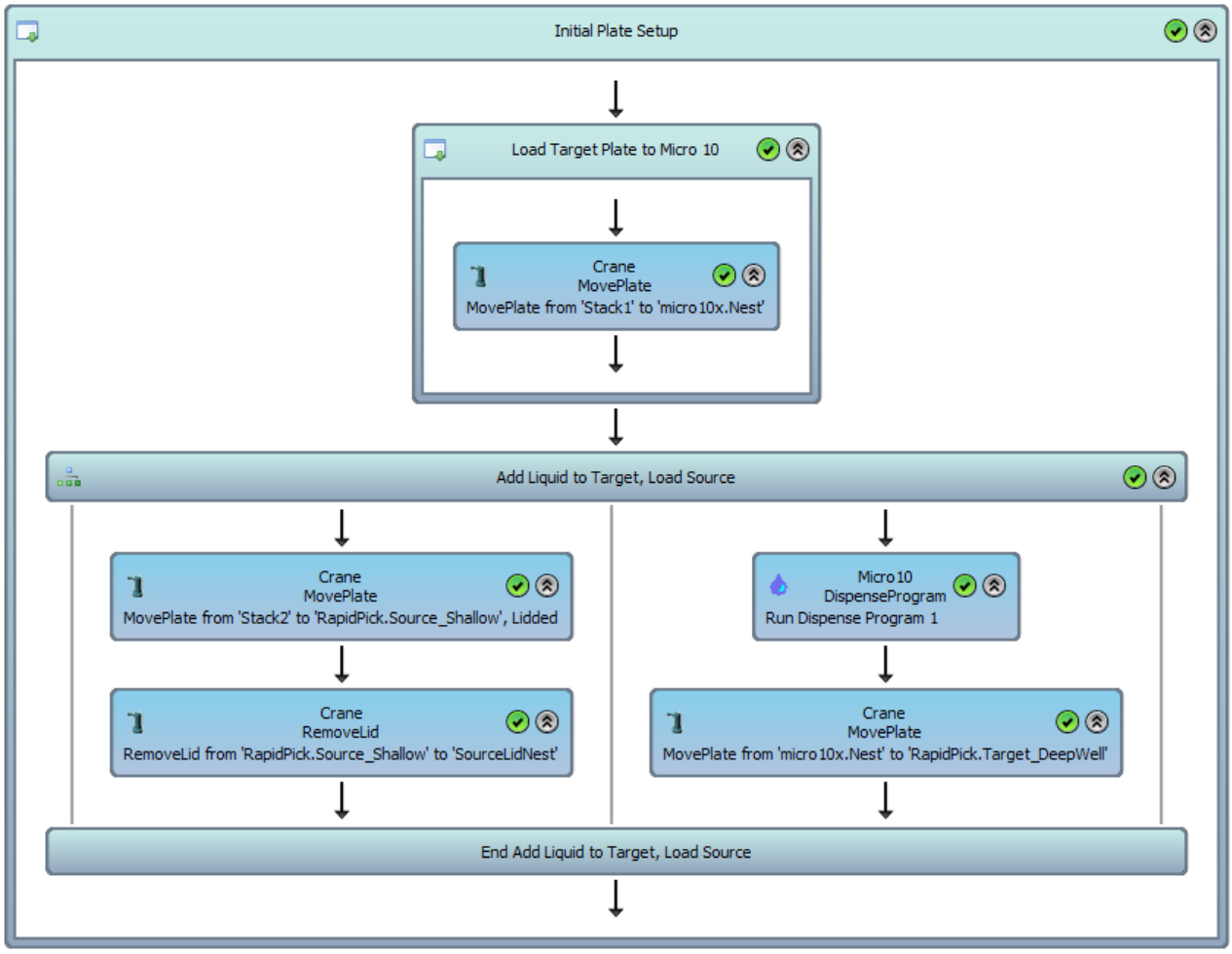
We can start by placing all the steps in a linear fashion, shown on the next page.



This will work and load the plates. However, we don't have to wait for the micro 10 to be finished with the target plate before we move the other source plate. We can modify this part of the protocol even further by using a parallel process activity to run multiple instruments at once.

Place a parallel process onto the canvas. Place the Micro 10 dispense step and the move target plate step in one branch, and the two steps that handle the source plate in another branch.

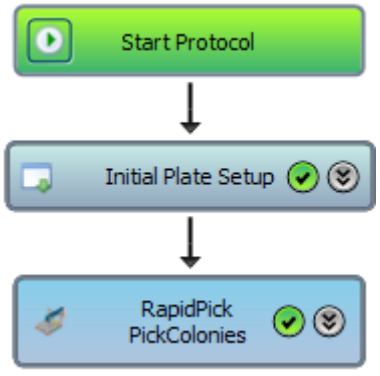
This part of the protocol will look like this:



This allows the Crane to prepare the source plate while running the Micro 10 on the destination plate simultaneously, saving time to run the protocol.

2B. Build the Pick Colony region.

The next part of the protocol will consist of picking the plates, and reloading the picker with new plates. Place this after the initial plate setup region. Note that you can also minimize regions.



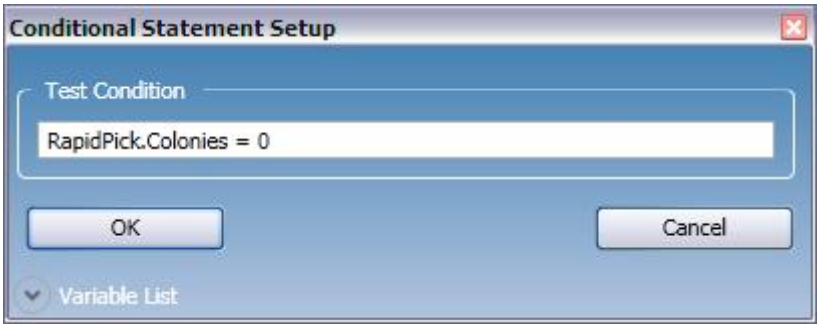
After a pick step is complete, we will need to do some type of clean up. The RapidPick will need to either remove the source plate, target plate, or both, before starting a new pick.

In this case, instrument variables can be used to make decisions within the protocol. The Rapid Pick has two variables which can be checked to see the status of the plates being worked on.

RapidPick.Colonies - Denotes the number of colonies remaining on a source plate. If this value is zero, remove the plate.

RapidPick.EmptyWells - Denotes the number of spaces left in a target plate that are yet to be filled.

When a decision needs to be made, the use of a conditional statement is generally required. Place a new conditional statement on the canvas. Enter the following condition:



When this statement shown is set to true, then the conditional statement's true branch will activate. Replacing the source plate will consist of the following steps:

- Replace the lid of the used source plate.

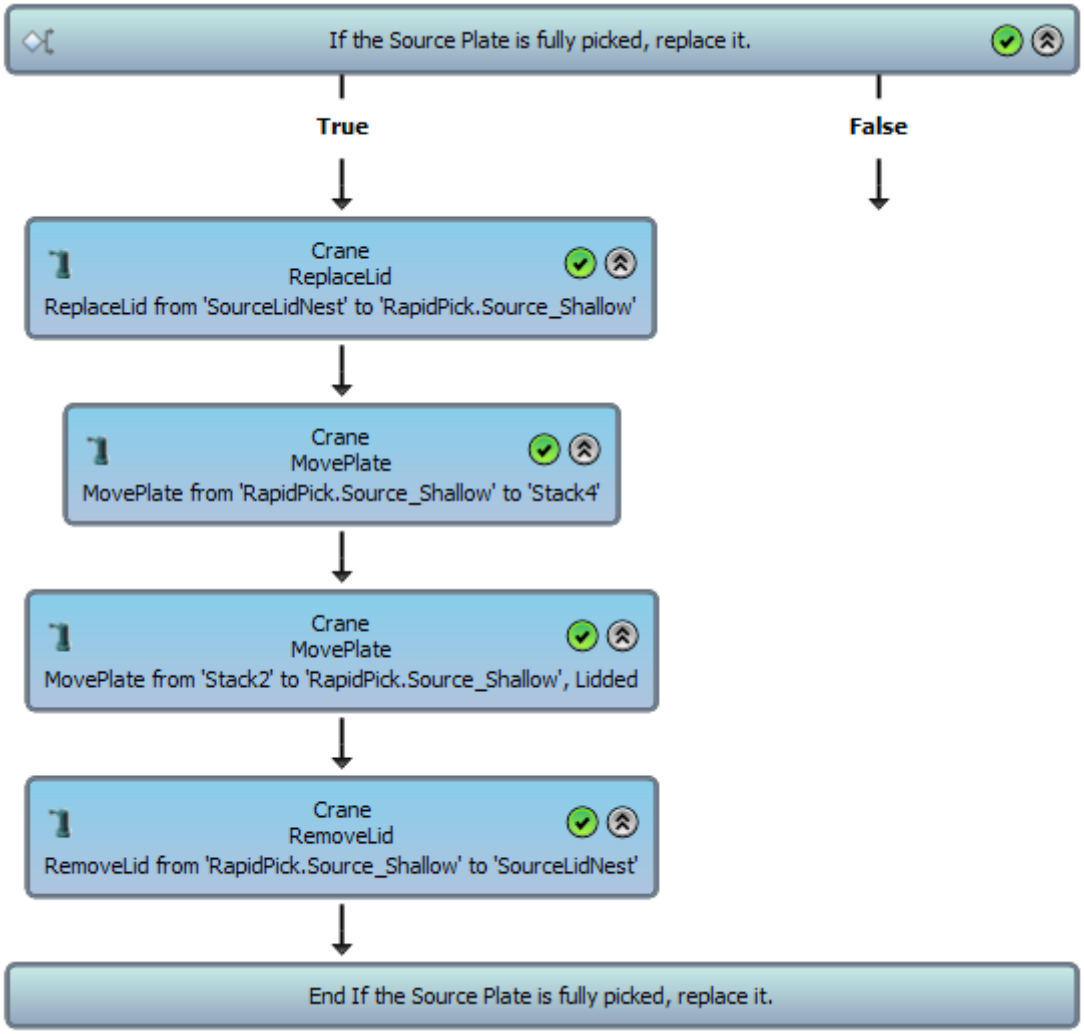
- Move the used source plate to a "used source stack".

- Move a new source plate to the source location on the RapidPick.

- Remove the lid from the new source plate.

Place the replace lid step, move plate steps, and remove lid steps into the true branch of the conditional statement. Do not place anything in the false branch.

The resulting conditional statement should resemble something like this:



This will replace the source plate when necessary.

The target plates are handled in a similar manner, although additional steps must be taken when preparing new target plates and storing finished target plates. The steps are as follows:

- Check the target plate. If it is not ready to be removed, do nothing.

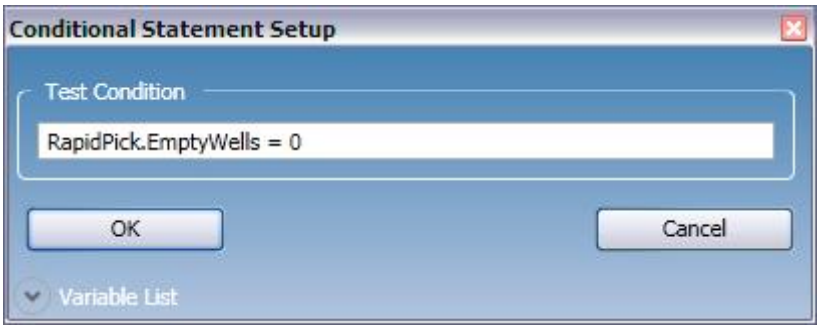
- If the plate is ready to be removed, move the finished target plate to the plate sealer.

- Begin sealing the plate. While sealing the plate, move a new target plate to the micro 10, if there are still source plates that can fill the remaining target plates.

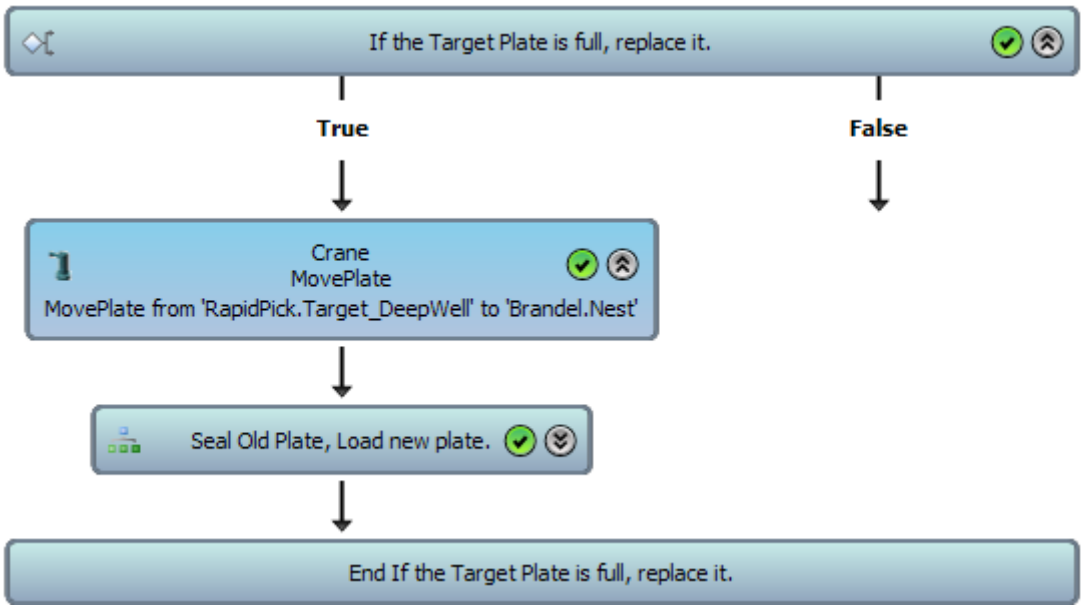
- Remove the finished target plate from the plate sealer, and place in a finished targets stack.

- Remove the newly filled target plate from the Micro 10, and move it to the target nest of the Rapid Pick.

As these steps have a few decisions to be made, place steps down, one at a time. Start with a conditional statement that checks the RapidPick.EmptyWells variable to see if we even need to replace the target plate.



Place a move plate step, directed at the plate sealer, and a parallel process activity labeled "Seal old plate, load new plate." as shown below.



Next, expand the "Seal Old Plate..." process. We need to run simultaneous actions here. We are going to simultaneously seal the old plate, while preparing the new plate, if a source plate is still available with colonies to be picked. The steps of sealing the old plate are as follows:

- Run the sealing program on the plate just placed there.

- Move the plate to a final destination.

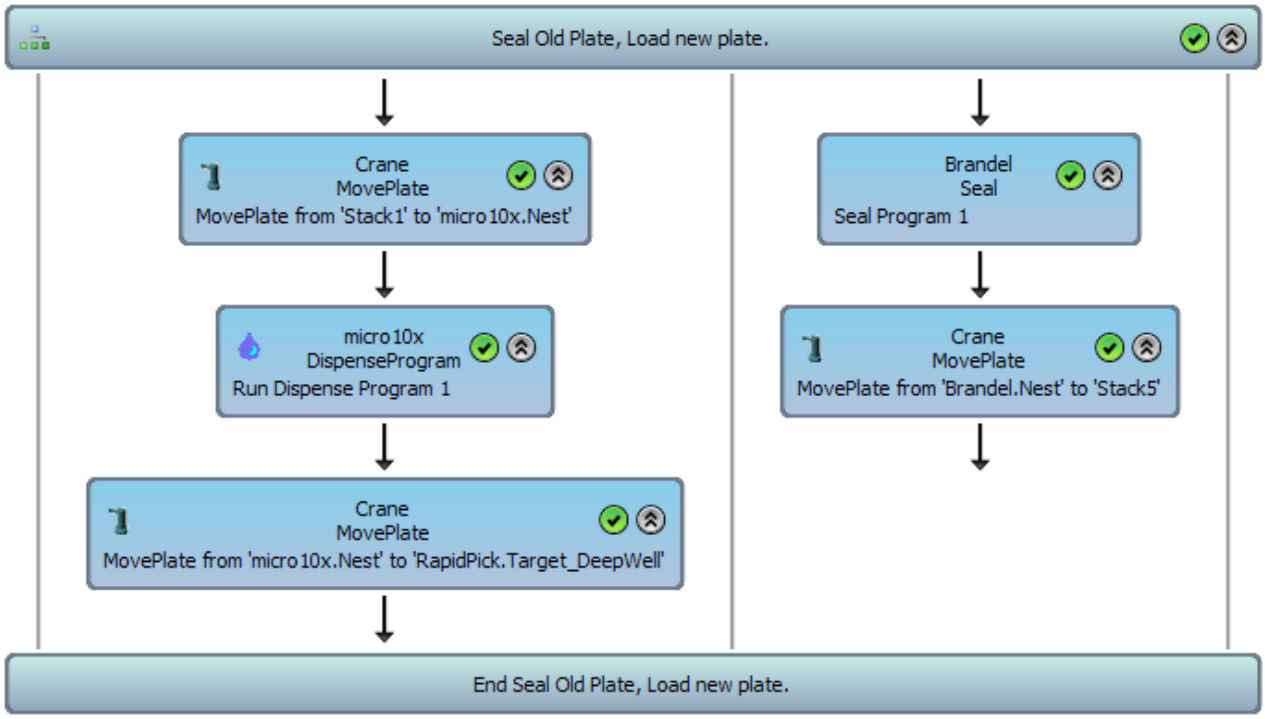
The steps of preparing the next plate should be the same as in the initial plate setup region:

- Move a new target plate to the Micro 10.

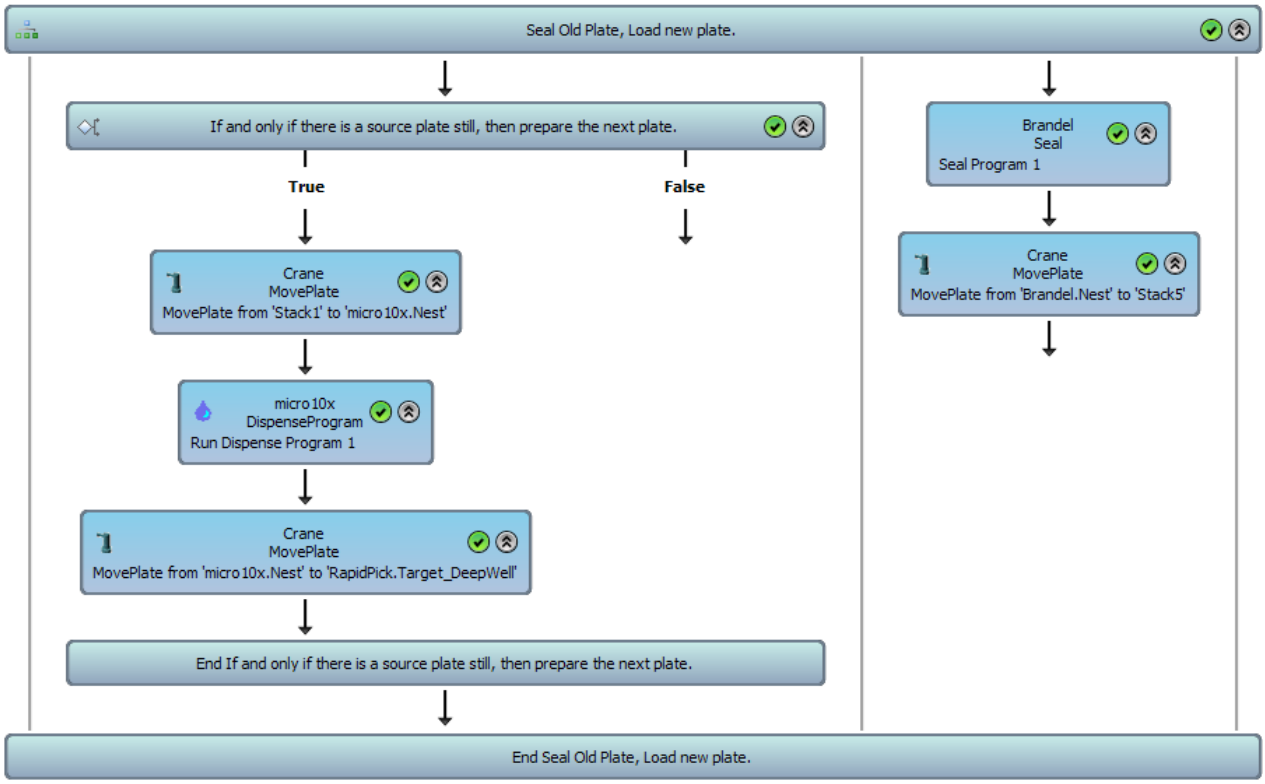
- Run a dispensing program on the new target plate.

- Move the new target plate to the Rapid Pick.

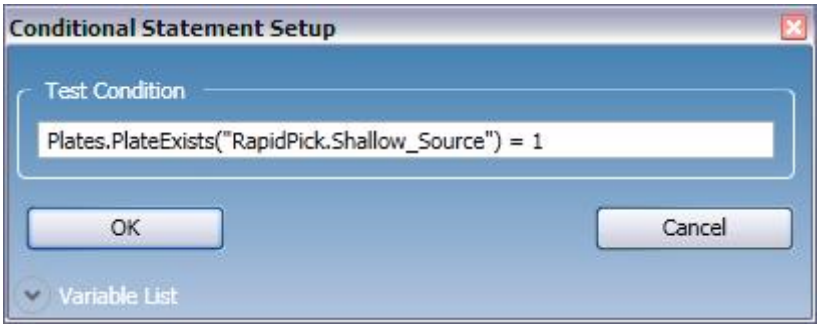
As these are two separate processes, we can split them in a parallel process, as shown here:



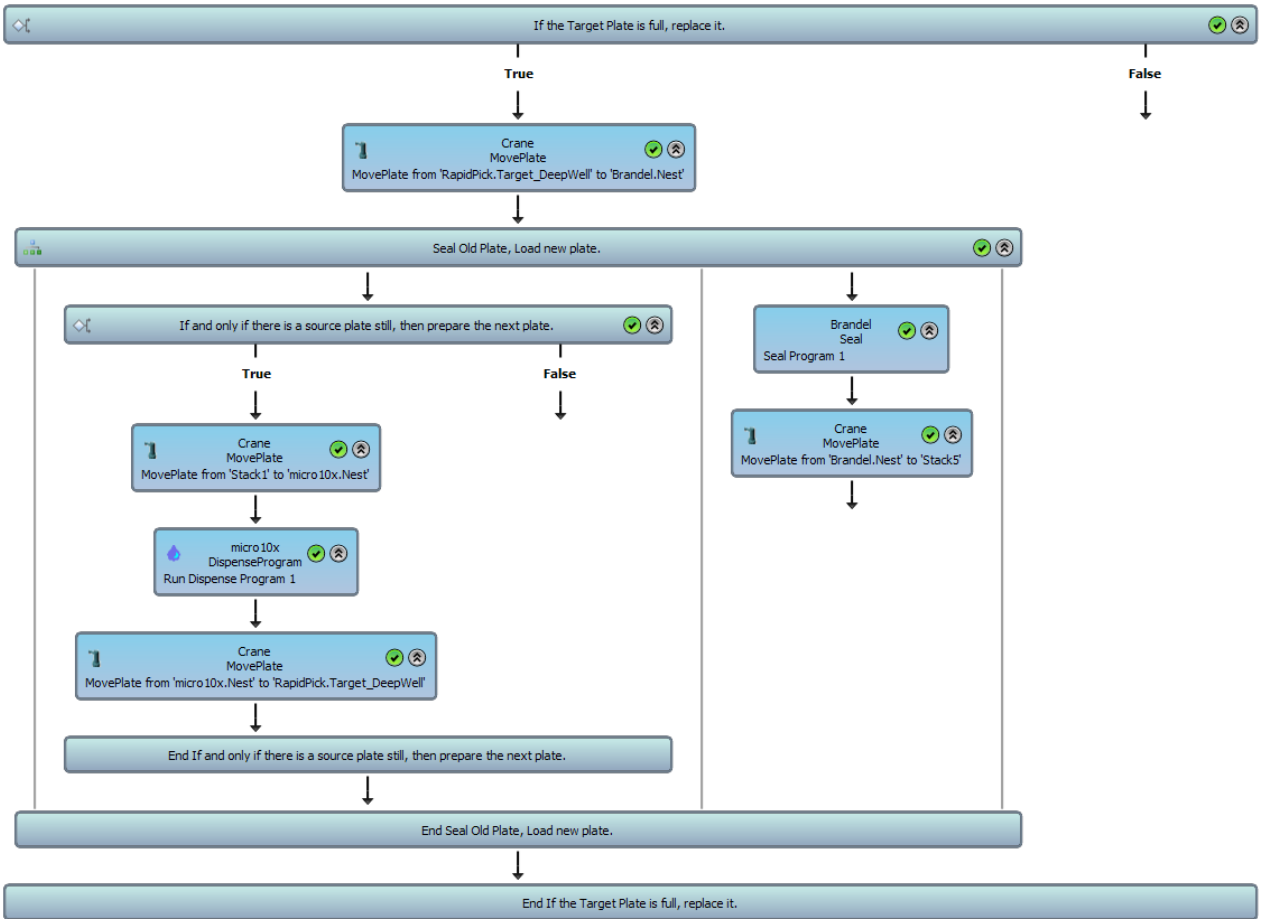
However, we only want to prepare the next plate, if a source plate is still present on the Rapid Pick. If the source plate has been removed, and not replaced from the previous conditional statement, we will not prepare a new plate. Since this is a decision, we need to place a conditional statement in the new plate process branch, like this:



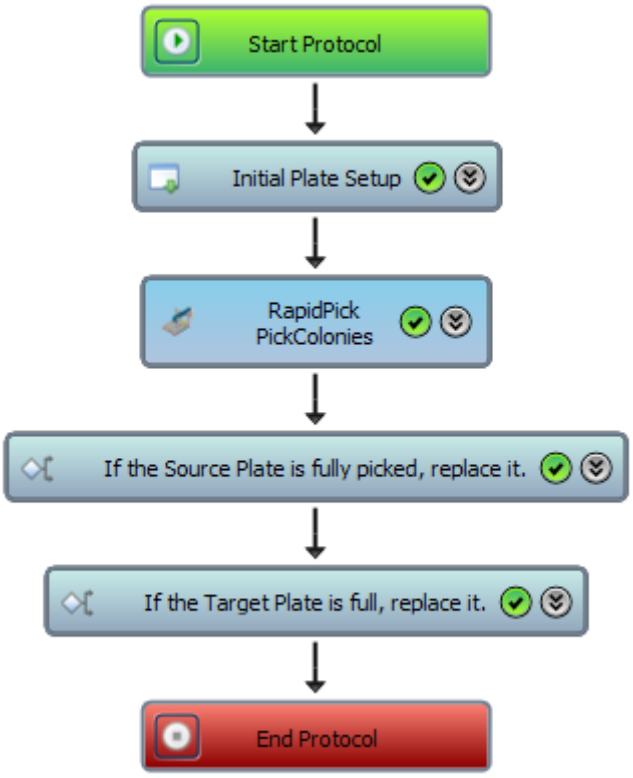
To detect a plate at a specific location, we use the statement Plates.PlateExists("X"), where X is the name of the nest where the plate is located. By typing the condition below, it will only prepare a new plate if a source plate is ready to go on the Rapid Pick source nest.



The full "Replace Target Plate" conditional statement, expanded, should look like this:



So, for a protocol, we currently have the following:



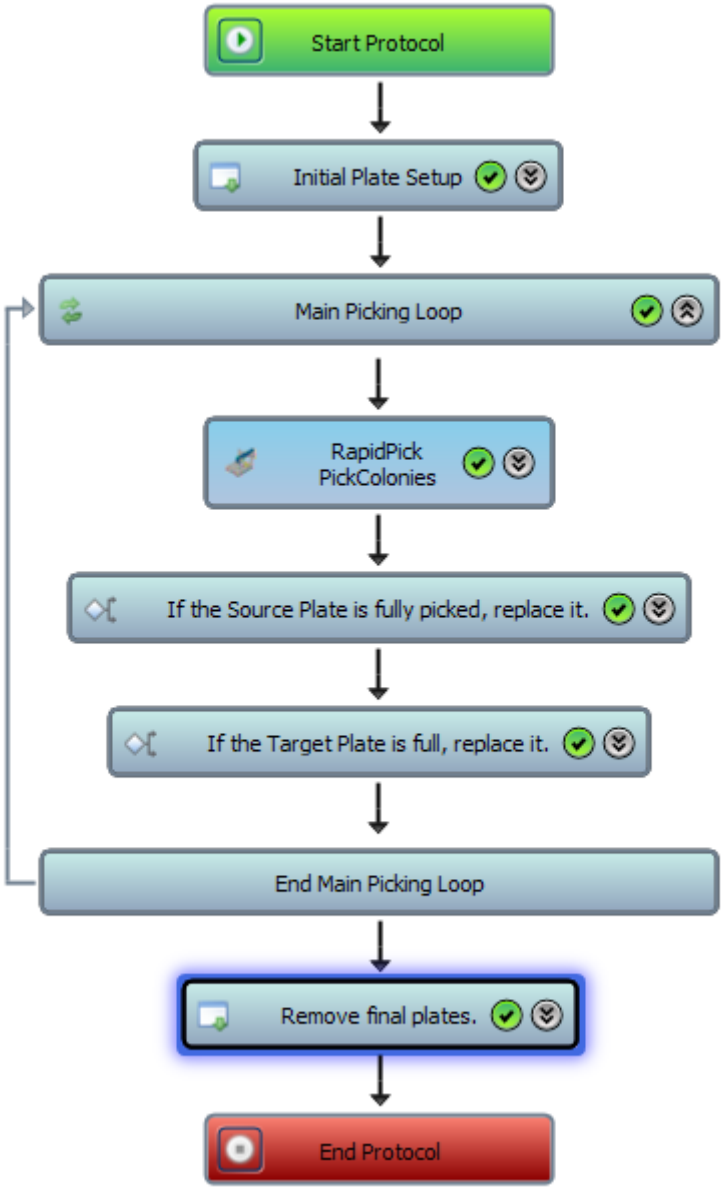
This may work for one plate. However, we want the protocol to repeat the picking and replace plate steps multiple times. We can place the last three items in a loop. We only want to run the pick step if there are plates in the source and target nests of the RapidPick.

To check that in the loop, enter the following statement in the "Loop While..." textbox:

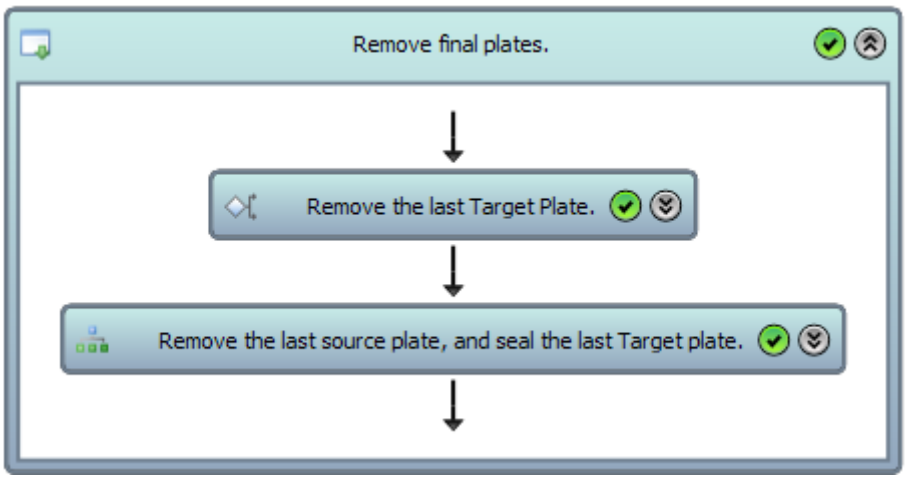
(Plates.PlateExists("RapidPick.Target_DeepWell") = 1) And (Plates.PlateExists("RapidPick.Source_Shallow")=1)

If there are not, then we know that we are either out of source plates or out of target plates. At this point, we must end the loop and initiate a cleanup routine, as we know we are at the end of the protocol.

2c. Final Cleanup

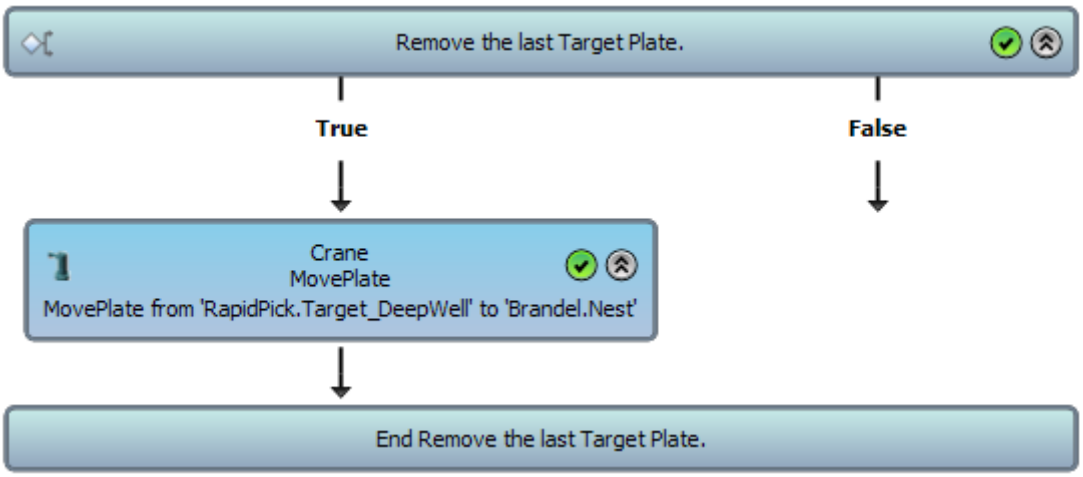


The last step, "Remove final plates", is a region activity with processes on removing the last target plate, and the last source plate. Place a region activity after the colony pick loop.

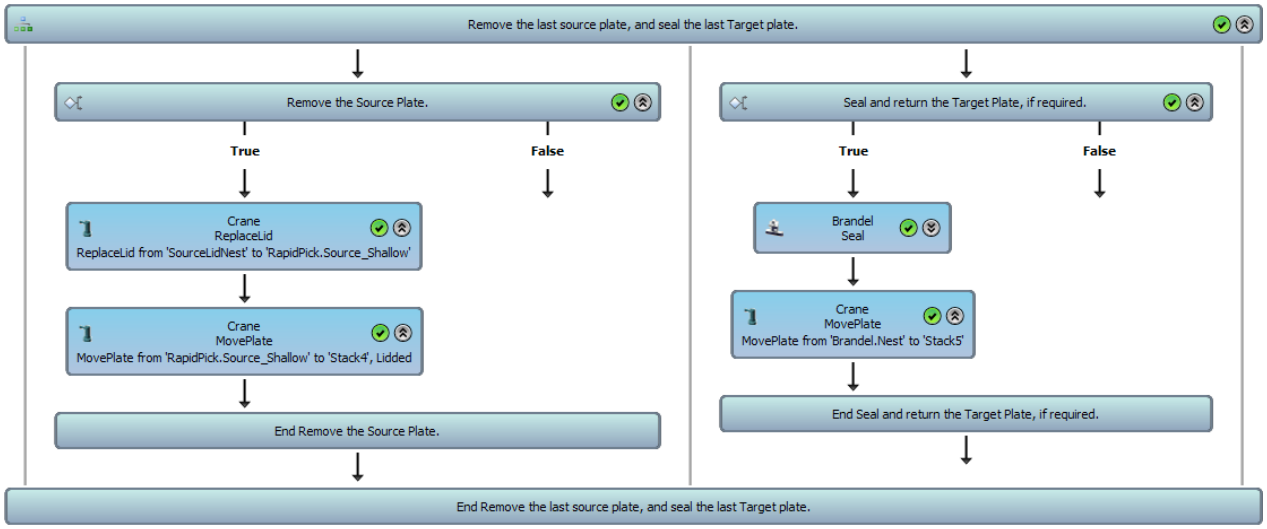


We will then check to see if there is a target plate that needs to be removed. Use a conditional statement, which tests the statement below:

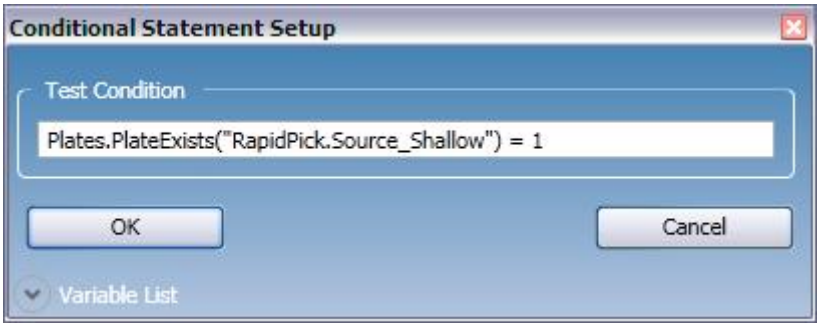
Then, place in the true branch, a move plate step from the target nest of the RapidPick to the plate sealer.



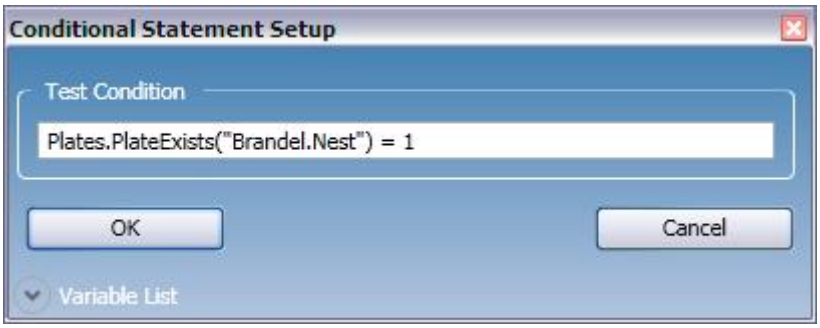
Finally, we can simultaneously check to see if there is a plate to seal, and check to see if the source plate has to be removed. Simultaneous actions require the use of the parallel process step. Using a conditional statement in each branch, we can check for a source plate to remove and a target plate to seal.



Like the previous steps beforehand, check for a source plate using the following:



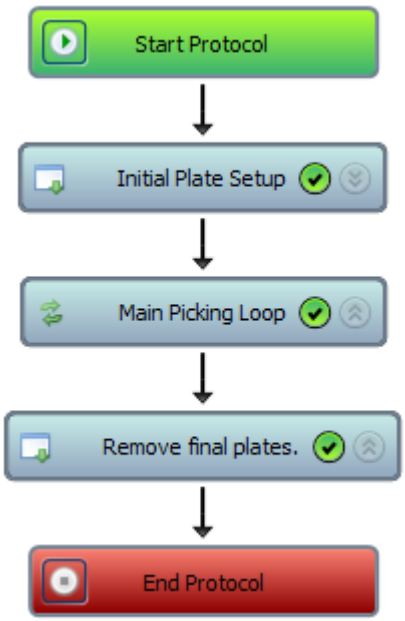
Check for a target plate in the sealer using the following:



3. Test the protocol.

Finally, once this part is done, **test** the protocol using empty plates and no liquid. Ensure that all of your source plates and destination plates are defined, and all initial plate locations match those used by this protocol. By testing the protocol, you will ensure that the protocol is written properly and no media is wasted on a faulty run.

The final protocol can condense to this:



This concludes the advanced protocol tutorial.

SoftLinx GO Wizard

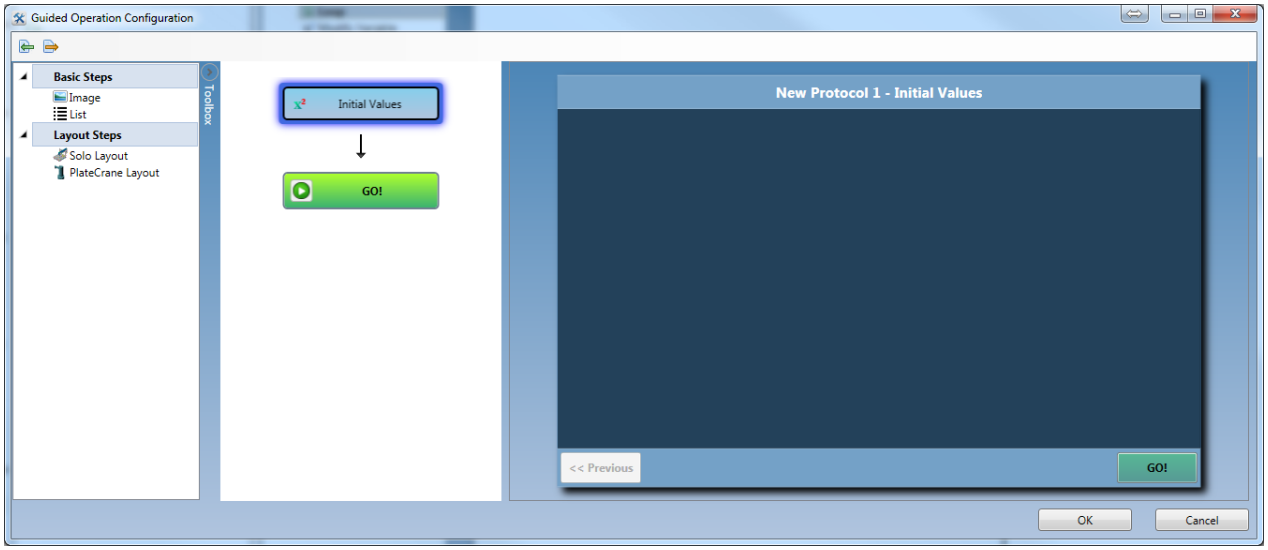
The SoftLinx GO (Guided Operation) wizard allows a user to add illustrations to a protocol prior to execution when run in the SoftLinx player.

The GO wizard editor is accessible within the SoftLinx Protocol Builder, or via the shortcut shown above.

GO Configuration access

All protocols made in the SoftLinx protocol builder can also have a guided operation (GO) protocol attached to them. To access the GO builder for a specific protocol, click on the GO icon located in the Protocol Builder with the selected protocol:

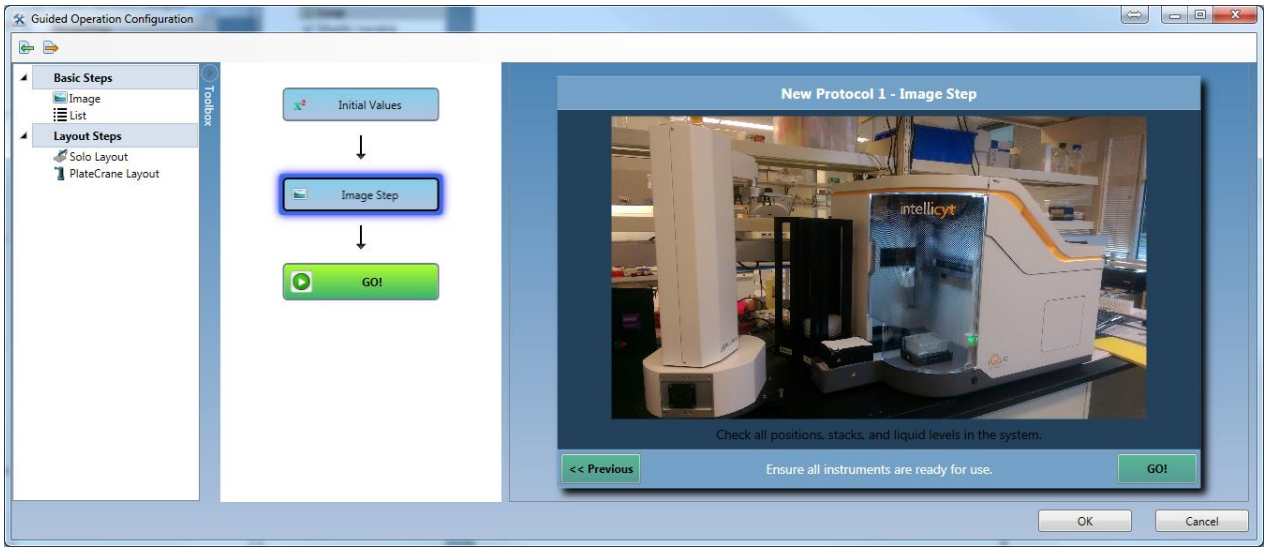
This will open the following window:



Much like a SoftLinx protocol, users drag and drop steps onto the center window to create a GO protocol, which will be shown to users when running the system through the SoftLinx Player.

Image Step

Image steps allow users to supplement their GO protocol with images to guide the user. The image step can be placed onto the protocol to produce the following:



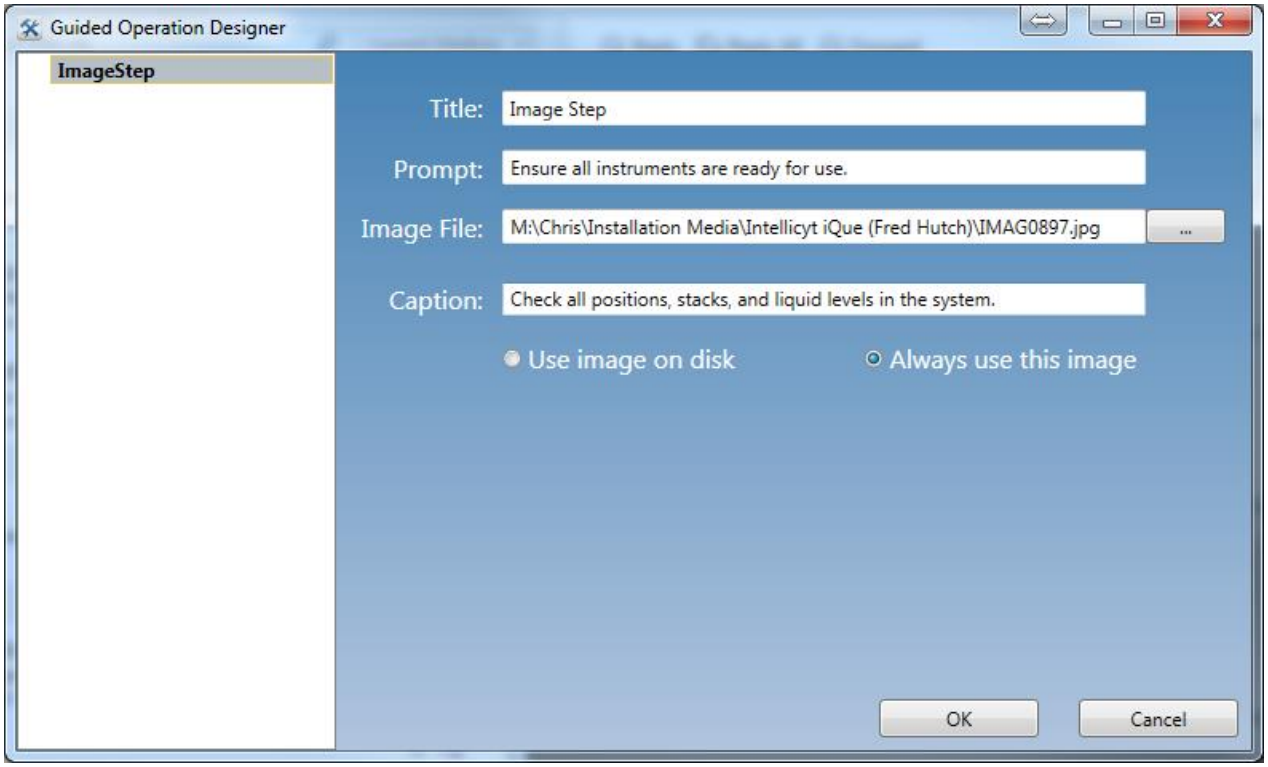
When placed, a parameters window will appear. The following items can be added:

Title: Top text placed above the image.

Prompt: Lower text placed between the Previous/Next/Go buttons.

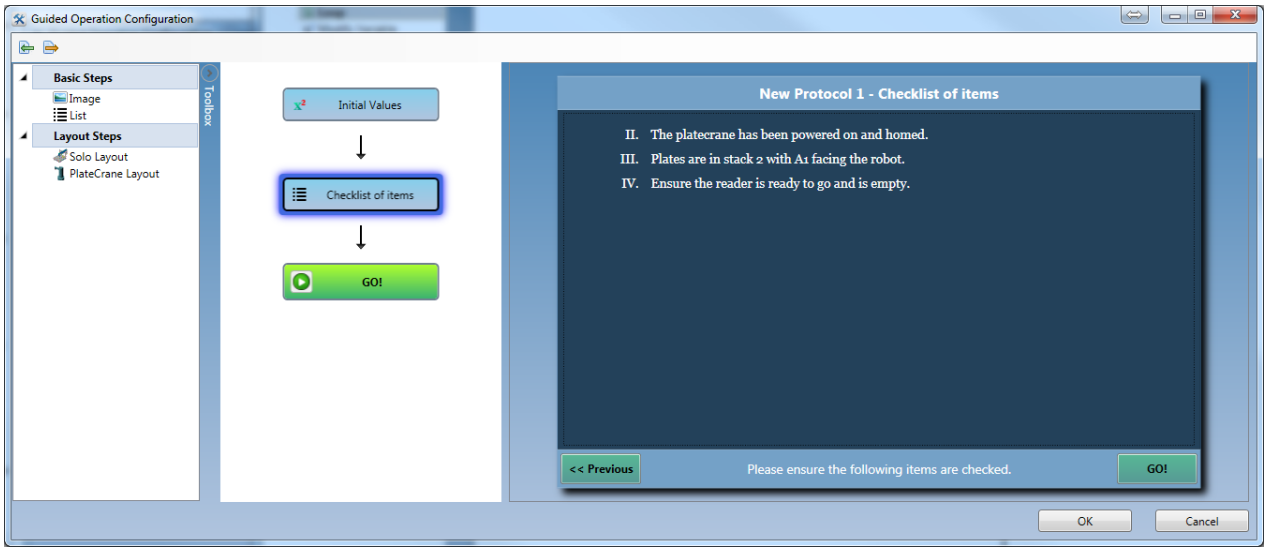
Image File: Location of the image to be used. Images can either be saved immediately to the protocol (Always use this specific image), or can be loaded every time the GO protocol is loaded (Use image on disk).

Caption: Text placed directly beneath the image.



List Step

List steps allow users to supplement their GO protocol with a list to guide the user. The list step can be placed onto the protocol to produce the following:



When placed, a parameters window will appear. The following items can be added:

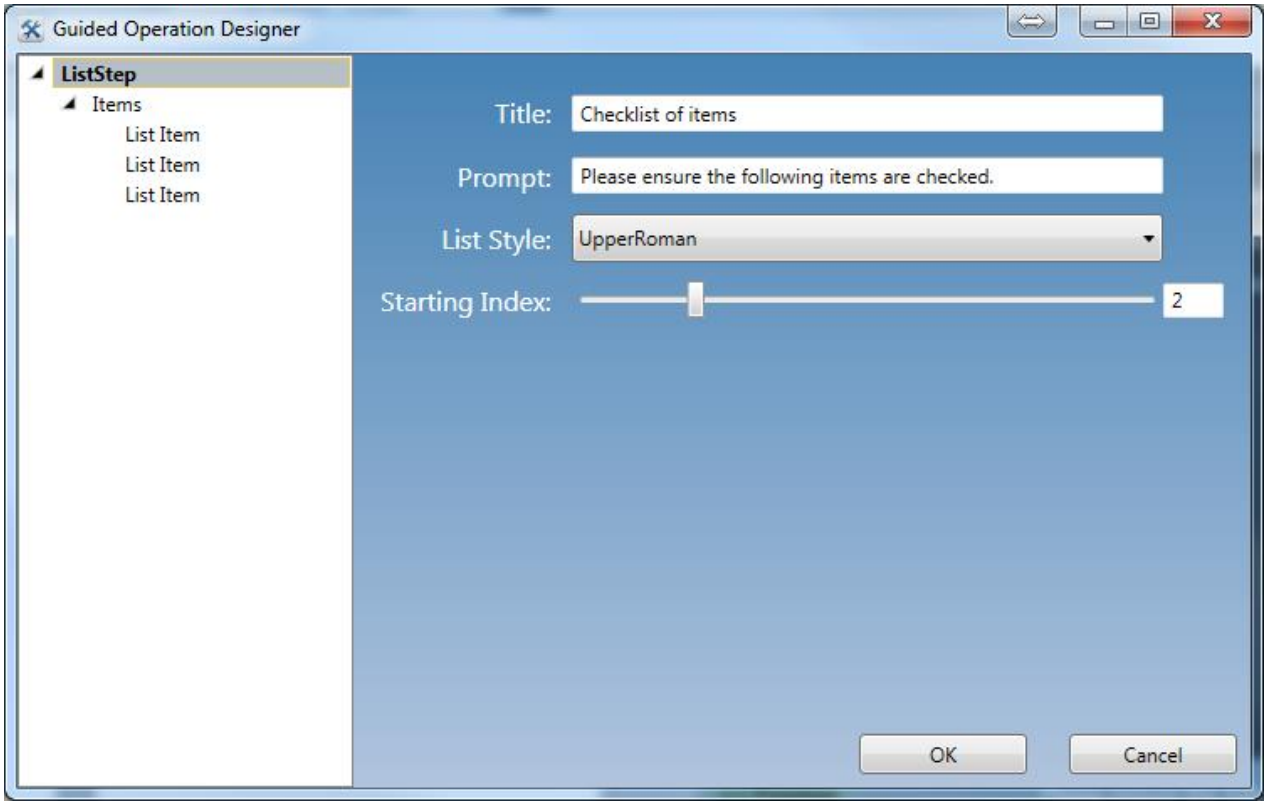
Title: Top text placed above the list.

Prompt: Lower text placed between the Previous/Next/Go buttons.

List Style: Type of bullets used in the list. Multiple styles are available.

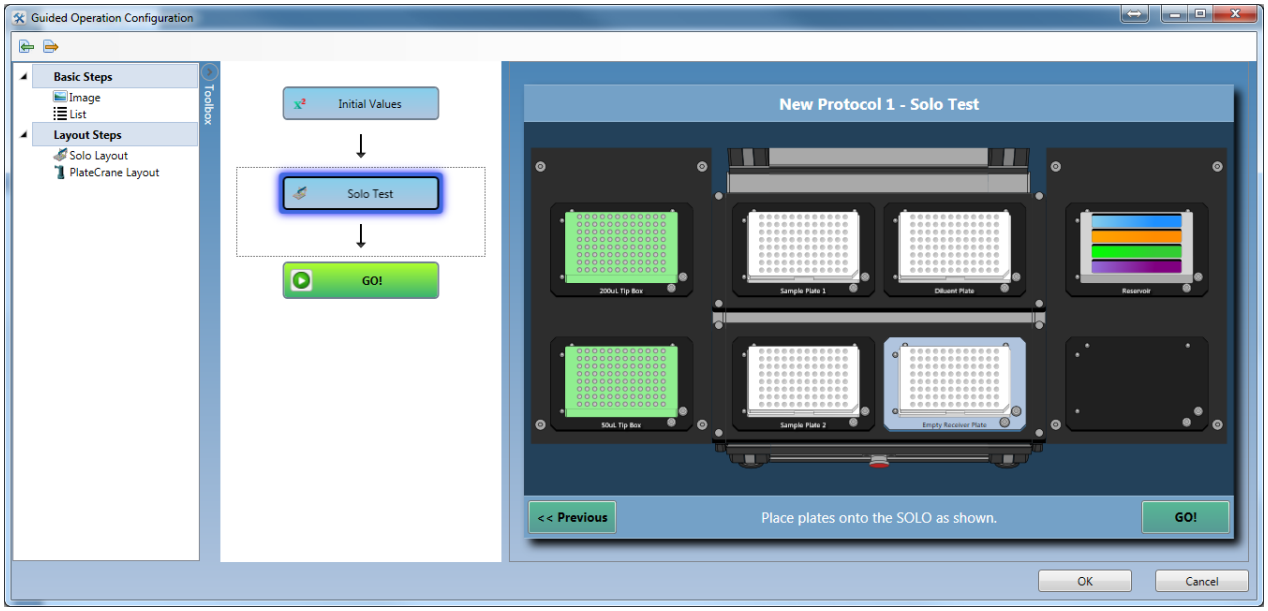
Starting Index: Starting number of the list, if a number or letter type list is used.

In addition, clicking "Items" will allow the user to add List items. List items can then be modified to display the text per item in the list.

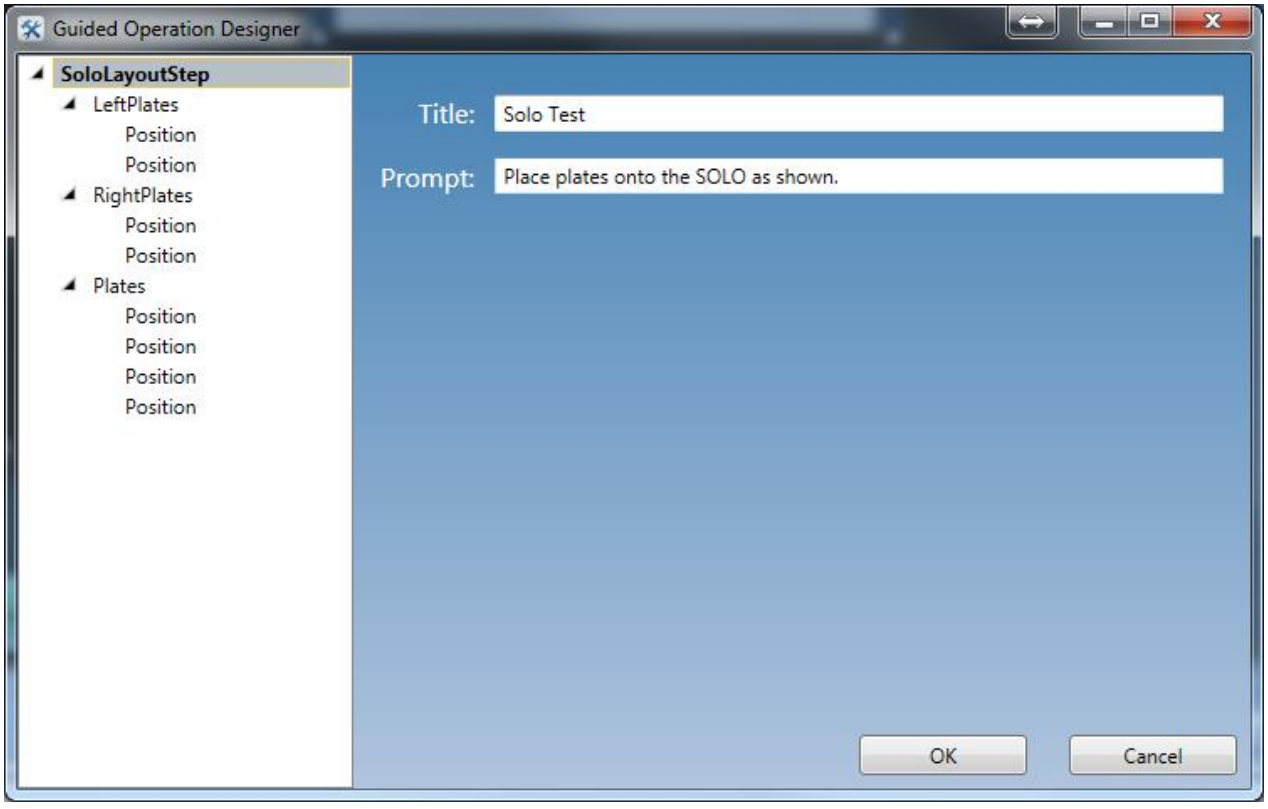


Solo Layout Step

Solo Layout Steps allow for diagrams to be drawn that represent the Solo with its plate locations and sidecars.



When a step is placed, the following window appears:



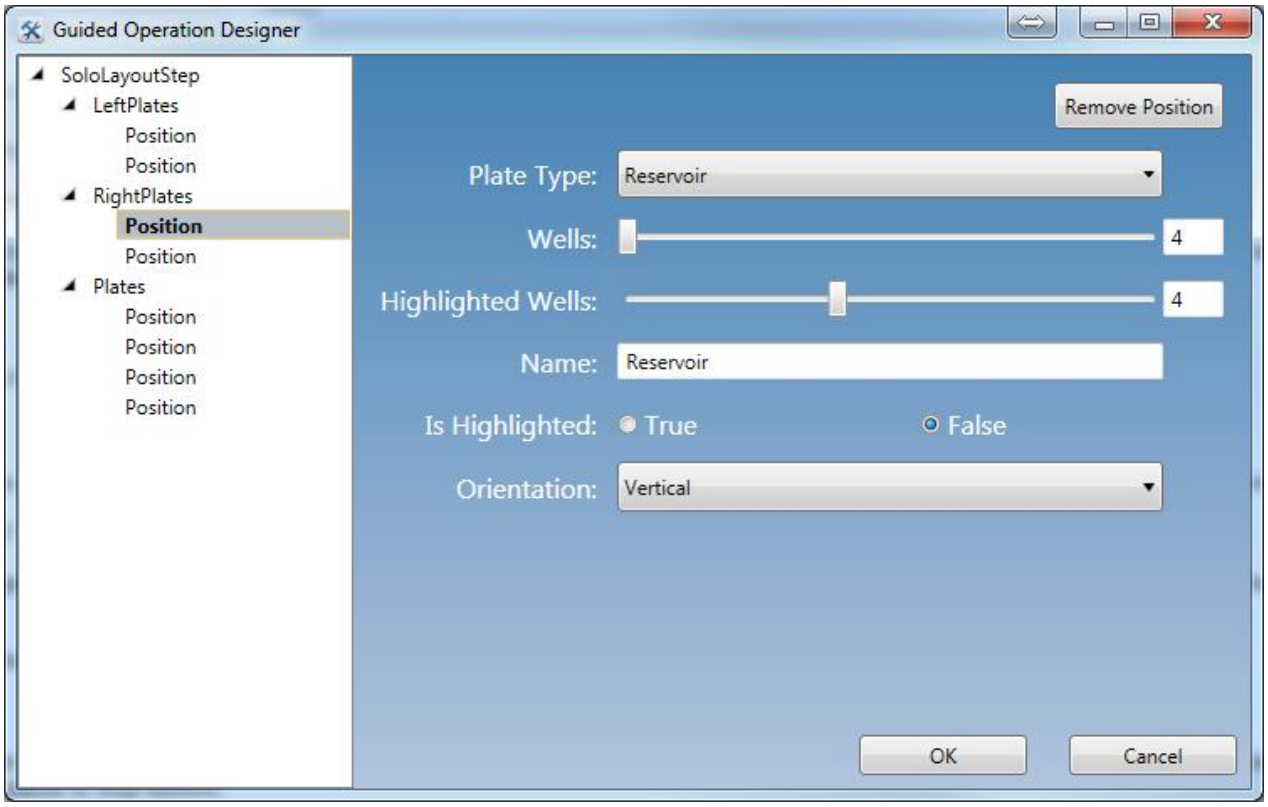
The selectable items are as follows:

Title: Top text placed above the image.

Prompt: Lower text placed between the Previous/Next/Go buttons.

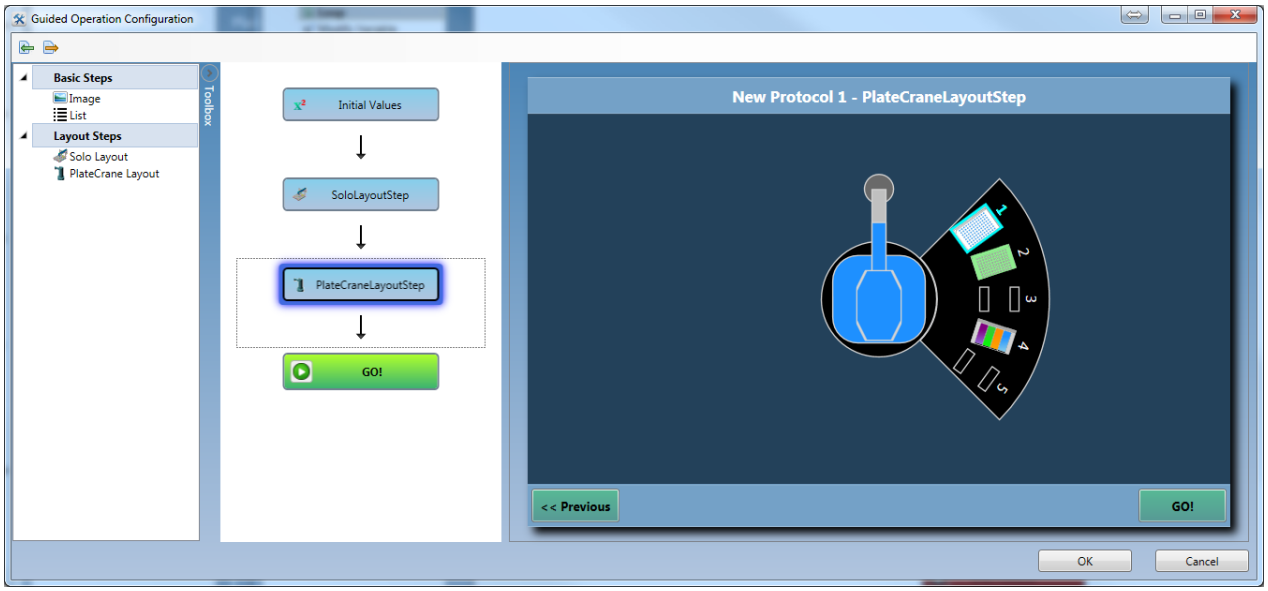
Also, you may click on the LeftPlates, RightPlates, or Plates items to add positions to the deck. Plates represents the positions on the main deck, and should have 4 plates. The LeftPlates and RightPlates represent the optional sidecars.

Once plates are added, define the plates using the window below. Tip boxes are green-colored plates. Microplates are white. Reservoirs are dark gray, and should have the number of wells defined.

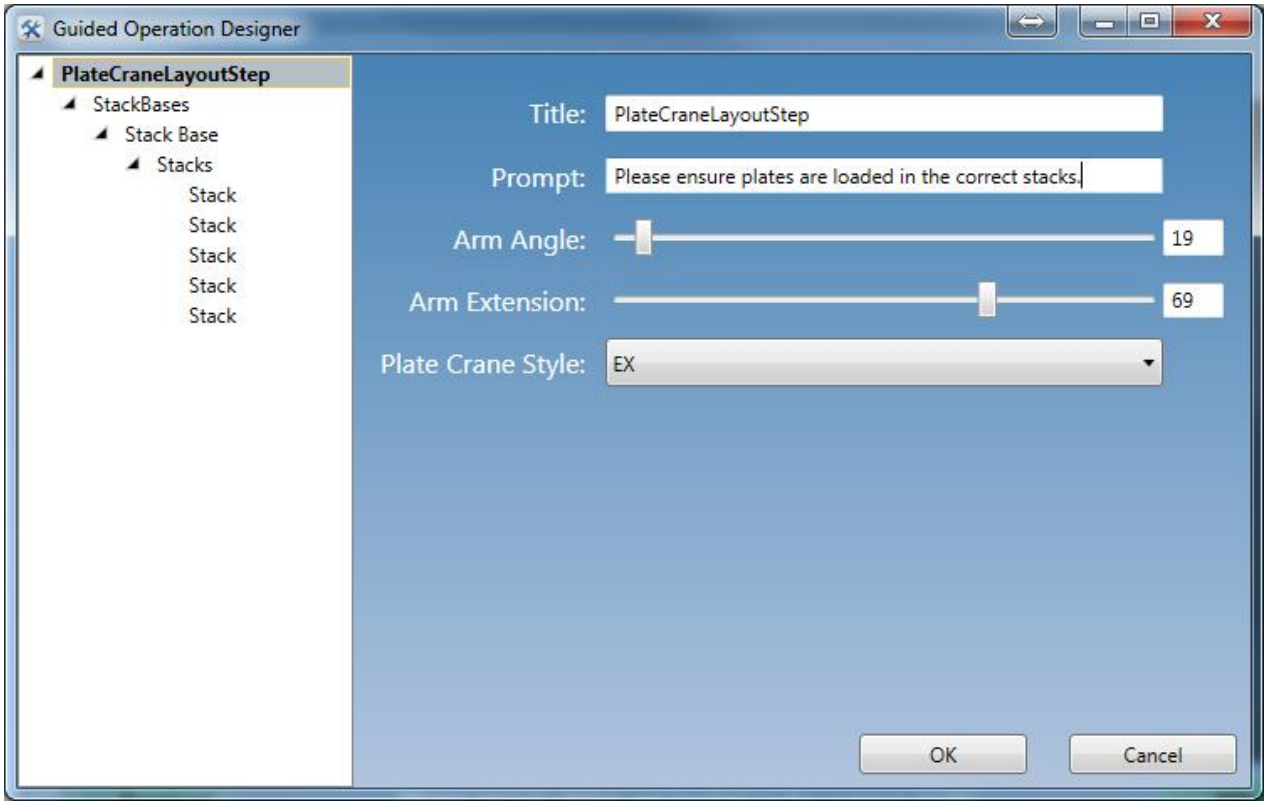


Platecrane Layout Step

Platecrane Layout Steps allow for diagrams to be drawn that represent the Platecrane, input plates, and stacks with stackbases.



When a step is placed, the following window appears:



The selectable items are as follows:

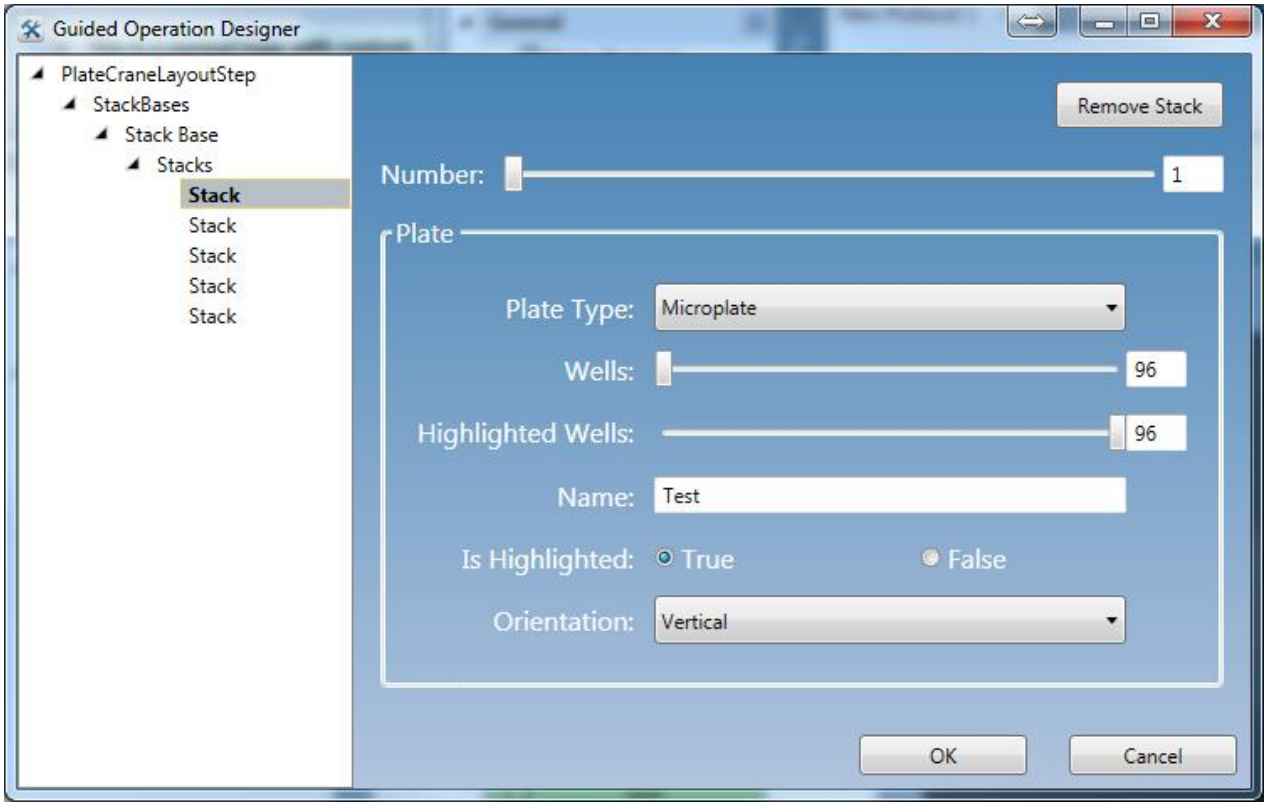
Title: Top text placed above the image.

Prompt: Lower text placed between the Previous/Next/Go buttons.

Image File: Location of the image to be used. Images can either be saved immediately to the protocol (Always use this specific image), or can be loaded every time the GO protocol is loaded (Use image on disk).

Caption: Text placed directly beneath the image.

In addition, stacks can be added and populated with plates. A number can be assigned to a stack. The plate type can also be assigned, with number of wells, plate name, orientation, and highlights.



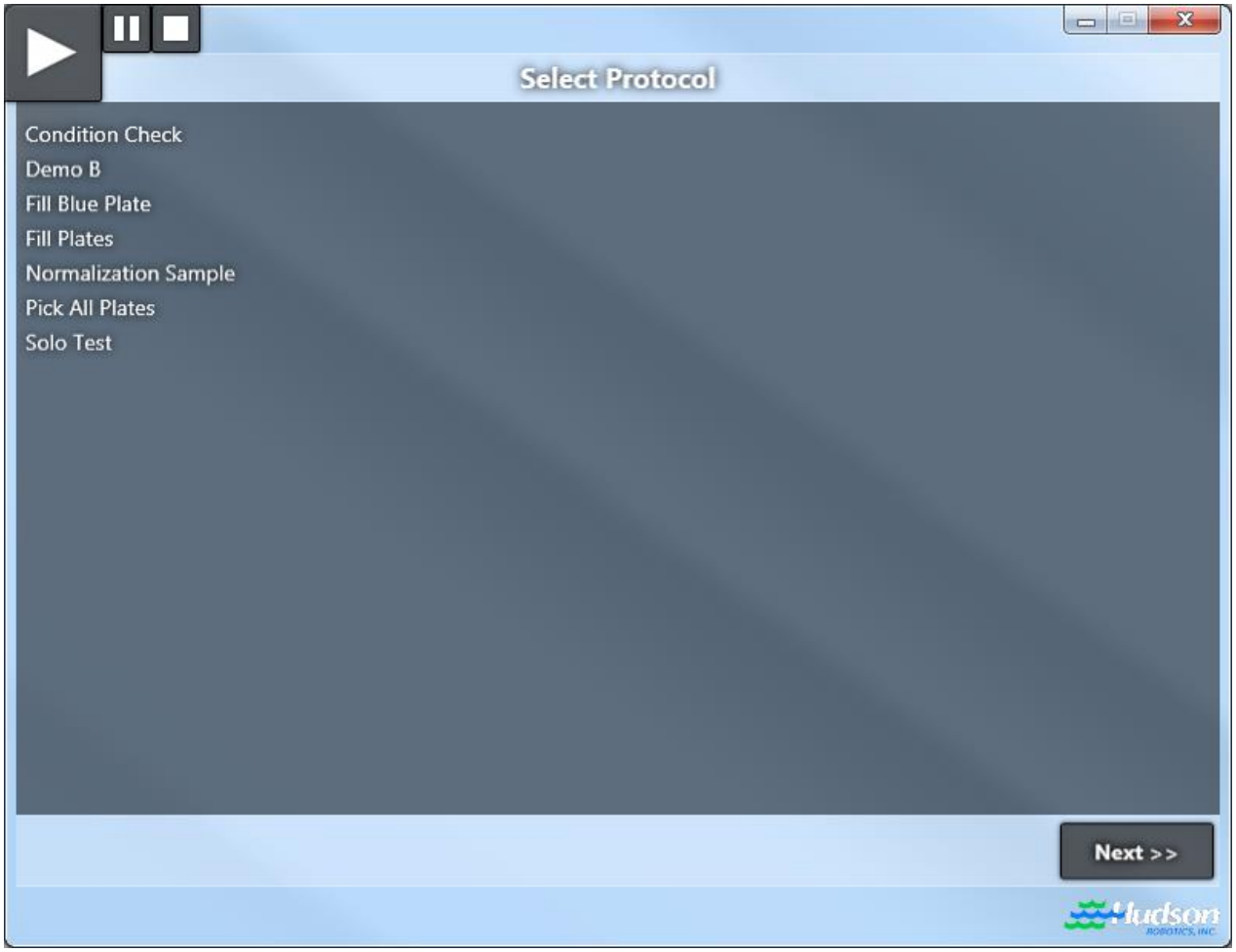
SoftLinx Player

The player allows the user to quickly run protocols without all of the overhead and UI that the builder provides. This allows teams to protect the protocols from editing, and can provide all users with a UI to assist in setting up their work cells prior to protocol execution.

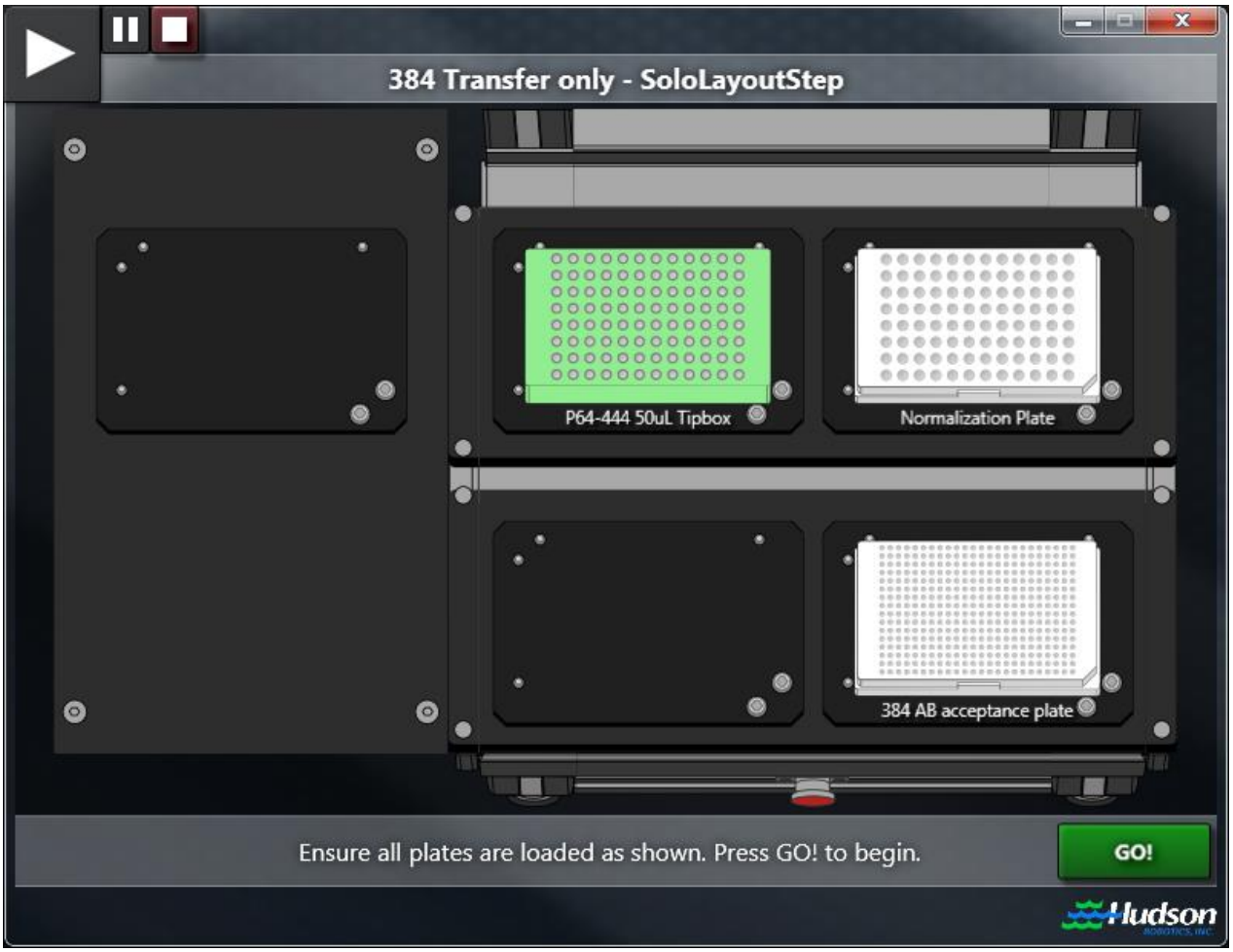
Protocol Selection

When opening the SoftLinx player, a list of protocols will be shown. On most systems, this list is populated from all protocols that sit in the current user's Documents folder.

This folder can be changed to another location. Contact Hudson Robotics to change the folder.

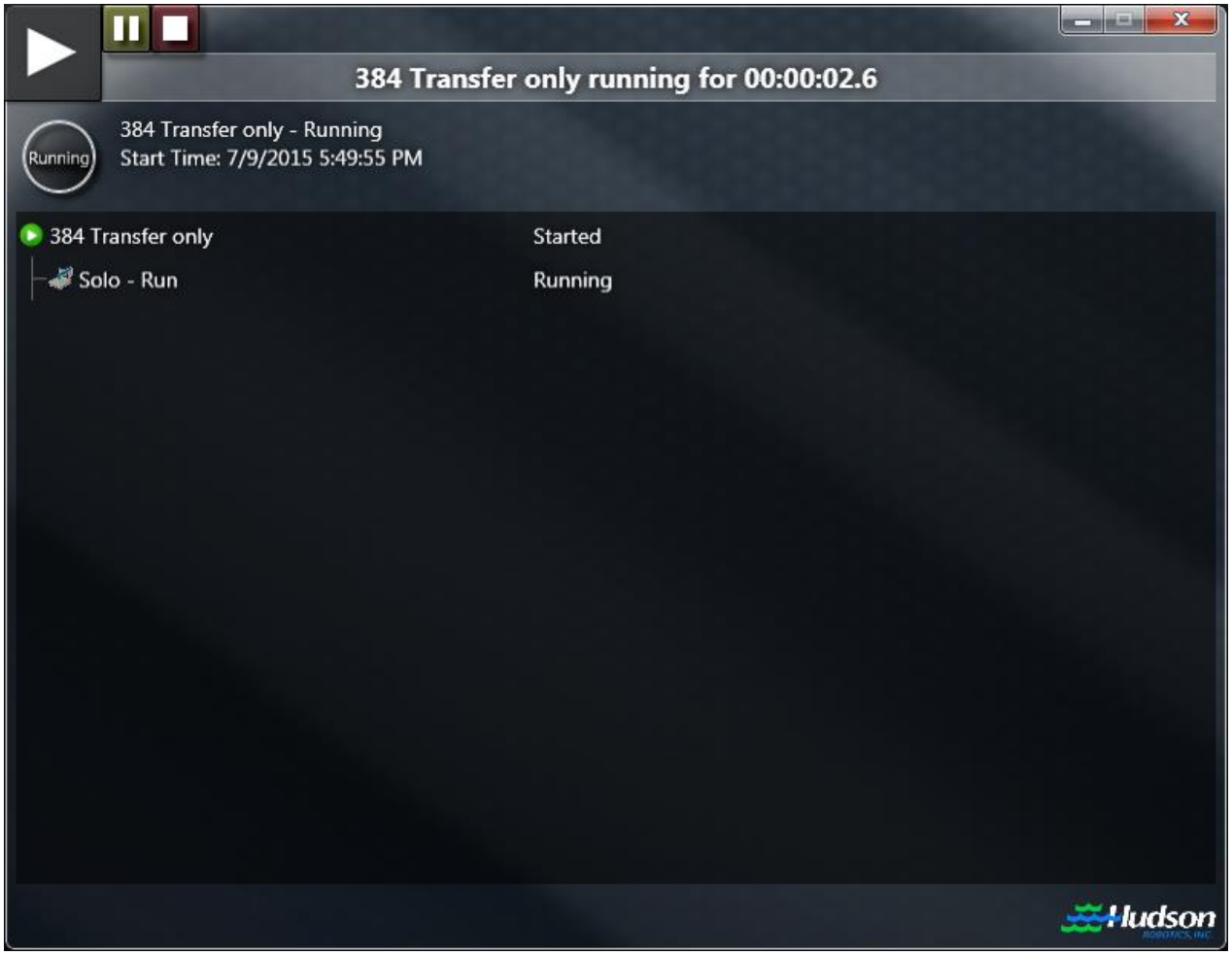


Select the protocol to run, and hit next. If the protocol has a wizard, the prompts will show. Otherwise, the protocol will be verified to exist and the protocol builder will prompt the user to hit GO! to start. An example is below.



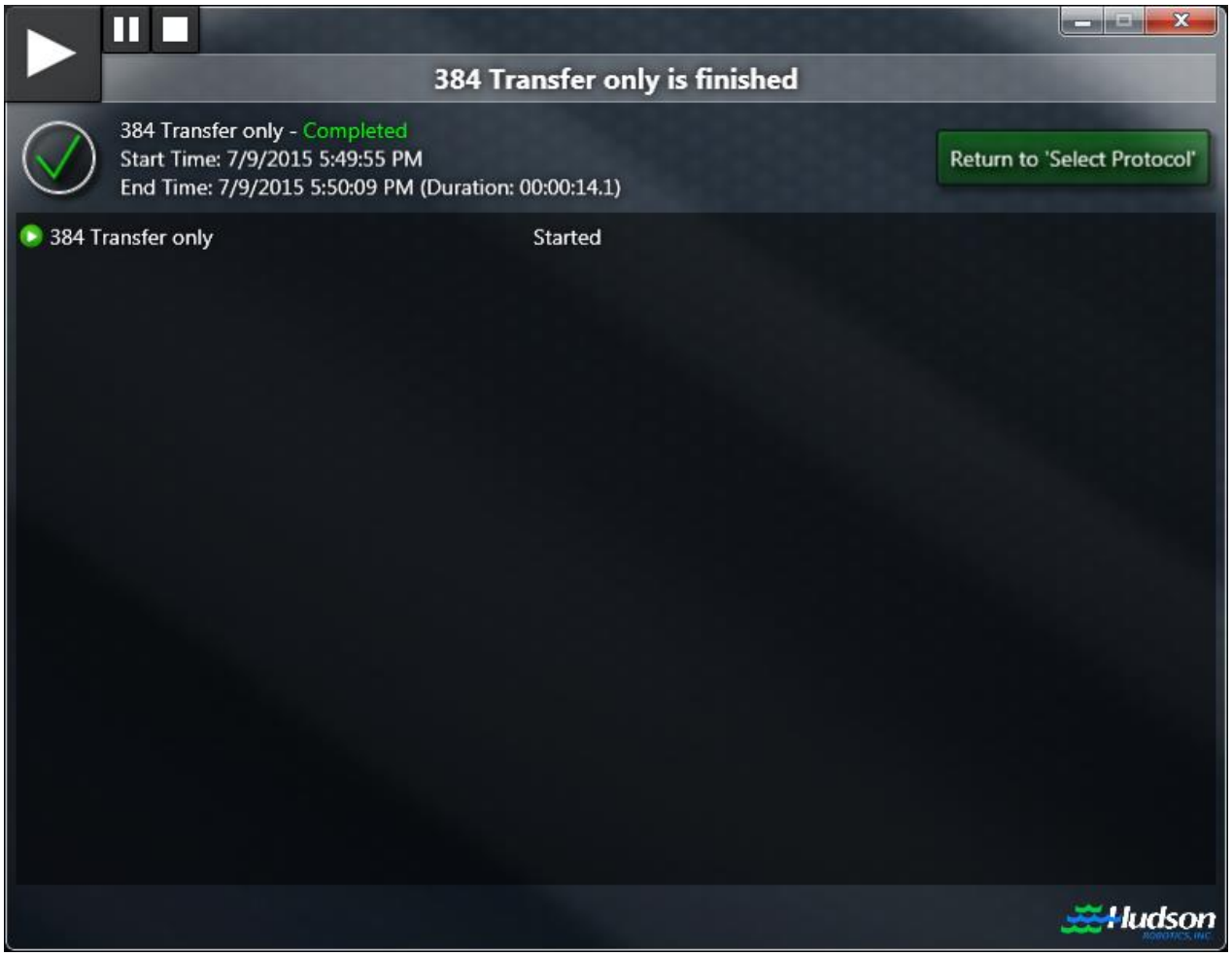
Protocol Execution

When the protocol is running, the following status screen will be shown.



Protocol title, duration, protocol status, and the current steps that are running will be shown. The pause and stop buttons are active. Click those to pause or stop the protocol.

When completed, the status window will look like this:



Other statuses that can appear:

Starting - The protocol is initializing all instruments to begin.

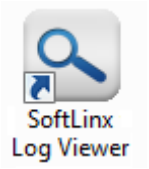
Running - Protocol is active.

Completed - The protocol finished without user intervention or errors.

Aborted - The protocol was cancelled by the system.

Cancelled - The protocol was stopped by the user.

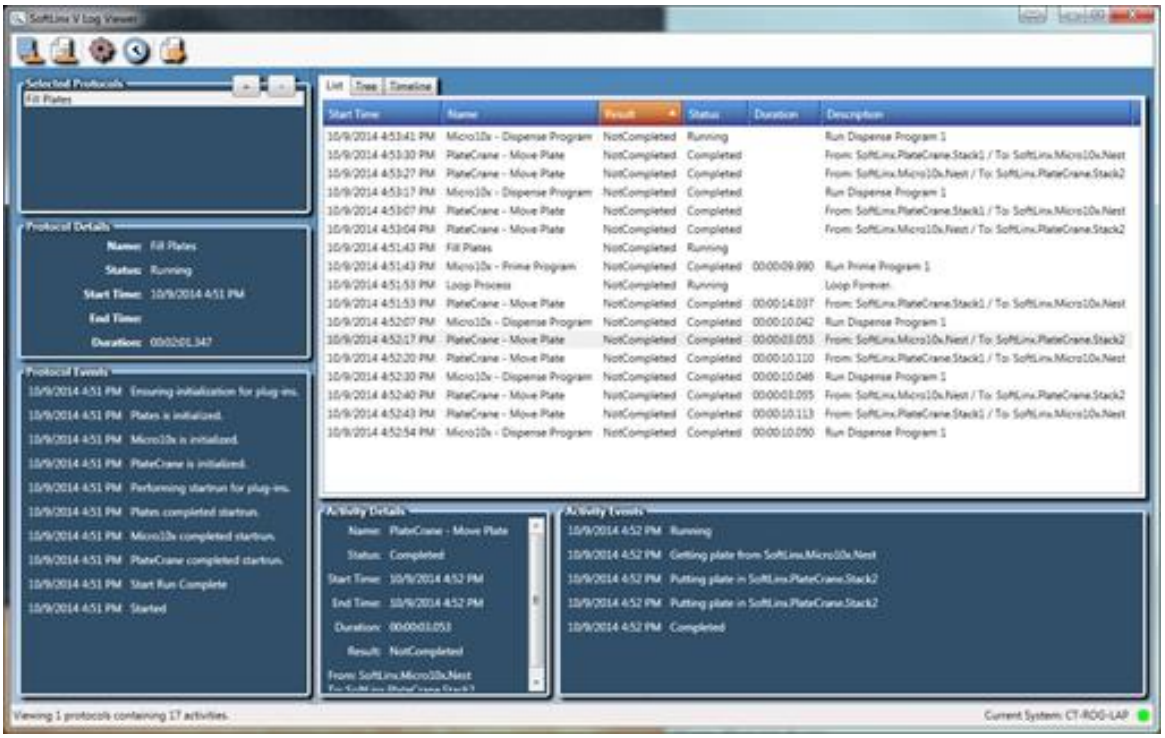
SoftLinx Log Viewer



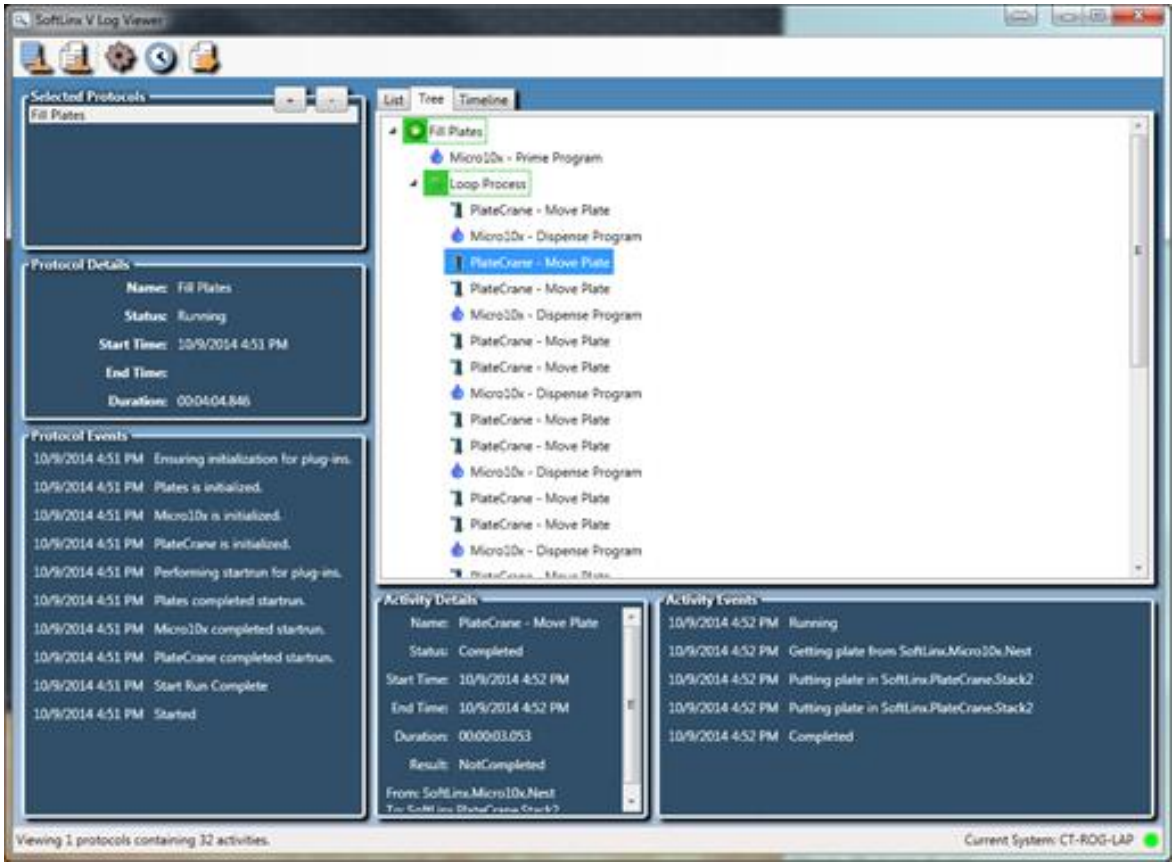
Log Viewer

The log viewer allows a user to access information about each time a protocol has run on the machine. This log shows every step, its result, duration, execution time, and description, as well as any internal messages that it has sent. All of this can be exported to Hudson for further support.

There are 3 views. The list view shows every step run in chronological order, its result, duration, and description.



The tree view shows every step that was activated, in chronological order, and by parent activity (such as loops).



The Timeline is a Gantt-chart visualization of the protocol when it was run. The timeline can be dragged to the point in time in which activities were run.