GREEN BUTTON GO™

Automation Scheduling Software

USER MANUAL

BIOSERO

Table of Contents

Chapter 1. Introduction and Quick Start

Biosero's Green Button Go™ automation scheduling software is designed to simplify the automation process and assist the user in developing integrated workflows on a system. The software is equipped with basic programming features and functions (i.e., loops, conditional branches, message boxes, etc.), and also has the ability to create sub-processes and to employ advanced error handling techniques. It employs plugins and drivers to control the instruments, databases, and resources needed for individual integrations.

Green Button Go (GBG) has been specifically optimized to facilitate rapid method development. The design process itself has been automated to the point that "one-off" methods are truly practical.

1.1. Running GBG

In GBG, the typical user will select the desired project by clicking on the appropriate tile from the Start screen. The user will then select the Run tab and from there click the **Play** button (Figure 1).

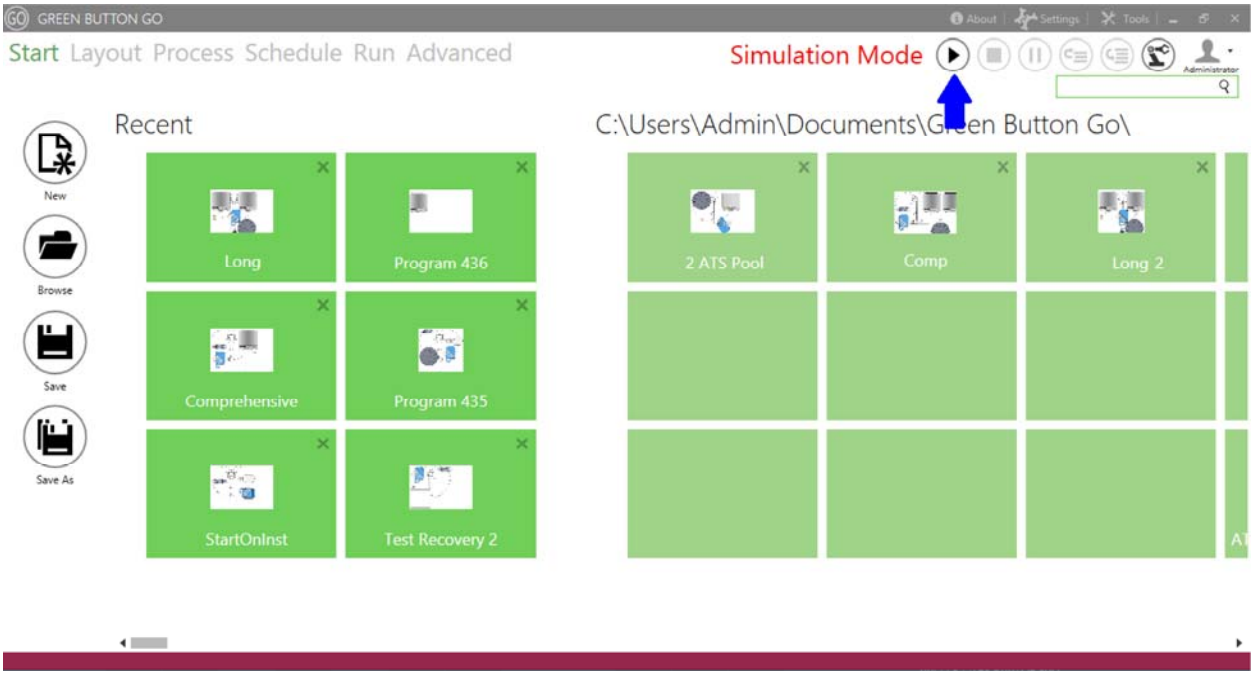


FIGURE 1. START SCREEN

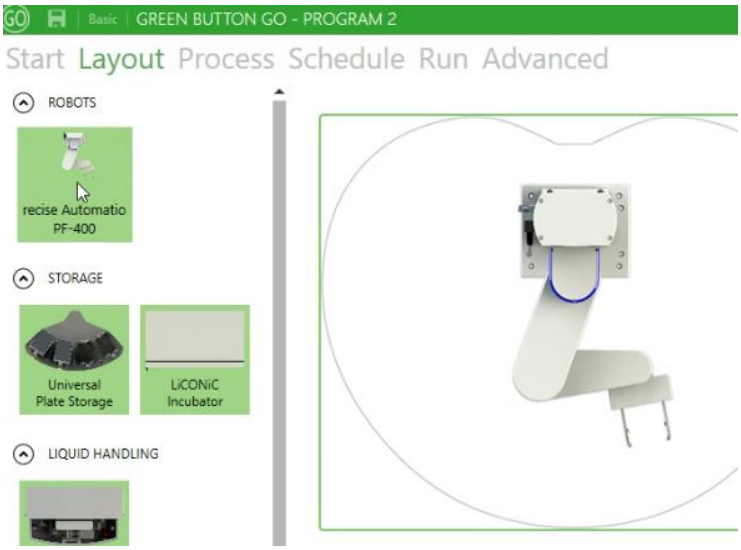
1.2. Designing and Running Your First Method

In GBG, method development is meant to be as automated as possible. The underlying philosophy is that the broad concepts will be laid out in the Process tab, but that customizations are available through the Advanced tab. The interaction here is between *processes*, which are broad declarative tools, and *procedures*, which are lower level tools. Processes automatically generate all procedures necessary to run an integration. Low-level interaction (with procedures) is only necessary if low-level customizations

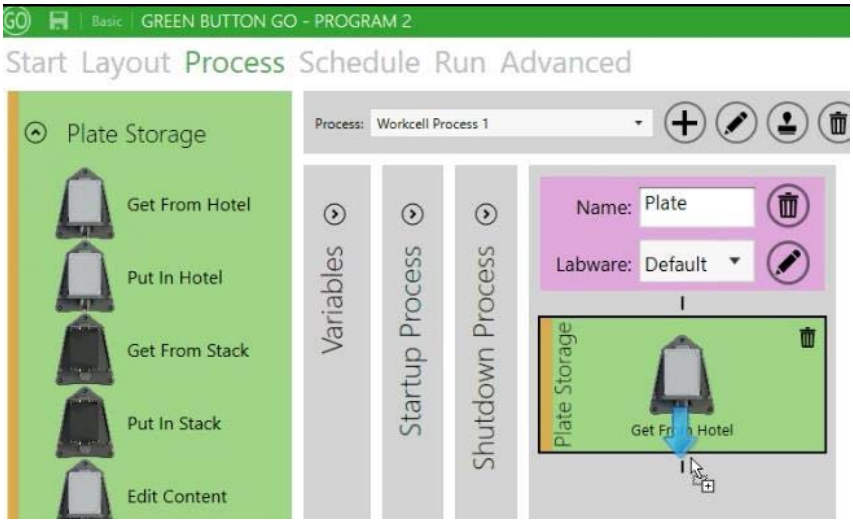
are desired. Note that details such as seal temperatures are set as process properties and do not require use of the Advanced tab. The design experience flows through the tabs from left to right.

1. Click **New** to create a new program.

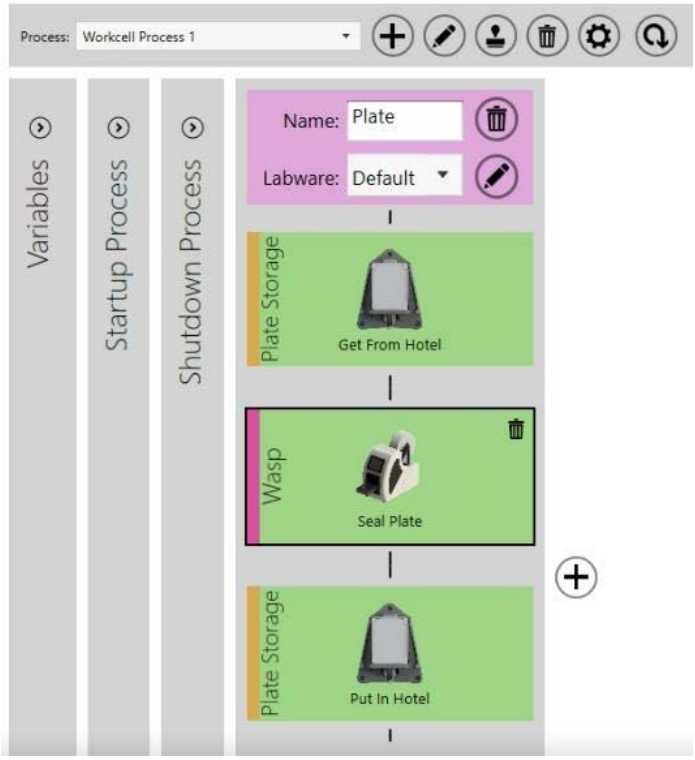
2. Click or drag to add instruments (arm, storage, function instruments such as liquid handler or sealer).



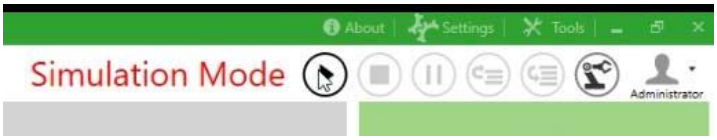
3. In the Process tab, set up a simple run.



4. Ensure basic ingredients are in place: Get plate from storage, perform one or more actions, place plate back in storage.



5. Click **Play** to execute the run. GBG will enter the Run tab to display runtime feedback.



Reference

For a quick tutorial on creating your first program, see the video *Your First GBG Program* on Biosero's user portal. (Contact your local Biosero representative for access.)

1.3. Definitions

TABLE 1. DEFINITIONS

Command	An executable action. Lowest level of logical tree.
Driver	One or more files used by GBG to control an instrument or occasionally another tool.
Drivers Folder	Drivers subfolder in the installation folder. For example, c:\Program Files (x86)\Green Button Go\Drivers. A driver must be in this folder to be recognized by GBG.
Instrument Process	Lowest level process. Auto-generates all necessary procedures to execute desired action.
Installation Folder	Folder where GBG installs. For example, c:\Program Files (x86)\Green Button Go.
Plate Process/Swim Lane	Defines the broad set of actions taken on each plate. If two stacks of plates experience a different set of actions, they should go through two different plate processes.
Plugin	Similar to a driver but typically adds functionality that does not include physical equipment. Examples: Communicate with a LIMS system, add a tool to the GBG tool menu, add a GBG command.
Procedure	A set of specifically ordered and executed commands.
Process	A broad set of actions which contains other processes and may auto-generate procedures.
Workcell Process	The broadest process. Will contain a startup and shutdown process as well as any number of plate processes.
Workflow Commands	Commands which control the execution of the program. A number of non-instrument-specific commands are available by default. Instrument-specific commands are added by the instrument to which they belong.

Chapter 2. Setup

2.1. Installation

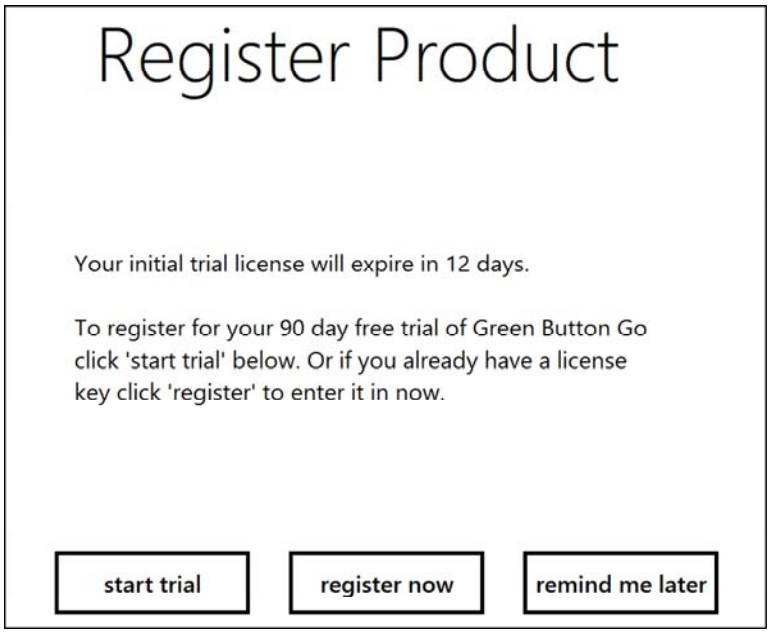
1. From the installation media provided to you, access the installer executable, for example:

   GBG Setup 2.1.6145.1009.exe

2. If it was downloaded, ensure it is unblocked (see Note: Unblocking DLLs, page 9)
3. Run it to install GBG. Depending on computer set up, GBG will typically install to c:\Program Files (x86)\Green Button Go 2017.

> **Note** Within this installation folder there are a number of important files and folders. One of these is the Drivers folder. Any driver file in this folder will be available within the program.

4. Register a license:
   a. Open GBG.
   b. A window will appear prompting you to register your copy.



   Note: The screenshot shown is on a computer where a trial has been started

   c. Follow the on-screen prompts to download either a trial or full license.
      - If a license key has been provided by Biosero, click the **register now** button to download a full license.
      - If the computer is not connected to the internet, click the **register now** button and then select manual registration to download the license separately and install it manually.
      - If you are starting a trial, click the **start trial** button.

2.2. Start Program

> **Note**
>
> GBG is typically installed with at least one sample program. Alternatively, follow the directions below to create a simple program.

Starting from within GBG

You can start a program by opening GBG and going to the Start tab. Here you will see programs available in the designated folder as well as recently opened programs. Click **Browse** to find programs elsewhere on your computer.

Starting from Program Folder

A program can be started directly by opening its folder and double-clicking the main program file, i.e., Program 1.gbg.

2.3. Installing Drivers

Defaults

GBG typically installs with a broad set of drivers. Installed drivers can be viewed within GBG in the Layout tab. GBG automatically loads any drivers placed within the installation's Drivers folder.

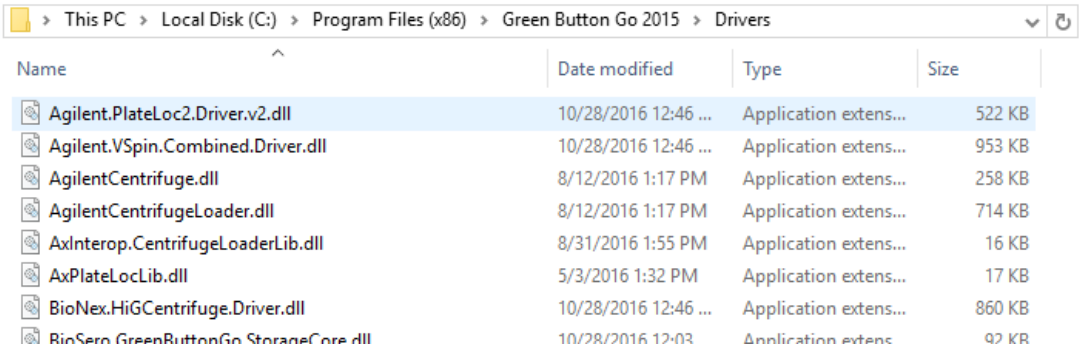


FIGURE 2. DRIVERS FOLDER

Desired Drivers and Archiving

Biosero typically sets up all necessary drivers, though occasionally an advanced user manipulates their driver set. If this is needed, use the techniques outlined below.

> **Note**
>
> If drivers or other DLLs are emailed, Windows may block them from execution. Put the DLL (or multiple DLLs) on the desktop, right-click it, go to **Properties**, and select **Unblock** on the lower right if available.

GBG pulls in all DLLs in the Drivers folder. It is therefore important for all desired drivers to be unblocked and placed in this folder along with their dependencies. This also provides an easy way of archiving drivers that aren't currently needed. Simply create a subfolder and place any unneeded drivers in it, thus preserving them but making them unavailable to the user. Alternatively, you may delete them. In this way, drivers can be easily added or archived to achieve the desired set. Do not place drivers in the parent Green Button Go folder.

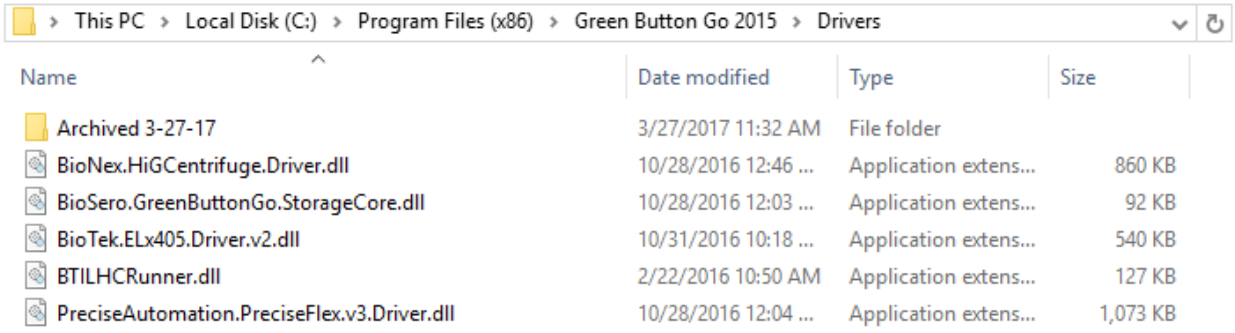


FIGURE 3. UNNEEDED DRIVERS ARCHIVED IN THE DRIVERS FOLDER

Backing Up Drivers

Once a set of drivers has been validated on a system, it is wise to back it up. Simply compress (zip) the Drivers folder and place it in a secure backup location.

Chapter 3. Program Architecture

3.1. Tabs

The six tabs in the GBG interface allow the user to build and run methods. See Chapter 4. Using Tabs for more information.

- **Start Tab:** Create a new program.
- **Layout Tab:** Add instruments.
- **Process Tab:** Define workcell processes and auto-generate underlying procedures, commands, and tables to run these processes.
- **Schedule Tab:** Schedule runs and launch multiple workcell processes.
- **Run Tab:** Start a workcell process and get feedback on state of current runs.
- **Advanced Tab:** Houses the workflow and all its components.
    - Commands are the basic unit
    - Procedures group commands into basic tasks
    - All other objects in the Program Explorer window support procedures and commands or allow user input/user feedback

3.2. Method Architecture

A method will be defined in the Process tab, and then will typically be edited or fine-tuned in the Advanced tab. Below is a brief explanation of the Process logic tree and the procedures that are auto-generated for these processes.

Method Overview

In a general sense, a program has a method which will contain one or more workcell processes. A workcell process will be a complete task such as an acoustic plate replication. The workcell process will have a plate process (a column in the Process tab) for each distinct set of plates. For instance, an acoustic transfer will have at least two: a source plate process and a target plate. Each of these plate processes is populated with drag and drop instrument processes to define the steps of the method. This is all visible in the Process tab. These instrument processes auto-generate procedures and commands in the Advanced tab in order to define the specific actions needed to accomplish their respective steps. When the scheduler sees that an instrument process is ready to accept a plate, it asks that process for the names of the appropriate procedures to run and then runs those procedures. The commands within those procedures are auto-generated but can be edited by the user with the drag and drop palette and canvas in the Advanced tab.

Processes

As you will see below, your method will primarily be laid out in the Process tab. Note that the yellow tile, Layout, represents the Layout tab. The blue tile, Workcell Processes, is defined in the Process tab. The grey area at bottom represents the Advanced tab, including the low-level procedures and commands that actually execute the defined processes.

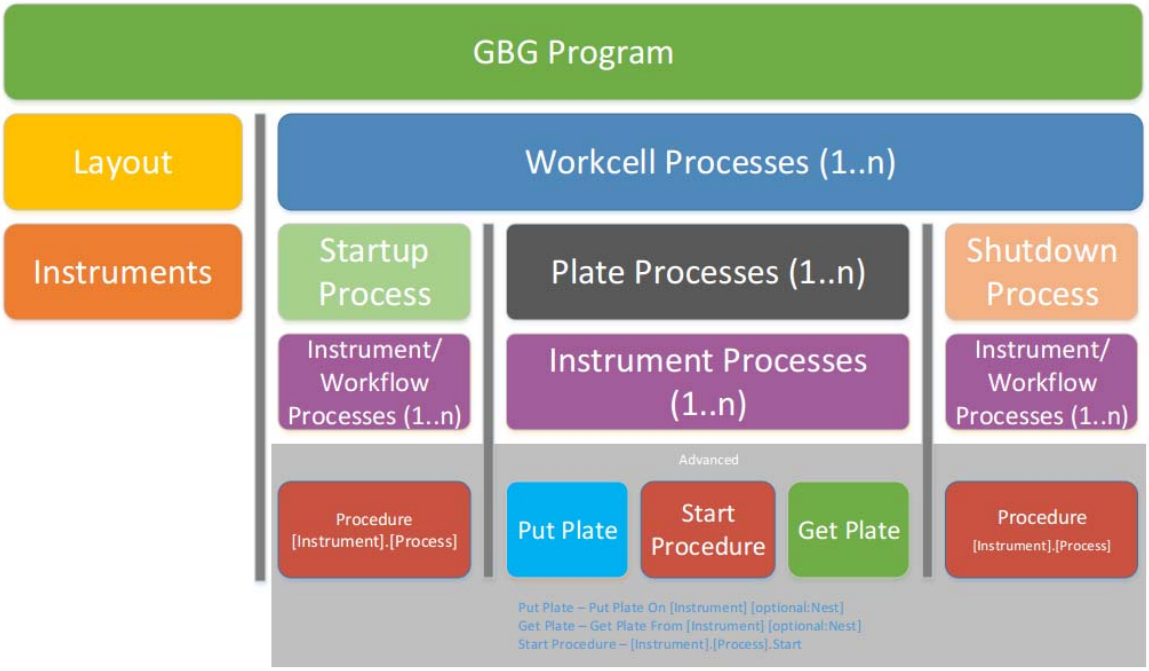


FIGURE 4. ANATOMY OF A GBG PROGRAM

Figure 5 shows how three procedures (in the Advanced tab) are typically auto-generated for each instrument process. The image also shows how multiple instrument commands (the lowest level of logic) may be fired in sequence within a procedure. Again, these are auto-generated but editable.

Process Tree

- Workcell Process (no limit on the number that can be defined in a program)
  - Startup process
    - Special type of plate process that executes at start of run.
    - Used to define initialization, QC (quality control) plate runs, and other one-time tasks.
  - plate processes (no limit on the number that can be defined in a workcell process)
    - This is where most of the “work” is performed in the system.
    - All plate processes, including startup and shutdown, contain instrument processes and workflow processes, which are the lowest level of the tree.
    - These instrument and workflow processes are the tools the user drags and drops to create the method.
  - Shutdown process
    - Special type of plate process that executes at end of run.
    - Used to define shutdown requirements and other one-time tasks.

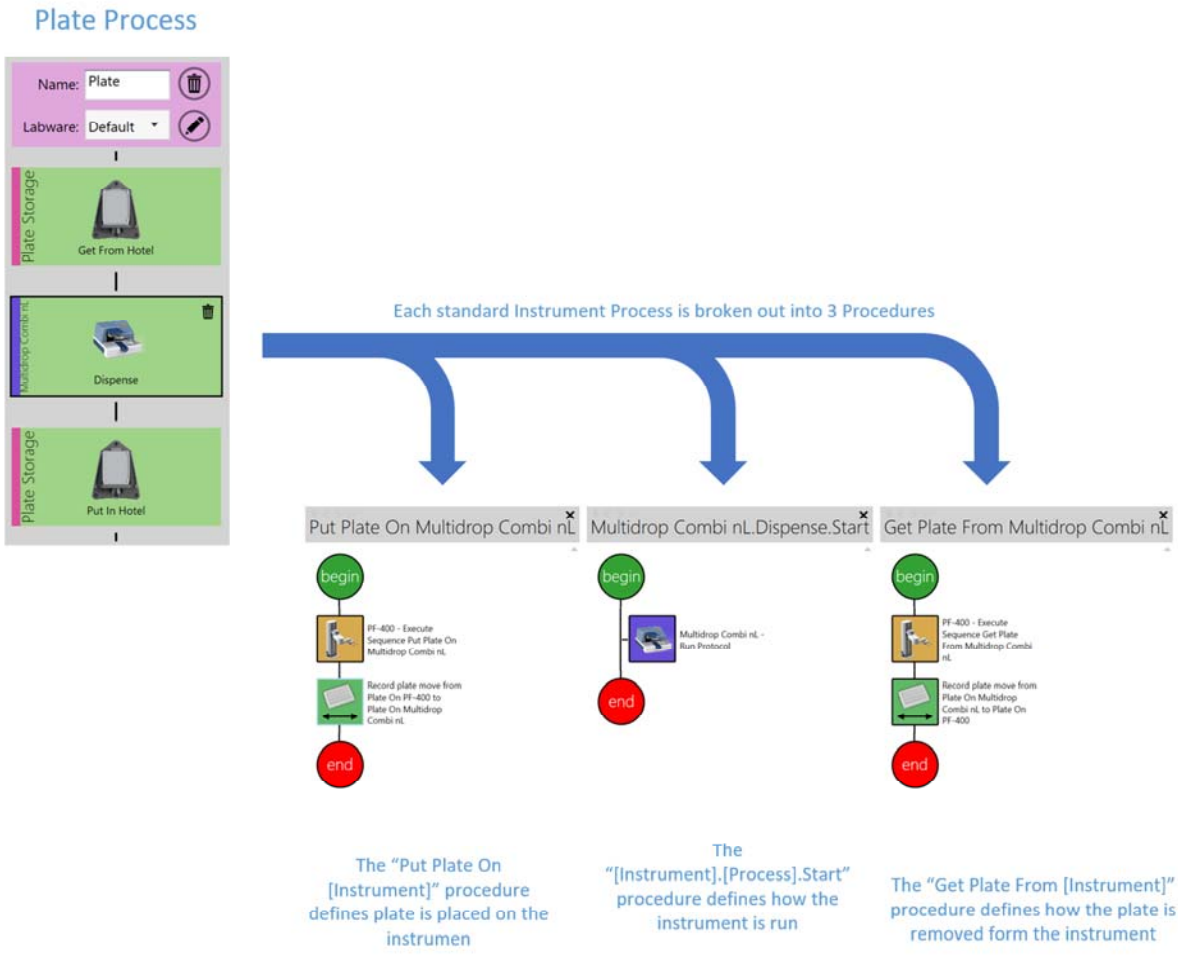


FIGURE 5. PROCESSES AND PROCEDURES

It is common for a simple program to have only a storage configuration process in the startup process and an empty shutdown process. The plate process(es) in this case define everything for the method with the exception of the initial storage data.

Procedures

A procedure encapsulate a given task, such as starting an instrument's dispense. It is populated with commands, which are the lowest level of logic visible to the user. Commands are individual actions taken, such as executing a single robot movement, setting a variable, or branching based on variable states.

Chapter 4. Using Tabs

The main GBG user interface consists of six tabs, accessible from the top of the GBG screen (Figure 6). Method development flows through the tabs from left to right.

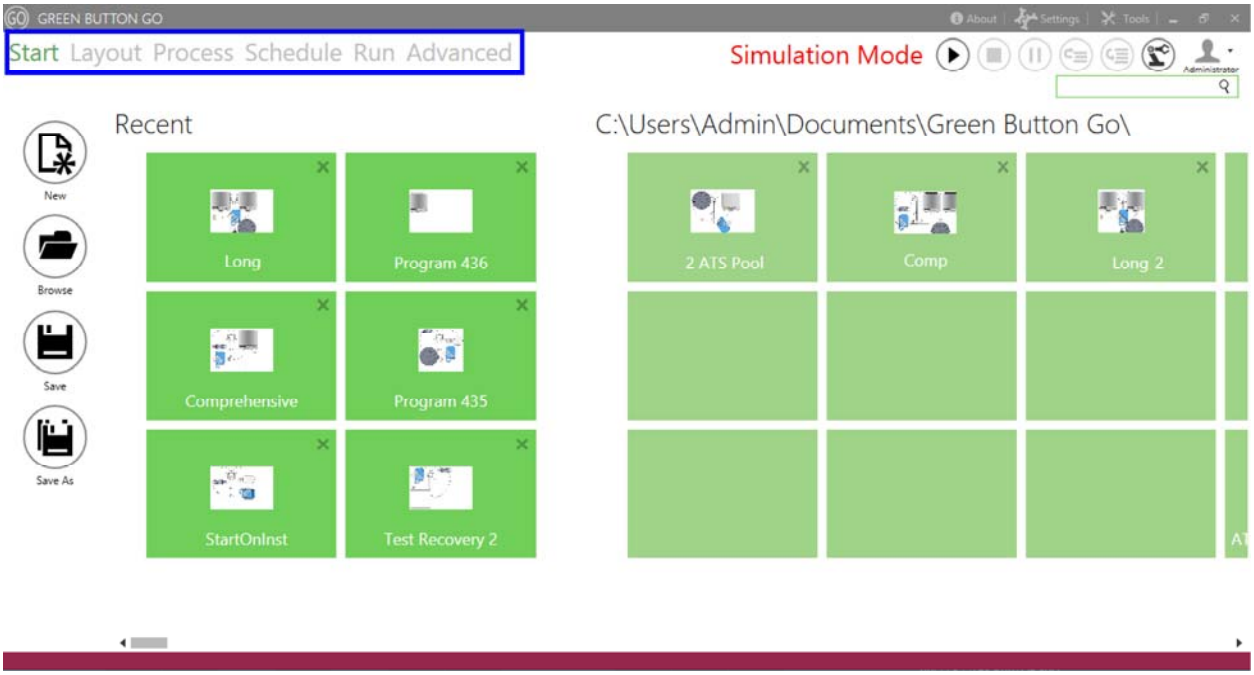


FIGURE 6. GBG TABS

TABLE 2. TAB OVERVIEW

Tab	Function	See page:
Start	Allows new programs to be created or old ones to be opened as well as saved.	15
Layout	Allows instruments to be added and configured. The image generated by the instrument layout will be used throughout the program for visual feedback, but no measurements are used; distances between instruments do not need to be to scale.	16
Process	High level method building interface. Drag and drop tiles allow the method builder to declare the steps that a plate will go through in the run. There are levels of processes, which are defined in Table 1 (Chapter 1). A full run, or workcell process, is the top level, while the bottom level is made of instrument processes, which define individual instrument tasks. Each of these instrument processes then automatically generates lower-level code which can be edited by the user if extra customization is required. This editing is done in the Advanced tab. Also note that a program can contain any number of workcell processes, which can be run in parallel if the necessary instrumentation is available.	19
Schedule	The schedule tab allows workcell processes to be scheduled ahead of time by specifying either a time to run or specifying another workcell process after which the desired run will start.	23

Run	The run tab is the main interface for verification of run state and error recovery. • Integration view: The view on the left of this tab offers a color-coded map of which instruments are running, in error, etc. • Advanced view: The middle view offers a window into the advanced, low-level operations that are customizable in the Advanced tab. By default, the procedures shown here will update as a run progresses. • Gantt and plate timing charts: Select different views on the top left of this interface. Some views only show information on a single plate process, which is chosen on the top right.	24
Advanced	Allows low-level method modifications. The Advanced tab holds some of the most powerful and customizable features. While all other tabs are designed to provide the fastest possible solutions for common needs or operations, the Advanced tab serves the purpose of facilitating any operation. Furthermore, the Advanced tab is capable of defining and running full laboratory methods.	28

4.1. Start Tab

The Start tab allows new programs to be created and old programs to be loaded.

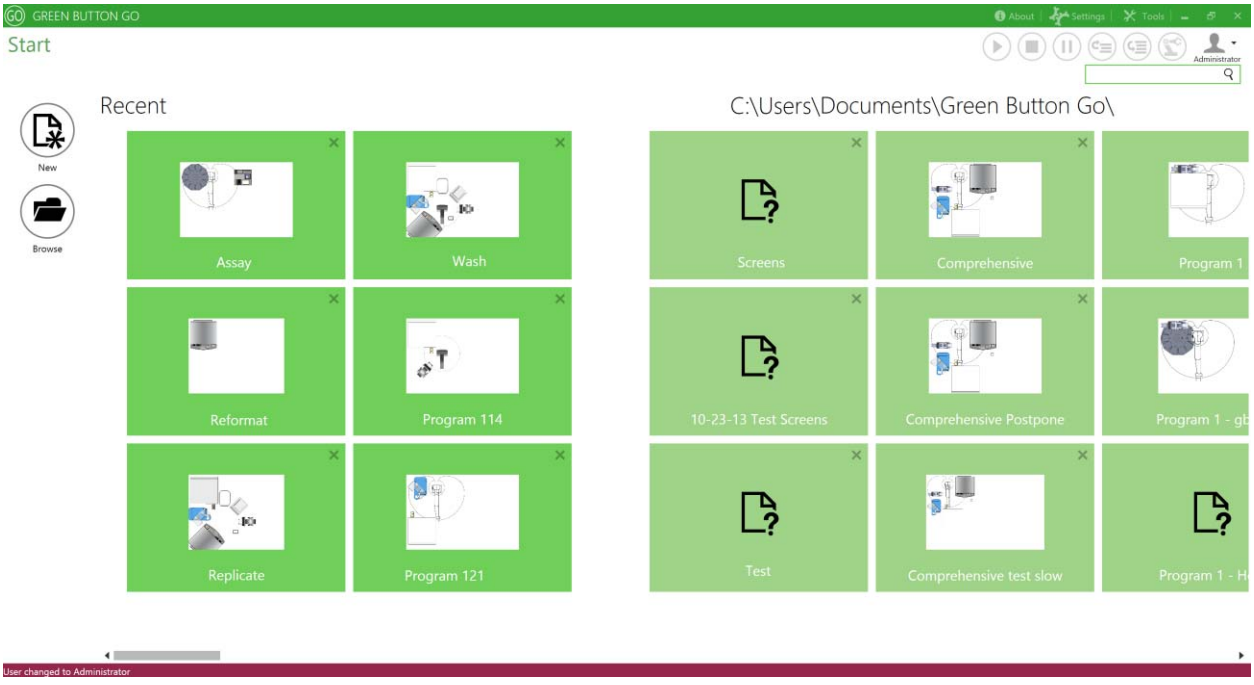


FIGURE 7. START TAB

The Start tab shows previously created programs in two sections, the six most recent programs on the left and all programs in the designated folder on the right. Images on the tiles allow for quick differentiation between different instrument setups.

Once a program is loaded, **Save** and **Save As** will be visible along with the buttons on the left. As with other software, **Save As** is a useful tool for maintaining a "template" program when a lot of similar

programs are needed. However, the ability to create multiple workcell processes in the Process tab mitigates this need somewhat.

4.2. Layout Tab

The Layout tab is used to set up a hardware diagram. It is also used to define instrument properties. Hardware dimensions are to scale, which allows for fast prototyping, but instrument locations do not have to reflect actual positions. This is used for reference throughout the program, but distances between instruments are not used except for visual effect. When the Layout tab is first accessed in a program, a blank canvas is visible with all available instruments in a palette (Figure 8).

FIGURE 8. LAYOUT TAB

The first step in any program is to add the applicable instruments. In the examples in Figure 9 and Figure 10 one of each instrument has been placed, but any number of a given instrument can be added (including 0). To add an instrument, drag it from the palette and drop it in place. Alternatively, click it to add it and then drag it. Once placed, it can be rotated using the selection handle or the controls at the bottom of the properties.

Instrument Properties

Selecting an instrument brings up the properties for that instrument, as shown in Figure 9 and Figure 10. Common properties include connection information such as TCP-IP ports or COM ports. Many instruments have additional configuration information, such as a button to launch the teach pendant (for arms) or to configure storage (for storage devices).

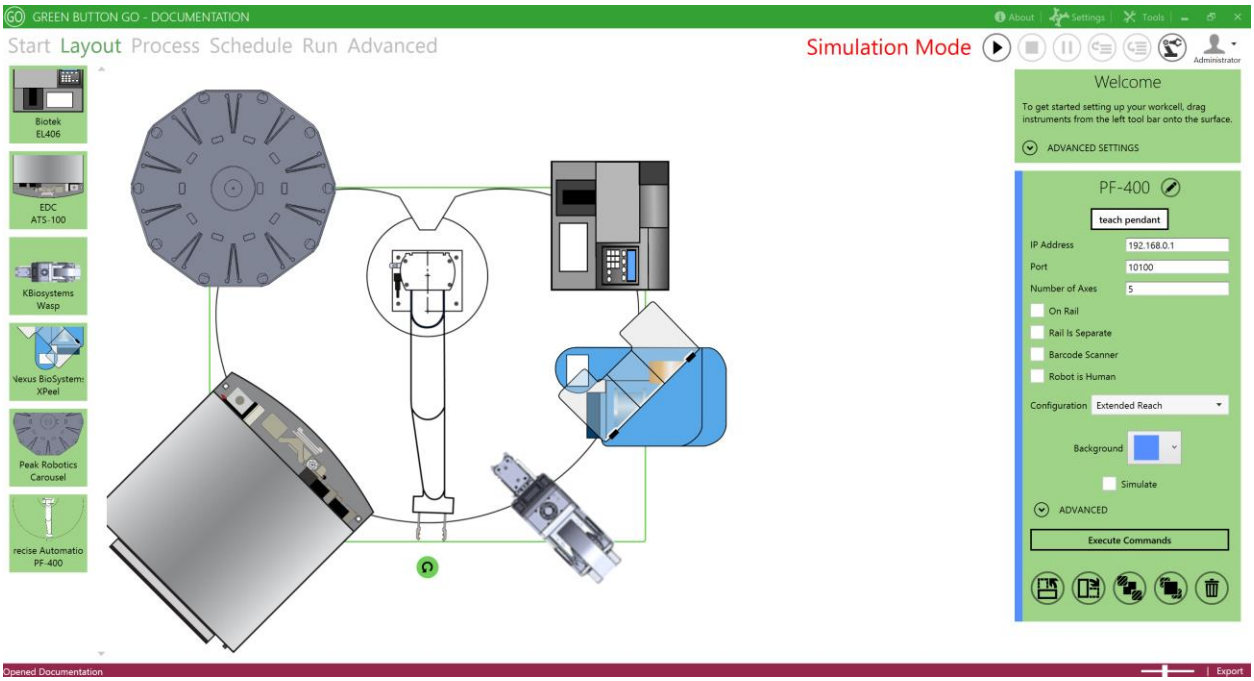


FIGURE 9. LAYOUT TAB: PROPERTIES OF A PRECISE FLEX ARM

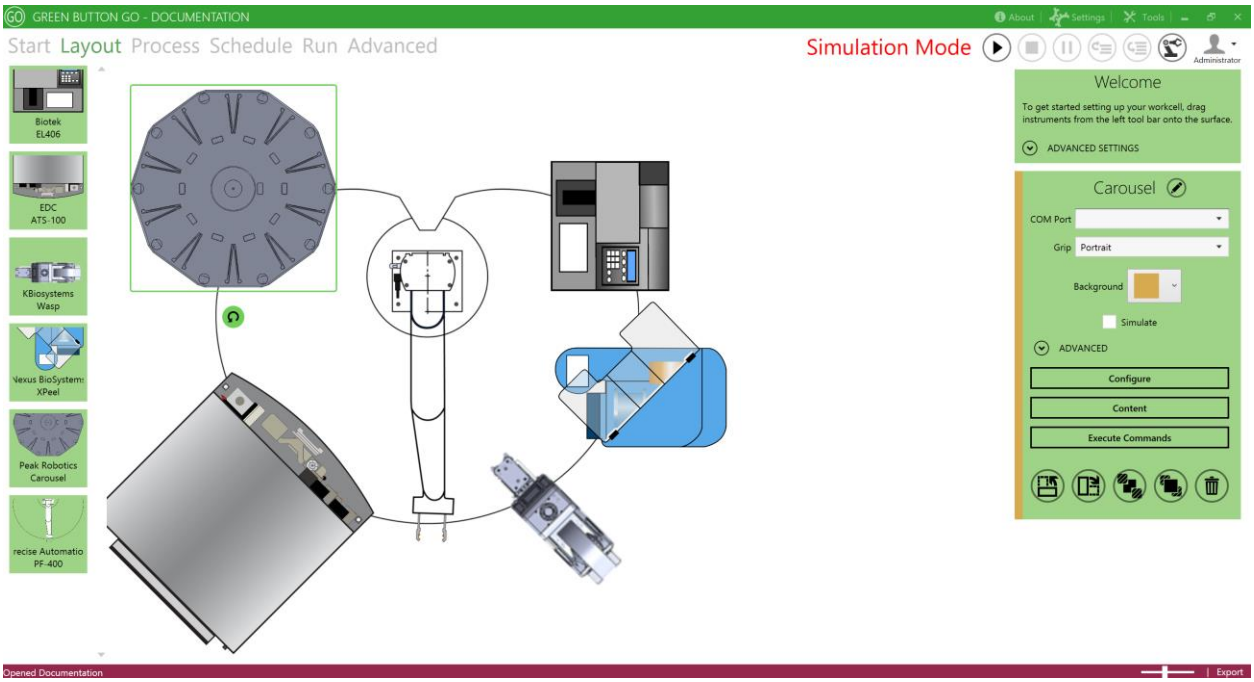
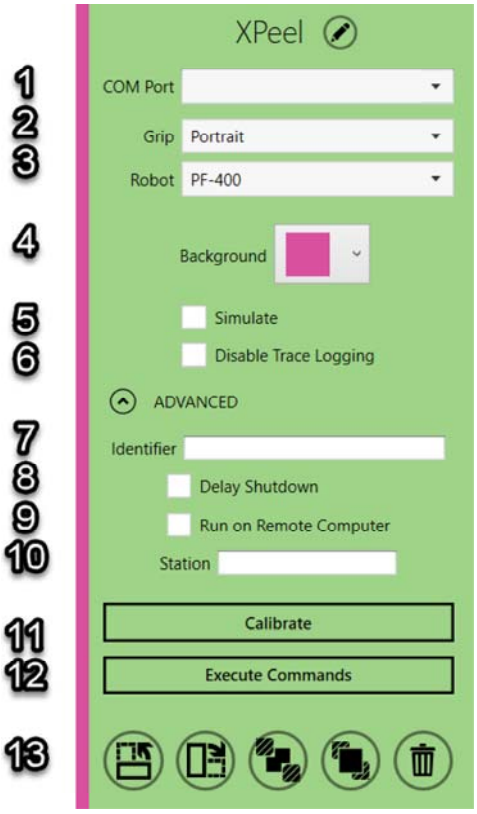


FIGURE 10. LAYOUT TAB: PROPERTIES OF A STORAGE CAROUSEL

TABLE 3. PROPERTIES OF AN XPEEL BY THE NUMBERS



1	**COM Port** to talk to XPeel.
2	**Grip**: Portrait or Landscape grip for the XPeel nest. Arm will automatically regrip plate if necessary and if a regrip nest is provided and configured.
3	**Robot**: Name of robot that accesses the XPeel (should be correct by default in a single-arm system).
4	**Background**: Color associated with this instrument in the GBG interface (for instance, on the strip along the left side of the properties).
5	**Simulate**: check if the instrument should be simulated and the system is not going to be run in full simulation mode, i.e., if other instruments will not be simulated.
6	**Disable Trace Logging**: Disables some of the more detailed logging for this instrument.
7	**Identifier**: Allows a pseudonym, such as a serial number, to be specified for this device.
8	**Delay Shutdown**: Normally an initialize method is called in the driver on program start, and shutdown is called on program end. Implementation details depend on the instrument. Checking this option will delay the shutdown method by a specified number of minutes. If another run is started within that time the subsequent run can occur without re-firing the initialize method. Shutdown will occur at the end of the subsequent run (plus the number of minutes specified). This technique is not necessary except in cases where re-initialization is lengthy.
9	**Run On Remote Computer**: If the GBG client software is installed on a remote computer, and the client software has a copy of this instrument's driver, checking this box allows you to monitor this instrument from the remote computer. For this option, the IP address and the port of the remote computer with the client software must be specified.
10	**Station**: This identifies the instrument's portion of the integration for the Advanced Scheduler or Multi-Path Scheduler.
11	Instrument-specific buttons appear below the Advanced section. This button performs an XPeel calibration action.
12	**Execute Commands** is provided for all instrument. It opens an interface that offers all available instrument commands. This is a way of manually executing individual instrument commands, typically for test purposes.
13	Layout buttons: rotate, move to front/back, or delete the instrument.

4.3. Process Tab

Overview

The Process tab includes some of the most vital time-saving features for laboratories that do method development in-house. Many methods can be fully developed with fewer than a dozen drop commands using this interface. When the Process tab is opened, a single blank plate process will be visible (a single column with the name “Plate” at the top; Figure 11). Please refer to Chapter 3. Program Architecture for an explanation of the role processes play in method architecture.

Instrument processes should be dragged and dropped from the palette on the left onto the desired plate process column in the middle of the Process tab. Figure 11 shows the Process tab with a single Get From Stack process added. Figure 12 shows a complete plate replication method. Note that with 10 drag and drop actions, a user can set up this method, which will automatically share the de-sealer and sealer between source and destination plates.

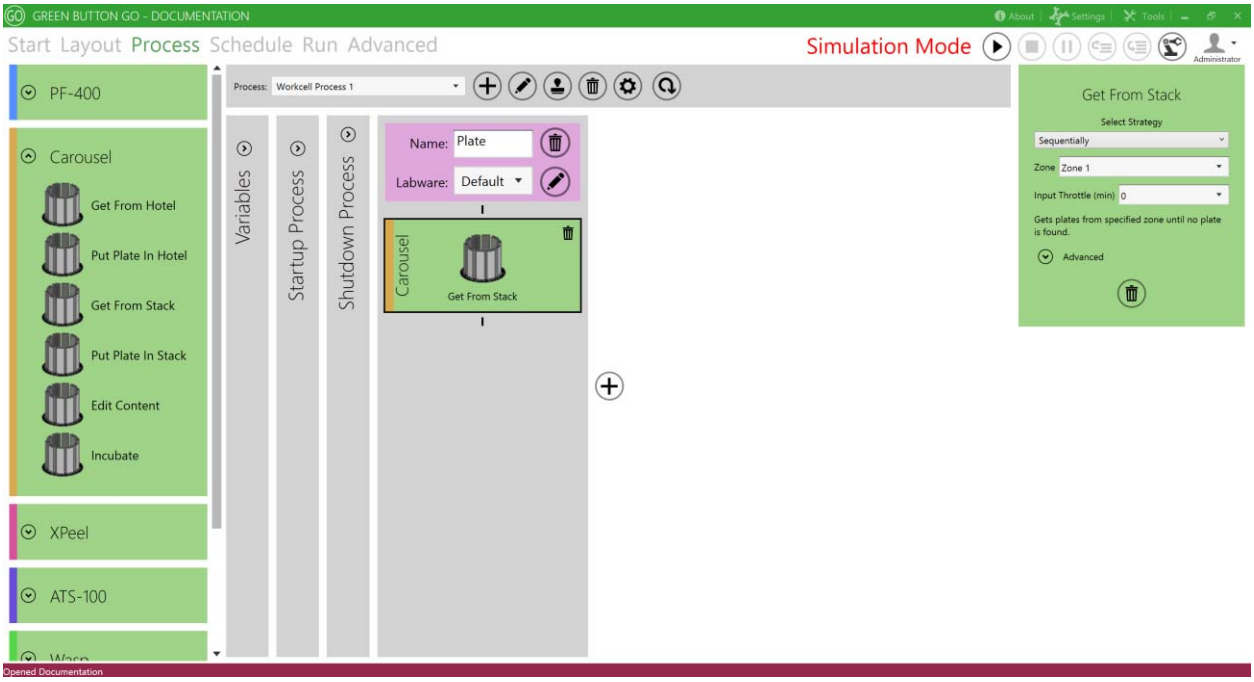


FIGURE 11. PROCESS TAB WITH A SINGLE GET OPERATION ADDED

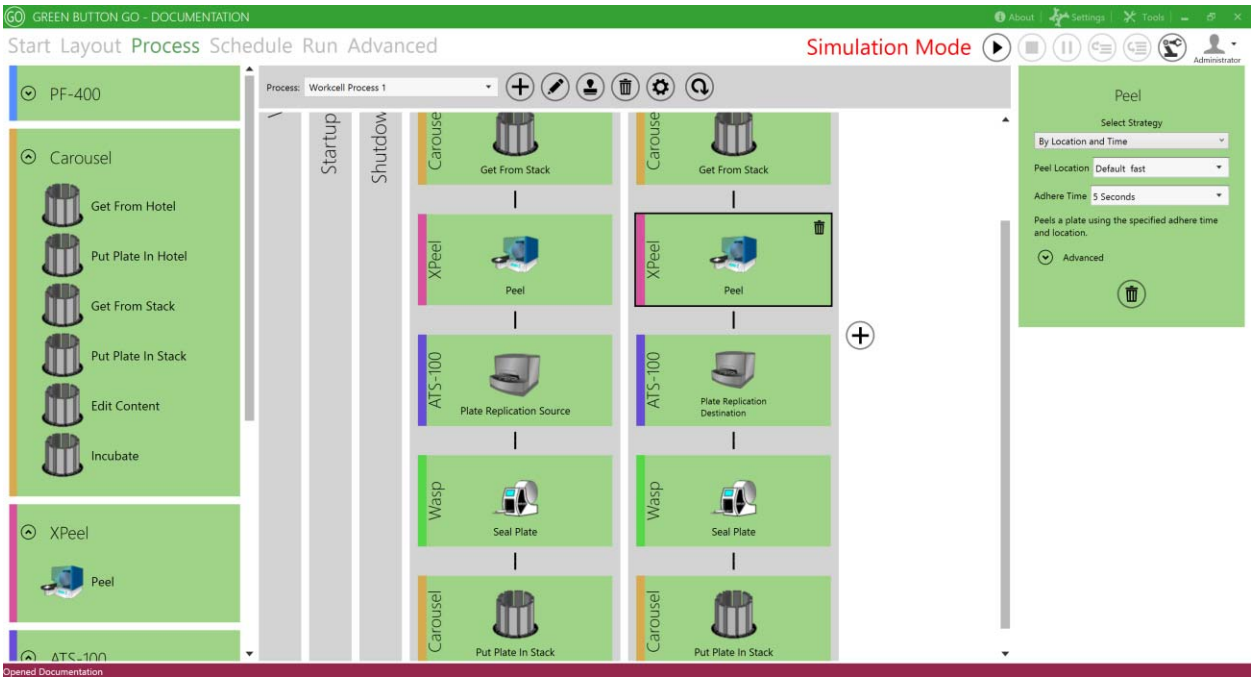


FIGURE 12. PROCESS TAB WITH A PLATE REPLICATION TYPE OF METHOD LAID OUT

Figure 13 shows the startup and shutdown processes expanded. The Startup Process has a storage content process that was automatically added when the Get From Stack process was placed. The Shutdown Process will not be needed for anything during this run. This method is ready to execute using the **Play** button ▶.

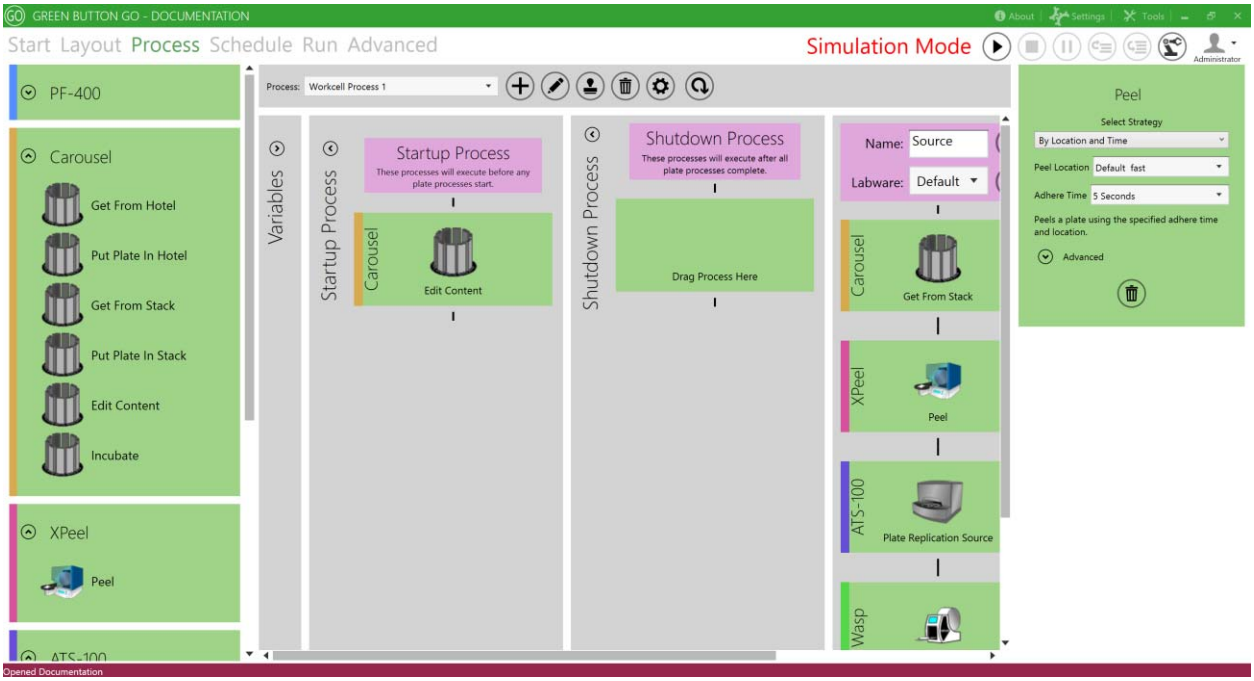


FIGURE 13. PROCESS TAB WITH STARTUP AND SHUTDOWN PROCESSES EXPANDED

Process Tab Details

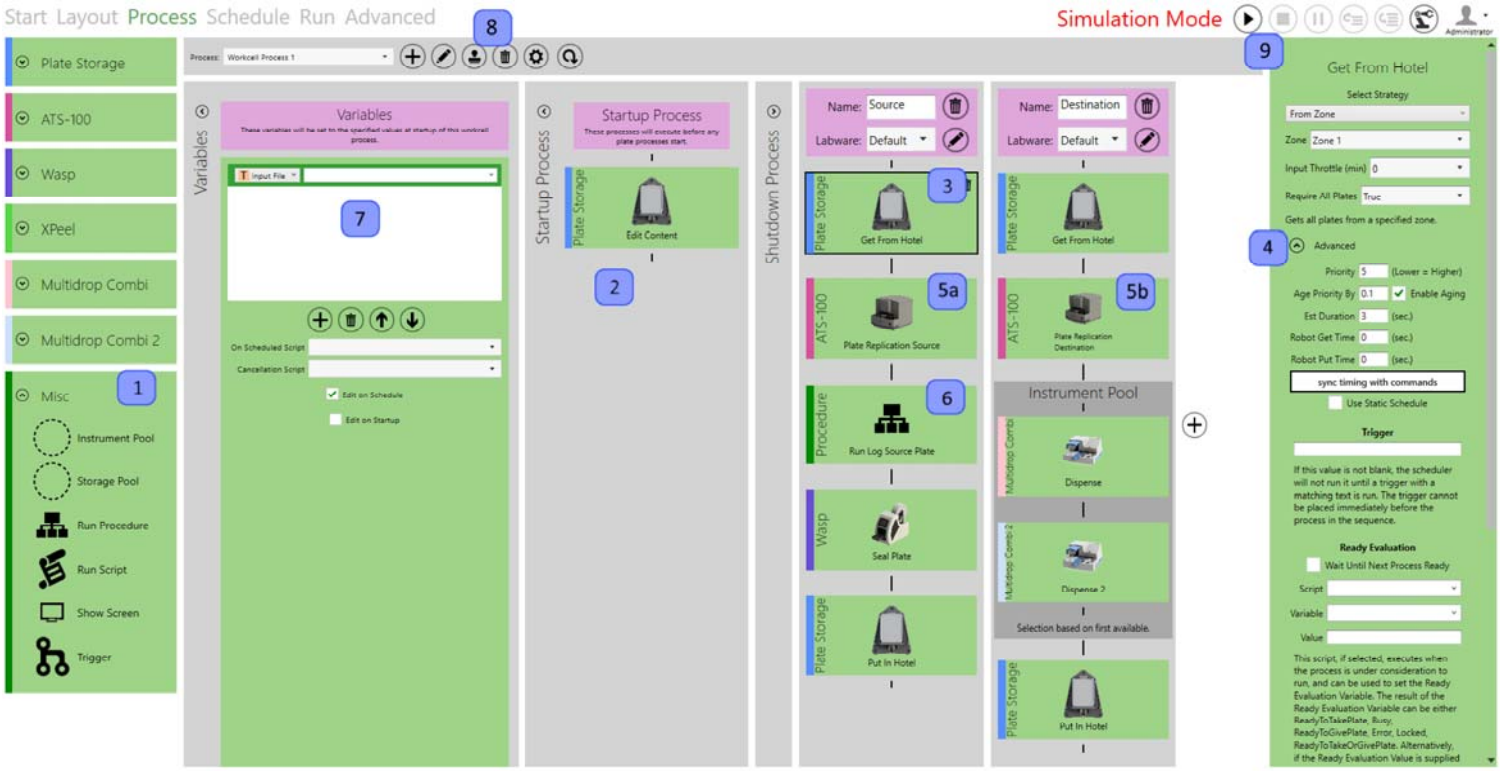


FIGURE 14. PROCESS TAB BY THE NUMBERS

TABLE 4. PROCESS TAB BY THE NUMBERS

1	Processes to add from the palette: Above the number (1) are the instruments and their instrument processes. Below the (1) you see the miscellaneous process options. Number (6) marks a use of the Run Procedure miscellaneous process. The Instrument Pool miscellaneous process is used in the Destination Plate Process.
2	Startup process: Startup and Shutdown processes are optional. A common scenario is depicted here: The Get From Hotel processes automatically added an Edit Content process to the Startup Process. This will prompt the user for input at the start of the run before the Plate Processes execute. The Shutdown Process, similarly, is used for one-time cleanup operations.
3	Instrument process: The primary work of an integration is defined with Instrument Processes. This one provides the correct plates for the Source Plate Process. Its properties can be seen in (4).
4	Process properties: The Get From Hotel process (3) is selected, so these properties belong to that process. Properties above the (4) are unique to this process. Note that the strategy From Zone is selected, meaning that all plates from Zone 1 will be provided as sources for the ATS plate replications. Below the (4) are the expanded Advanced properties. All instrument processes have similar advanced properties available. These are used for operations such as prioritizing processes and addressing deadlocks.
5	Multi-plate process: Clicking one will select both since these are linked. In this case, the ATS always requires a source plate and a destination plate. A similar construct is used for liquid handlers and all instruments with multiple nests (excluding storage). When the process is dropped into the workflow, it will create both the source and destination instances.

6	Run procedure miscellaneous process: Runs the specified procedure after the plate is grabbed from the ATS source and before it is placed on the Wasp. Note that this guarantees the plate is in the gripper when the procedure is called. Similarly, the Show Screen and Run Script processes take place between a get and put procedure. To the right of this process is an Instrument Pool process in the Destination Plate process. The pool means that a destination plate will go to whichever dispenser is available.
7	Startup variables: This column allows you to set specific start values for variables. This can be done in the Advanced tab as well, but in the Process tab the values are specific to each workcell process. This is frequently used with the **Edit On Schedule** check box checked. This means that when a user schedules this workcell process in the Schedule tab, they will be prompted for the specific variables' values. Those values will be saved and applied when the run is scheduled to start.
8	Round workcell process buttons (left to right): a. Create a new workcell process. b. Rename currently selected workcell process. c. Clone workcell process (creates an identical copy with a new name). d. Delete currently selected workcell process. e. Edit workcell Process properties such as deadlock prevention and simultaneous execution. f. Compile: This will auto-generate the procedures that go with the processes you have defined. Compile automatically occurs when the program is run (i.e., the **Play** button in (9) is clicked)
9	Round program control buttons (left to right): a. **Play**: Starts the program. If multiple workcell processes are defined, the user will be prompted to choose one. b. **Stop**: Stops the program, including all running workcell processes. c. **Pause**: Pauses the program, including all running workcell processes. d. **Step Back:** Enabled when a running program is paused; typically used from the Advanced tab. Steps back one instrument command. Note that this is different from an instrument process and is a much lower level of logic. e. **Step Forward:** Like the Step Back, but executes the current instrument's command, thus moving forward one step. f. **Teach Pendant:** Opens the GBG Teach Pendant, which is used for essentially all robotic arms regardless of manufacturer.

Processes and Dynamic Scheduling

By default, GBG runs as a dynamic scheduler, meaning that it optimizes for throughput. The program runs a loop (Figure 15) containing the vital Run Scheduler command. This command looks at the processes defined in the Process tab to see if any defined actions are ready to execute. When individual instrument processes report ready, the Run Scheduler command queues up the necessary procedures, which then run. See 4.6 Advanced Tab for information on procedures. See Appendix B: Instrument Process Status and Ready Evaluations (page 78) for information on process status and ready evaluation.

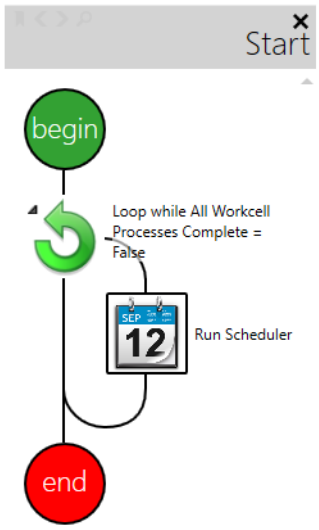


FIGURE 15. MAIN PROGRAM LOOP WITH THE RUN SCHEDULER COMMAND

4.4. Schedule Tab

Overview

The Schedule tab (Figure 16) is used primarily to set up runs ahead of time. It can also be used to view times on past runs, completion status, set up static operating options, and recovery runs that stopped prematurely.

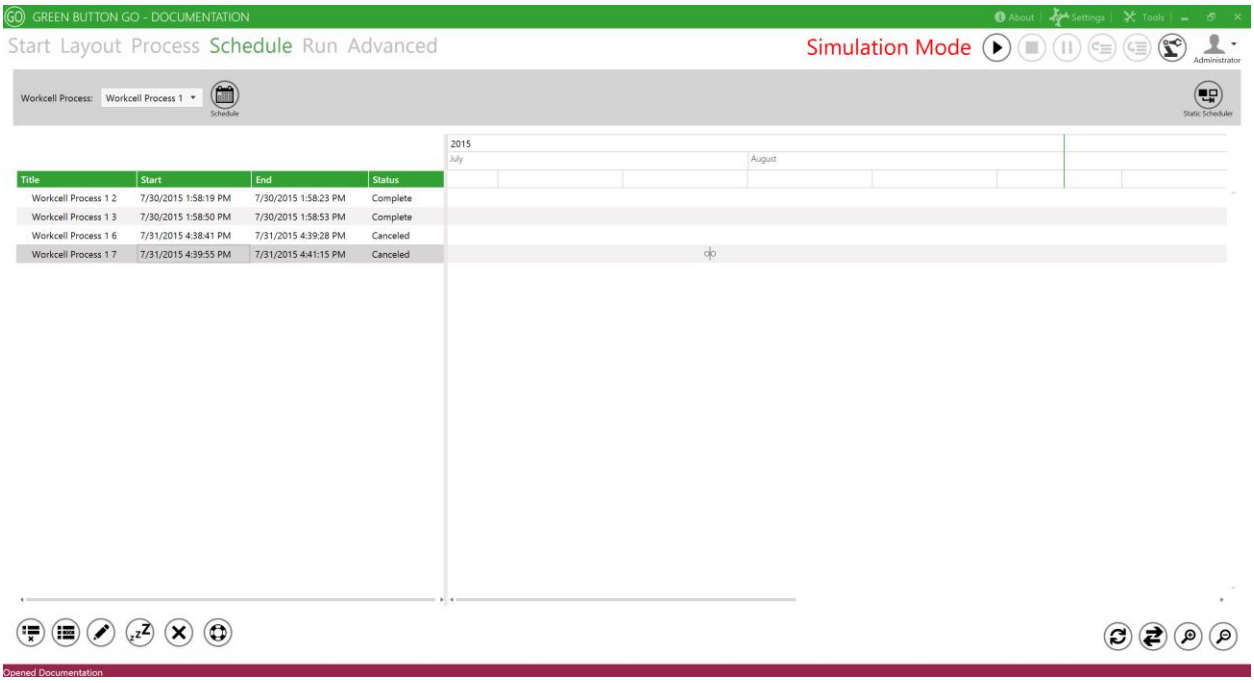


FIGURE 16. SCHEDULE TAB

Common Tools

The Schedule tab has many tools, but the two of the most useful are the schedule button and the recover button. The schedule button (see button highlighted with blue arrow showing a calendar in

Figure 17) is used to set up runs ahead of time. Select the desired workcell process, click this button, and then edit the information on the window that pops up. This window offers two methods of initiating the run:

- Run After: To use this option, select a currently executing run in the drop down. Using this technique means that the desired run will start as soon as the designated run completes.
- Timer: Using the timer allows the user to start runs concurrently, or even to initiate a run immediately regardless of how many other runs are currently executing. See Chapter 8. Scheduling for additional information.

The recovery button (button indicated at the lower left in Figure 17) allows a program to be restarted from the middle. This is used when a program is ended prematurely, either by the user clicking **Stop** or by a computer crash or other unforeseen event. See Chapter 9. Simulating a Run for additional information.

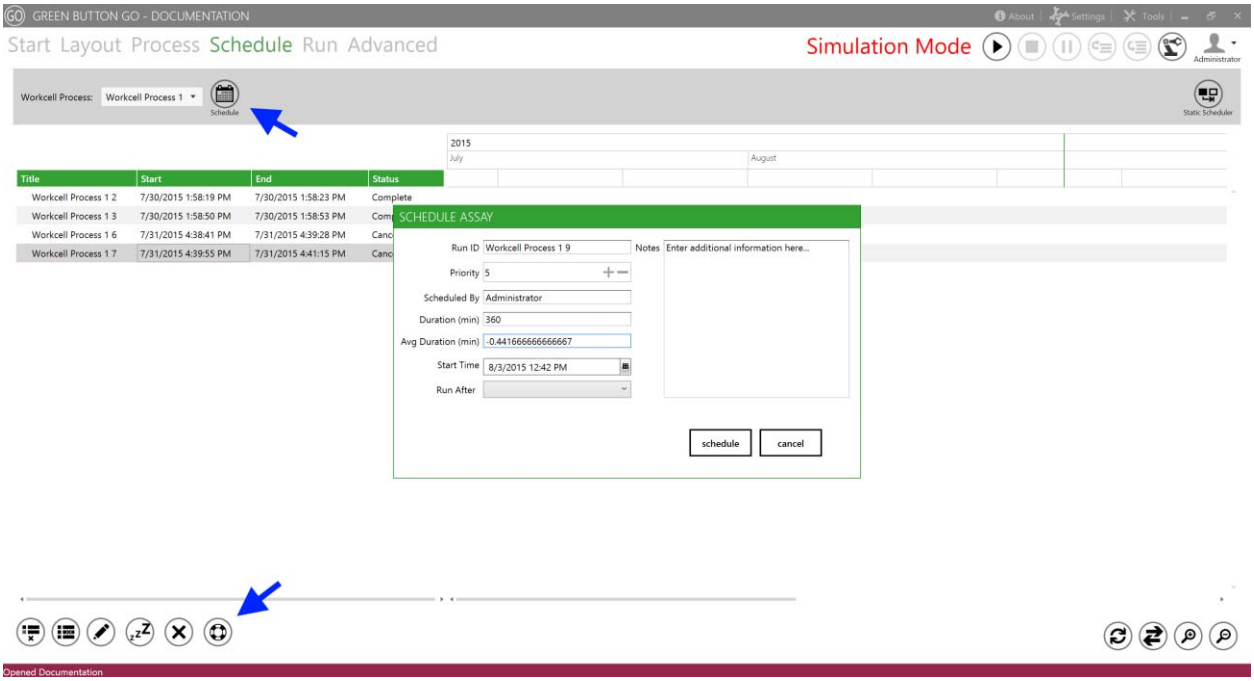


FIGURE 17. SCHEDULE TAB: SCHEDULE AND RECOVER BUTTONS

4.5. Run Tab

Overview

The Run tab (Figure 18) is the default tab from which to run an integration and, by design, will be the tab used most of the time throughout the life cycle of an integration. The **Play** button can be activated from most tabs, but the Run tab has been optimized to provide feedback on the status of the integration. While in the Run tab, the user can see at a glance how a run is progressing and whether or not any instruments are in an error state. The Run tab has three main segments: an instrument overview on the left, the workflow view in the middle, and a Gantt chart on the right.

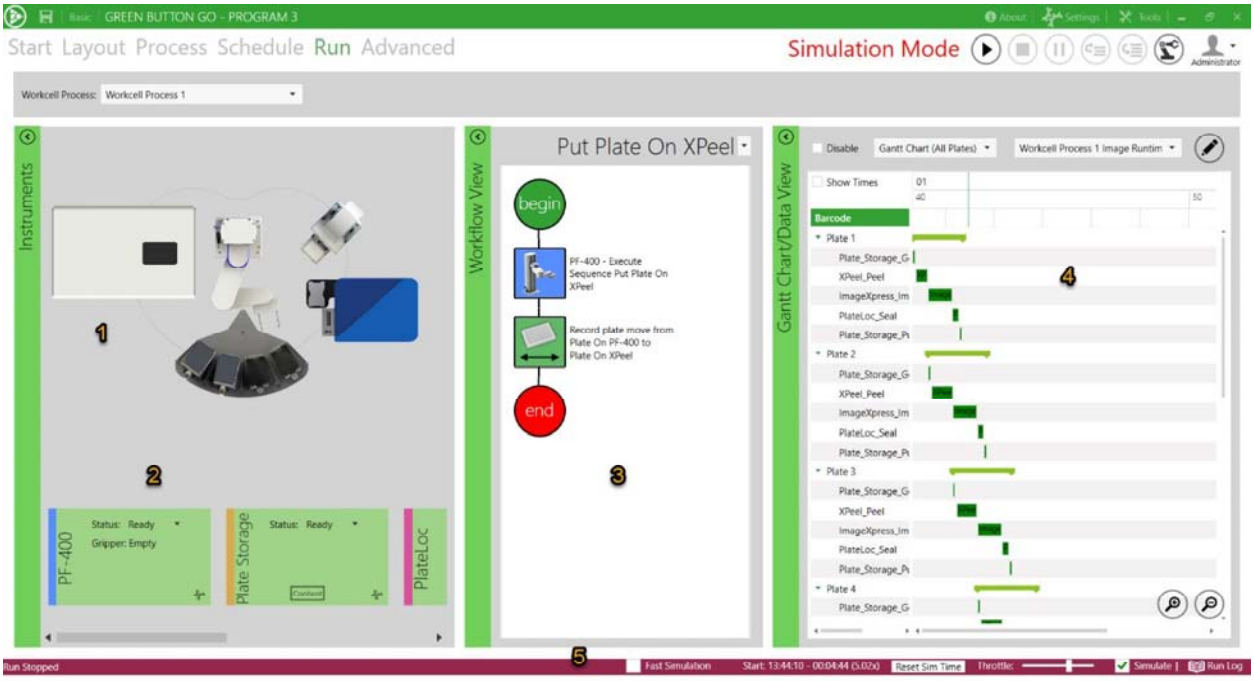


FIGURE 18. RUN TAB BY THE NUMBERS

TABLE 5. RUN TAB BY THE NUMBERS

1	Instrument Overview: The view on the left of this tab offers a color-coded map of which instruments are running, in error, etc. This view also offers a right-click menu on each instrument allowing actions such as selectively simulating an instrument or locking an instrument for offline use.
2	Instrument state tiles will say which barcodes are present on the instrument. They also offer buttons for certain actions such as opening storage contents tables.
3	Advanced view: The middle view offers a window into the advanced, low-level operations that are customizable in the Advanced tab. By default, the procedures shown here will update as a run progresses.
4	Gantt and plate timing charts: Select different views on the top left of this interface. Some views only show information on a single plate process, which is chosen on the top right.
5	Simulation controls: See Simulating a Run for further information on the controls at the bottom of the Run tab.

Tab Sections

Instrument Overview Section

The instrument overview provides exceptional feedback on the current state of all hardware using a color-coded system. Instruments will be overlaid with green when operating, red when in an error state, and yellow when taken offline for stand-alone use. Figure 18, for example, indicates that all hardware is idle, while Figure 19 indicates that the arm is in an error state. Below the instrument top-down view is a series of tiles, one for each instrument, that contain the status of the instrument, the name of the plate on the nest when applicable, and any other controls. In the case of storage instruments, this means a

content button that gives access to the instrument's content data table. Additionally, an ellipsis in the lower right corner opens the advanced window for each instrument. This window allows individual commands to be run on the instrument.

Workflow View

The workflow view is a window into the Advanced tab. It shows the currently executing procedure (unless this behavior is turned off in Settings). For advanced users, this provides a quick reference to the exact low-level location in the program that is current executing.

Gantt Chart

The Gantt chart/data view allows data tables to be visualized either in Gantt chart format, plate timing format, or regular grid format. While all data tables can be seen and edited in the Advanced tab, this tool is for seeing and manipulating the current run data. Specifically, each plate is logged as it enters and leaves processes. In this sense, the Gantt chart section is very plate-centric and gives a good account of what each plate has experienced in the integration. This also makes it easy to see which plates have completed, which barcodes are on the deck, and which still need to run. A plate can also be canceled out of the run by right-clicking on the plate in the Gantt chart and choosing **Cancel** (see page 70).

Error Handling

Since the Run tab is configured for optimal user feedback, error handling will often be accomplished in this tab. When an error occurs, the default action is for GBG to pause and show an error handling window. See Chapter 13. Basic Error Handling and Chapter 14. Advanced Error Handling for more details on this window. As seen in Figure 19, the window is one of three visual cues that an error has occurred. The instrument overview will additionally show which instrument is in an error state, while the workflow view will show exactly which command threw the error.

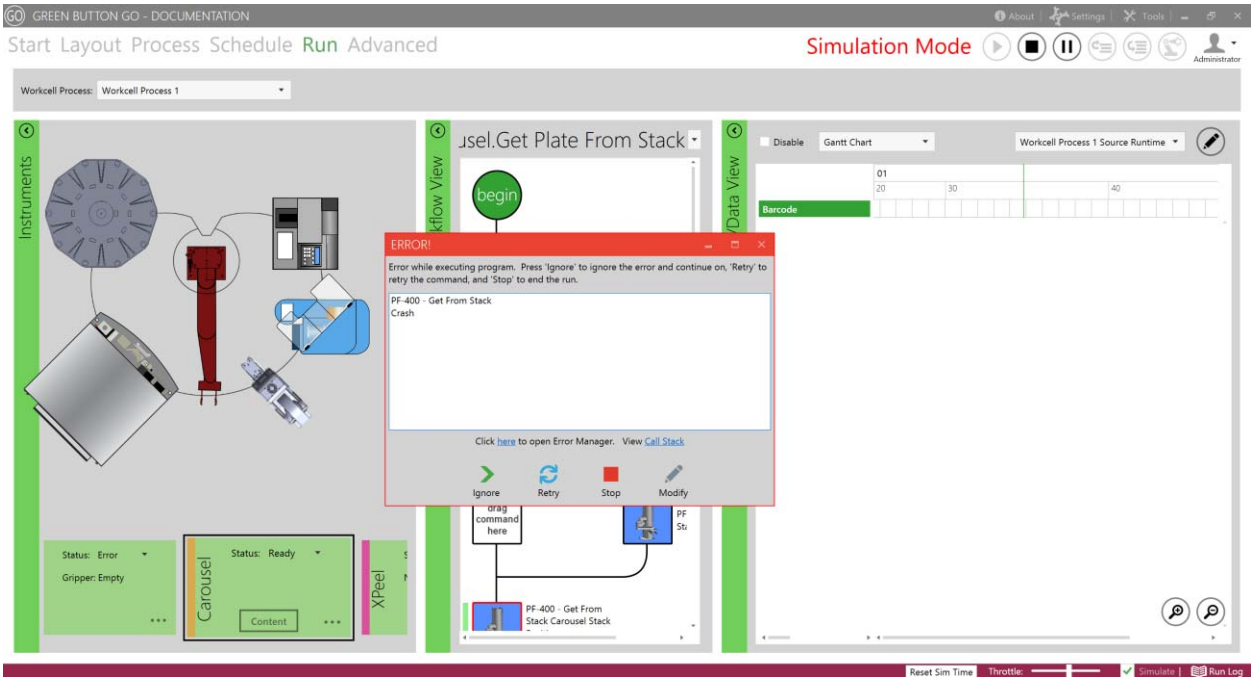


FIGURE 19. RUN TAB WITH ERROR WINDOW

A common error response is the **Modify** button. This pauses the integration and allows the user to look through the program and make edits if desired. It also allows the user to access individual instruments, such as the arm, and fire specific commands. This allows the user to respond in virtually any way necessary to recover. When this is complete, click the **Play** button and the run will continue, starting with a retry on the command that erred.

Note in Figure 20 that after **Modify** is selected, the instrument resets its state to ready but the integration remains paused and the faulty command continues to show a red outline. In this image, the user is ready to take action to fix and resume the run. In this case, the error is simulated so the response would have to include going to the Advanced tab and adjusting the command's parameters. This is displayed in Figure 21.

Numerous tools are available in this tab to handle errors apart from method modifications. As described above, some instruments in the instrument overview offer controls such as content editing or teach pendants. Additionally, the primary robot teach pendant can be accessed using the button at top right with an arm symbol. The tool menu (top right) is available in any tab, and one tool accessed here is the Process Status View. This gives a view of all processes and what statuses are being reported, which allows the user to analyze why plates are moving (or not moving) incorrectly.

The primary role of this tab is feedback. When the integration is running smoothly, it offers detailed information on the run's progress. When things are not operating smoothly, it offers error indicators as well as numerous tools to correct the situation. In both cases, the feedback tools are optimized to provide the necessary information at a glance in order to save the user time when evaluating a complex integration.

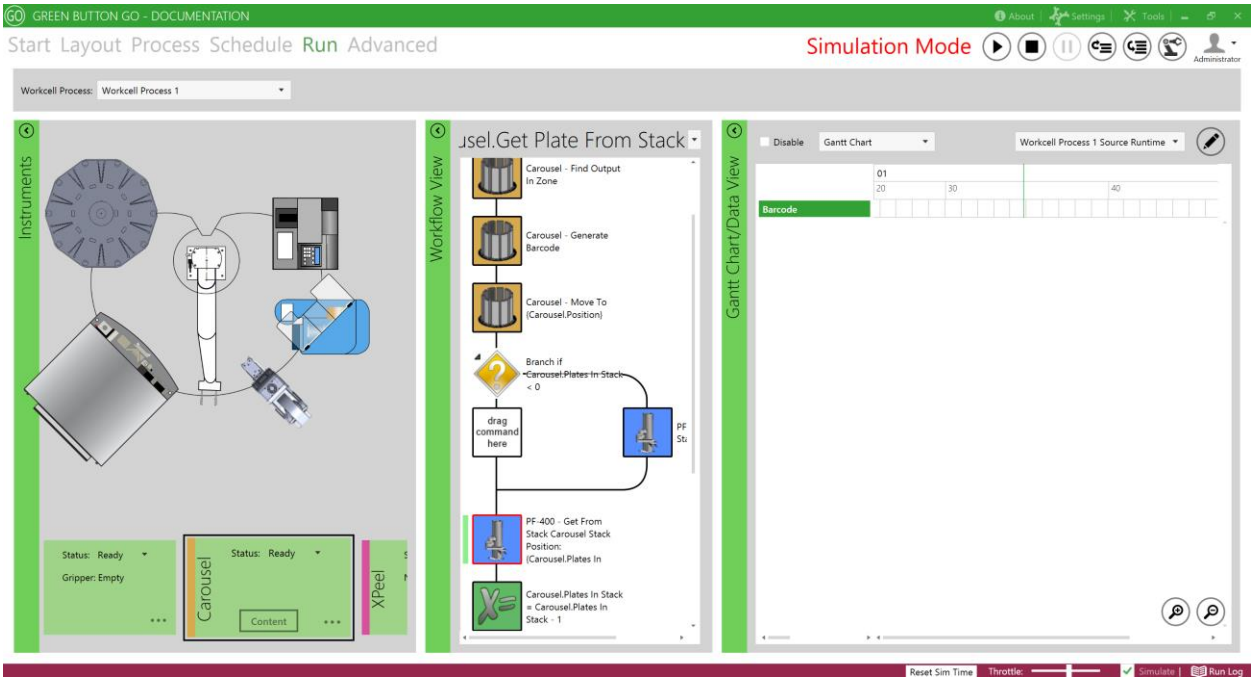


FIGURE 20. RUN TAB AFTER *MODIFY* SELECTED IN ERROR WINDOW

4.6. Advanced Tab

Overview

The Advanced tab is used for a wide variety of tasks including:

- Troubleshooting methods
- Adding optimizations to the method
- Resolving errors

Common Uses

Figure 21 shows the same example program as shown in previous figures. Like the Process tab, this interface offers drag and drop modifications to the method but at the lowest level of the program's structure, instrument commands.

In Figure 21, an error is being analyzed. A movement command was set to simulate a crash (see the Properties pane at the lower right). The run is going, which can be observed based on which buttons are enabled at top right. The currently executing command is the one in an error state: the green bar to its left indicates it is the current command, while the red outline shows it has erred. This tab is an excellent resource for troubleshooting and then continuing the run. You can also make major edits to the method while the run is paused in order to catch and handle edge-case scenarios.

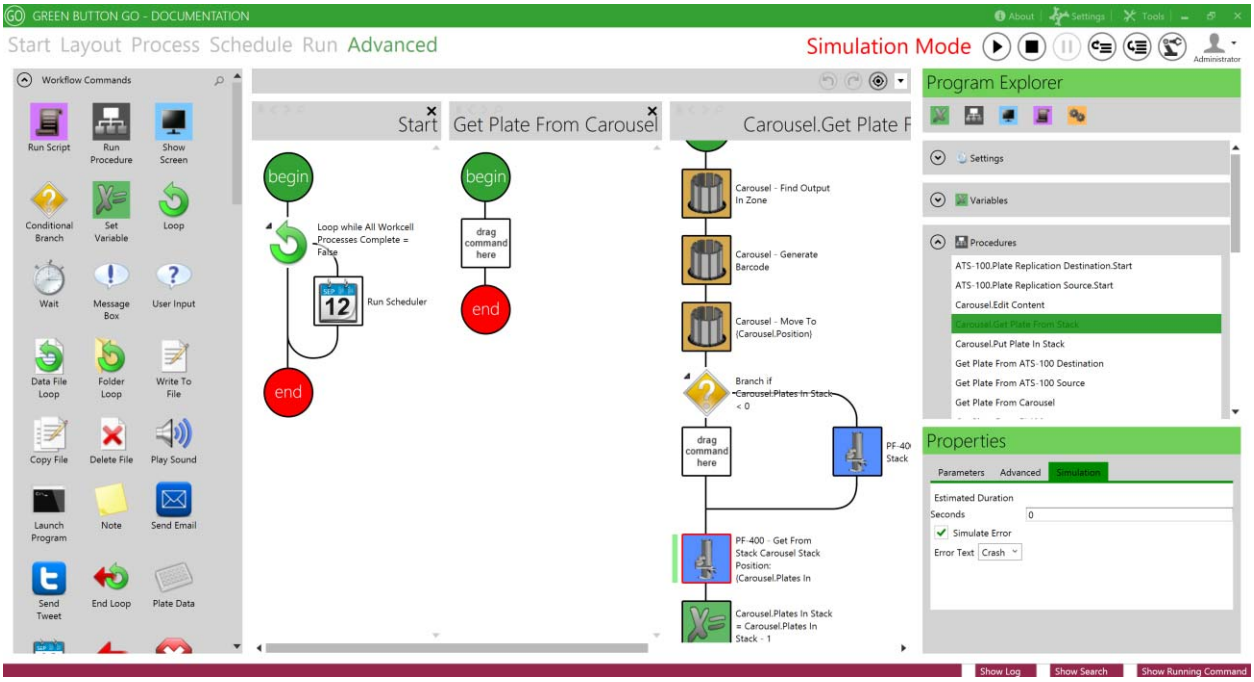


FIGURE 21. ADVANCED TAB

Workflow Commands

Workflow commands are found within the Command Palette (the left panel in Figure 21). They are used to define the tasks and logic of your program. To place them, first open the desired procedure. Start is the default procedure, and all procedures open as a pane in the Workflow View. Once the desired procedure is open, drag and drop commands onto it to define your program's logic. Table 6 lists the common commands which are not instrument-specific and are available for use in every program.

TABLE 6. COMMANDS (NON-INSTRUMENT SPECIFIC)

 Conditional Branch	Define the desired variable to this command.
 Copy File	Create a copy of a file or variable under a new name.
 Data File Loop	Loop once through each line in the selected file.
 Delete File	Delete the specified file.
 End Loop	End the current loop at this point without completing.
 End Program	End the entire program at this point without executing the remaining commands.
 Exit Procedure	Exit the current sub-procedure, but continue to execute the remaining commands of the outer program.
 Folder Loop	Loop through each file in the selected folder.
 Launch Program	Launch any program.
 Loop	Define the number of loops to be performed.
 Message Box	Define a message to appear at this time in the program.
Note	Leave comments throughout the Workflow.
 Play Sound	Play a selected mp3 file.

 Raise Error	Raise an error at this point.
 Run Procedure	Select a procedure to execute at this point in the protocol.
 Run Script	Select a script to run.
 Send Email	Use this command to send an email to the specified recipient.
 Send Tweet	Use this command to post a tweet.
 Set Variable	Set the value of a variable.
 Show Screen	Pop up a customized user interface screen during the run.
 User Input	Trigger a standard form user input screen to manually define a variable of any type.
 Wait	Pause the system at this point for a specified amount of time, or until a specified time.
 Write To File	Create an output file with the values of the specified variables.

Moving/Copying Commands

Workflow commands are used in a drag and drop fashion, allow multi-selection, and can be copied using the right-click (context) menu. A copy/paste can also be performed by holding down the Ctrl key and dragging the desired command.

Enabling and Disabling Commands

Enabling and disabling commands can also be performed through the right-click menu.

Program Explorer Window

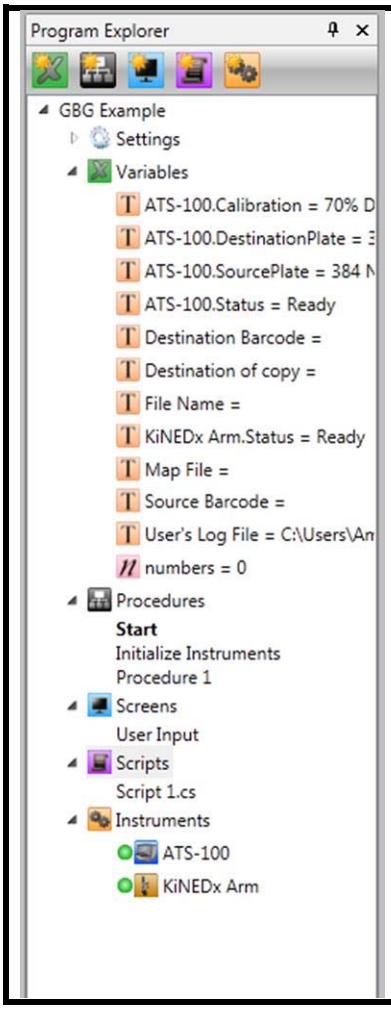
The Program Explorer in the Advanced tab is where all program objects can be found, including:

- Variables
- Procedures
- Screens
- Scripts
- Instruments
- Data tables

Many of these are auto-generated when processes are compiled in the Process tab (See Appendix A: Auto-Generated Components, page 77).

The Program Explorer window allows the user to create all the underlying program objects such as variables, as well as to access the most common settings. All program objects except for data tables can be created using the buttons at the top of the Program Explorer window or by right-clicking on their respective headings.

TABLE 7. PROGRAM EXPLORER

		Create New Variable Add a new variable to the program. Give the variable a meaningful name and define it as a text, number, or true/false variable. Right-clicking on the Variables heading also gives the option Common Variables. This brings up a list of common variable name/type combinations from which to choose.
		Create New Procedure Add a new procedure to the program. This helps to organize large workflows. Note that when using the full scheduling components, the Run Scheduler command will queue up by name the next set of procedures to run.
		Create New Screen Custom design a new user input screen.
		Create New Script Write a new script. The script can be in C#, Visual Basic, JavaScript, or Python.
		Add New Instrument Select the desired instrument from the list of instrument drivers installed on the hard drive. Note that this is normally done in the Layout tab.

Once program elements have been created or added, they appear in the Program Explorer tree. Selecting any element in the Program Explorer tree will open the properties for that element in the Properties window located below the Program Explorer window. Right-click any element in the Program Explorer tree to view available options.

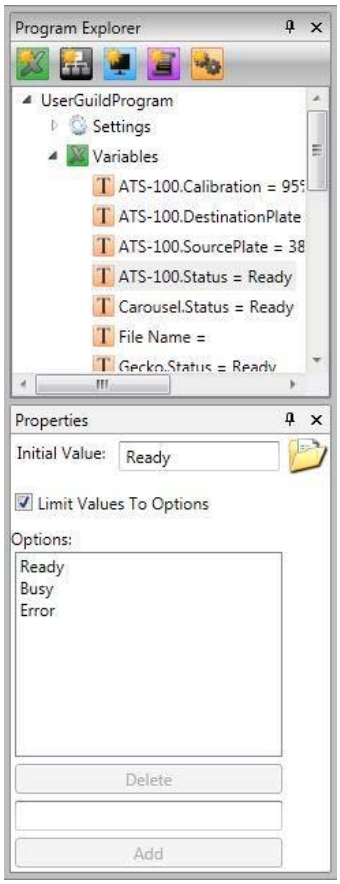
Variables

Use variables to propagate information throughout a program. Add variables as described in Table 7. Examples:

- Count the number of times a loop has been executed.
- True/False variables to confirm actions.
- Ready variables to know if an instrument is available.
- Save file names, text, transfer values, plate identification information, etc. as variables.

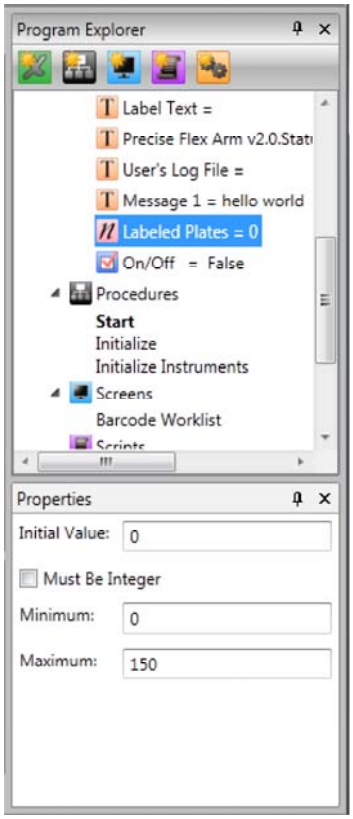
There are three basic types of variables: text, number, and true/false.

TABLE 8. TEXT VARIABLES



- Text variables can hold any character string as its value.
- Set the initial value:
  - Manually, by typing it into the **Initial Value** field, or
  - **Browse** for a text file using the folder button.
- Limit the possible values of the variable to a finite list of possibilities by checking the **Limit Values To Options** box.
  - Add options by typing them into the **Options:** text box.
  - Use the **Add** and **Delete** buttons to add or delete possible options.
  - Ensure that the initial value is listed in the **Options** field or the protocol will display an error when a run is attempted.

TABLE 9. NUMBER VARIABLES

	• Number variables can carry a numeric value within a set range. This type of variable can be incremented or decremented through different command options. • Set the initial value. • Restrict the number to an integer value by checking the **Must Be Integer** box. • Set the minimum and maximum possible values.

True/False Variables

True/False variables will only hold the values of True or False and are used to drive decision making in the program. As such, they are often used with conditional branch commands.

Special Variable Classifications: Instrument Variables

Upon adding an instrument to the program, variables tied to that instrument will also be added to the current variable list. Every instrument is tied to at least its Status variable; for example, My Instrument.Status. The Status variable is a text variable with its options set to Ready, Busy, and Error. The program will set these variables throughout a run to track the status of its instruments.

Setting Variables

Variables have a current value, which is set throughout a run, and an initial value. At the start of each run, the current value is set based on the initial value. Select a variable and use the Properties window to set an initial value. Double-click on a variable to manually set its current value.

Throughout a run there are a number of ways to set variables' current values to new values:

- The Set Variable workflow command sets a variable to a predefined value.
- The User Input workflow command allows the user to choose a value at runtime.

- Use a Show Screen workflow command. Screens are described in 4.3 Process Tab and allow users to quickly and easily set multiple variables.
- Double-click on a variable to manually set it. At runtime, this requires the program to be paused and in Developer Mode.
- Use a script to set a variable.
- Any workflow command that has an output variable (seen in its properties) will set the value of the selected variable at runtime. Many instrument commands have output variables.

Common Variables

Common variables are variables stored in GBG that are frequently used in protocols. These variables have predefined values in the Common Variables *.csv files in the install folder. The values can be edited by navigating to the files and changing or adding values.

To add a common variable to a program:

1. Right-click on **Variables** in the Program Explorer tree.
2. Select **Add Common Variable**.
3. Select the desired variable to add from the popup window and click **Okay**. The variable type will be specified by the icon next to the variable name.

Screens

User input screens are user-definable windows that appear throughout the run, allowing the user to receive and input data quickly and easily. For example, screens can be used to track the status of instruments, watch or change database values, and set variables. Screens should contain one or more screen controls, which are described below. Controls give feedback to the user and/or take input from the user. Most enable the user to easily set variable values.

By default, the program waits for a screen to close before continuing (the same behavior as a User Input or Message Box command). Clicking any button control will close the screen. This behavior can be changed by checking the **Runtime Screen** checkbox in a screen's Properties. Checking this property significantly changes the function of the screen in that the screen stays open throughout the run. Feedback and input are much more dynamic on such a screen and care should be taken to avoid setting variables at an undesirable time in the run.

Things to consider when creating a screen:

- What feedback and input are desired? All screen controls provide information to the user, take information from the user, or do both.
- Where is the information coming from? Files, folders, physical documentation, observations, external data? Be sure to use the correct User Input Tool for the type of information.
- What data type is to be input? Use the proper variable type.
- Do all necessary variables already exist?

Customize User Input Screen Appearance

A user input screen can be customized to meet the needs of the users. The size, color, position, duration of visibility, and contents can be specified for each input screen.

To customize a screen:

1. After creating a user input screen, click on the desired screen name in the Program Explorer Tree to bring up its properties.
2. Enter numeric values for the width and height of the screen in the Properties window.
3. Check **Runtime Screen** if the screen is to stay open throughout the run.
    - Leave this box unchecked to cause the screen to be open only until the user inputs the requested information and clicks a button (any button on the screen will close it).
4. Check **Center** to center the input screen in the display screen.
    - Leaving this box unchecked allows the user to define the X- and Y-position of the input screen.
5. Select the color of the input screen using the color block in the Properties window.

Scripting

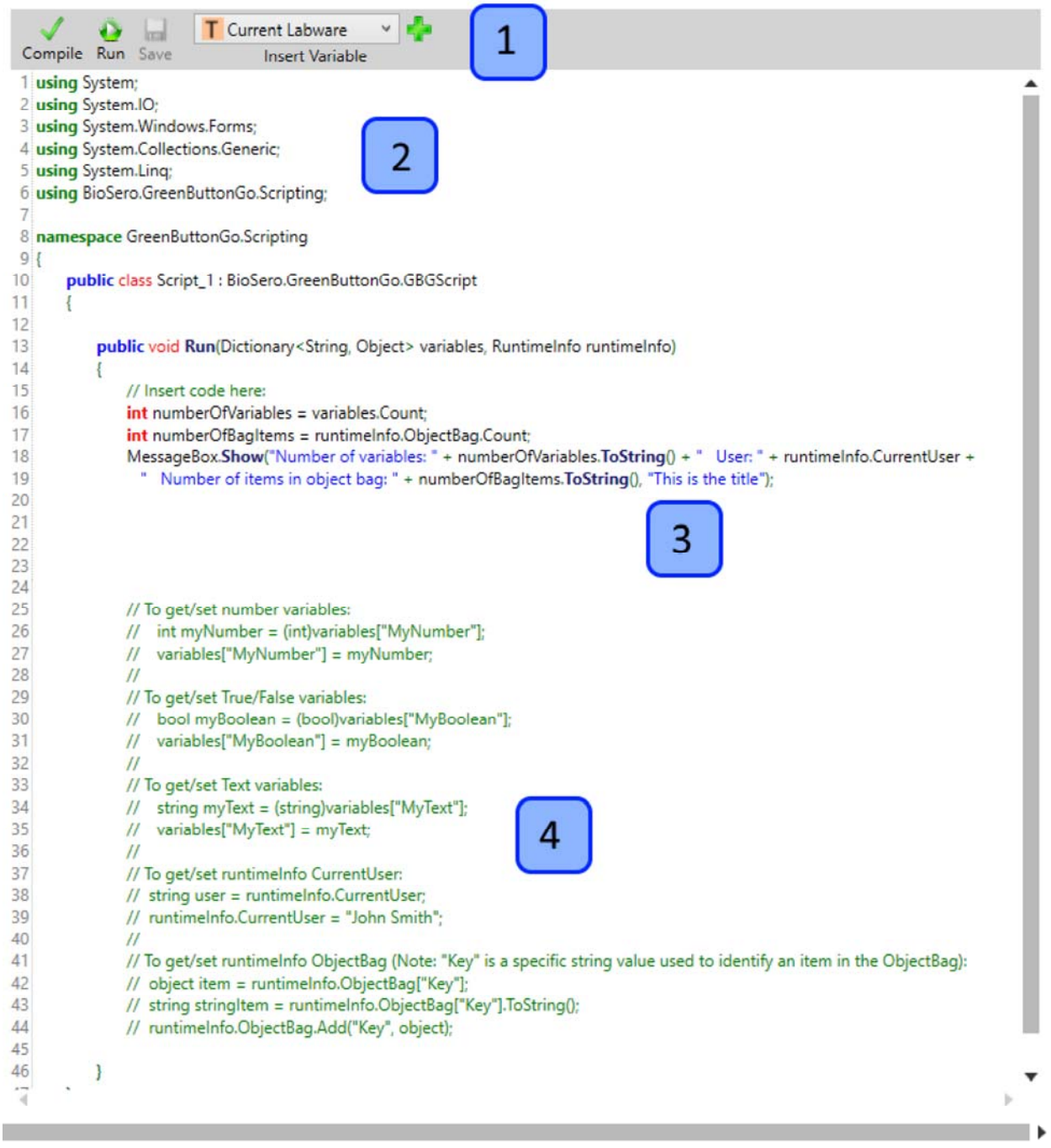
A script can be added in the Program Explorer in the Advanced tab. Current language options include C#, Visual Basic .NET, Javascript, and Python.

See Figure 22 for an example script, described below:

1. Control panel: Use the **Compile** button to check if the script will build correctly. Note that if it compiles but fails at runtime, it is most often due to a misspelled variable name.
2. Using statements: The default using statements are typically adequate. It is also possible to add "Reference" above the using statements and to reference a custom DLL. See Appendix C: Referencing External DLLs in GBG Scripts for more information.
3. Body of the script: This is where the main logic is written. It usually entails setting variables based on values in other variables or in data tables. The code shown here is a default which simply displays a message box showing the number of variables in the program.
4. The green text in the default script is "commented out", meaning it is instructional only. Use this as a guide to get and set the values of variables within the program.

Reference

For help writing scripts, see Appendix C: Referencing External DLLs in GBG Scripts.




```
using System;
using System.IO;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Linq;
using BioSero.GreenButtonGo.Scripting;

namespace GreenButtonGo.Scripting
{
    public class Script_1 : BioSero.GreenButtonGo.GBGScript
    {

        public void Run(Dictionary<String, Object> variables, RuntimeInfo runtimeInfo)
        {
            // Insert code here:
            int numberOfVariables = variables.Count;
            int numberOfBagItems = runtimeInfo.ObjectBag.Count;
            MessageBox.Show("Number of variables: " + numberOfVariables.ToString() + "   User: " + runtimeInfo.CurrentUser +
              "   Number of items in object bag: " + numberOfBagItems.ToString(), "This is the title");



            // To get/set number variables:
            //   int myNumber = (int)variables["MyNumber"];
            //   variables["MyNumber"] = myNumber;
            //
            // To get/set True/False variables:
            //   bool myBoolean = (bool)variables["MyBoolean"];
            //   variables["MyBoolean"] = myBoolean;
            //
            // To get/set Text variables:
            //   string myText = (string)variables["MyText"];
            //   variables["MyText"] = myText;
            //
            // To get/set runtimeInfo CurrentUser:
            //  string user = runtimeInfo.CurrentUser;
            //  runtimeInfo.CurrentUser = "John Smith";
            //
            // To get/set runtimeInfo ObjectBag (Note: "Key" is a specific string value used to identify an item in the ObjectBag):
            //  object item = runtimeInfo.ObjectBag["Key"];
            //  string stringItem = runtimeInfo.ObjectBag["Key"].ToString();
            //  runtimeInfo.ObjectBag.Add("Key", object);

        }
```

FIGURE 22. DEFAULT C# SCRIPT BY THE NUMBERS

Common Uses

GBG is designed to use scripts sparingly by providing drag and drop or settings options for most scenarios. However, scripts are often still used for customized control such as:

- Ready and skip evaluations that are applied in the Process tab
- Populating storage from LIMS
- Debugging workflows
- Triggering subsequent runs
- Advanced/automated error handling
- Registering plates with hits based on reader threshold

References

References can be added to the top of a script and must include the full directory of the DLL being referenced. All reference lines must be placed at the top of the script (before using statements or empty lines) and must start with “REFERENCE”. For example, one might put “REFERENCE c:\Full Path\File Name.dll” to reference a file called “File Name.dll”. Troubleshooting tip: If this doesn’t seem to work, verify that there are no permissions issues by running GBG as an administrator and testing with the file in different locations.

Accessible Objects

Almost any object in the program can be accessed from a script, typically by using the ProgramManager object or the RuntimeDataManager object. The ProgramManager can access virtually anything in the Process or Procedure trees, as well as the execution engine. The RuntimeDataManager can access and alter essentially anything in the data tables. Between these, a great deal of optimization is possible within GBG.

Chapter 5. Run Scheduler Command

As described in Chapter 3. Program Architecture, processes are laid out in the Process tab, and their actual implementation is performed when their associated procedures are executed during a run. The Run Scheduler Command is the connecting point.

When GBG is run, the Start procedure is executed. By default, this procedure is populated with a loop and one instance of the Run Scheduler command (after the program is compiled). The Run Scheduler command looks for processes that are ready to execute. When it finds a possible plate movement, it enqueues the associated procedures in GBG's run engine, meaning that those procedures will execute immediately after the Run Scheduler command finishes execution.

5.1. Run Scheduler Command

The Run Scheduler command (Figure 23) is the only command that dynamically adds procedures to the internal workflow engine. In the simplest example, an integration might be pulling plates from storage, sealing them, and putting them away. In this case, the scheduler asks the storage instrument if it has any plates available, queues up the procedure in which the arm moves the plate to the sealer, then calls the procedure in which the seal operation is run. This last "run" procedure usually contains an asynchronous command, meaning that the program continues operation while the operation is being performed. In this case, the scheduler loop runs through another iteration and finds that nothing can be performed, as the sealer is occupied and running. After the sealer finishes, the next time the scheduler command runs, it will queue up commands for picking up the plate and putting it away. The next time the scheduler runs, it will find that the system is ready for another plate.

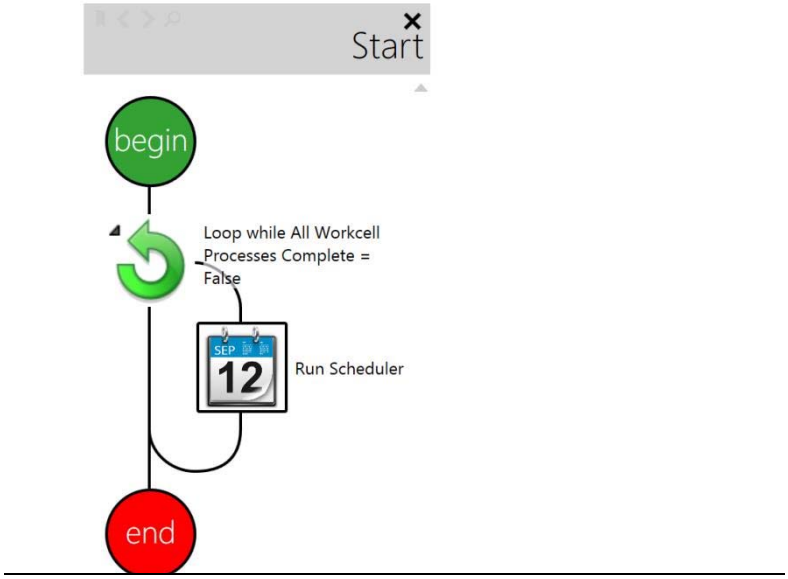


FIGURE 23. RUN SCHEDULER COMMAND IN START PROCEDURE

Note that this command must work with the processes to determine what actions are ready. This includes a great deal of reading from and writing to the data tables, which can all be viewed (and edited) in the Advanced tab.

Tools available to see the inner workings of the Run Scheduler command

1. The commands and procedures which are dynamically added can be viewed in a procedure called Scheduler Trace. This procedure is not visible by default, but can be made visible by clicking **Settings** in the top tool bar (Figure 24), then checking the **Show Scheduler Trace** box in the window that appears (Figure 25).
2. You may see the last set of decision-making that was performed by the scheduler by selecting **Tools** in the top tool bar, then selecting **Process Status View** in the drop-down menu. This is invaluable when troubleshooting.
3. The Process/Procedure View (access by selecting **Tools > Process/Procedure View**) looks similar to the Process Status View, but shows which procedures will be called for each process.

FIGURE 24. TOP TOOL BAR

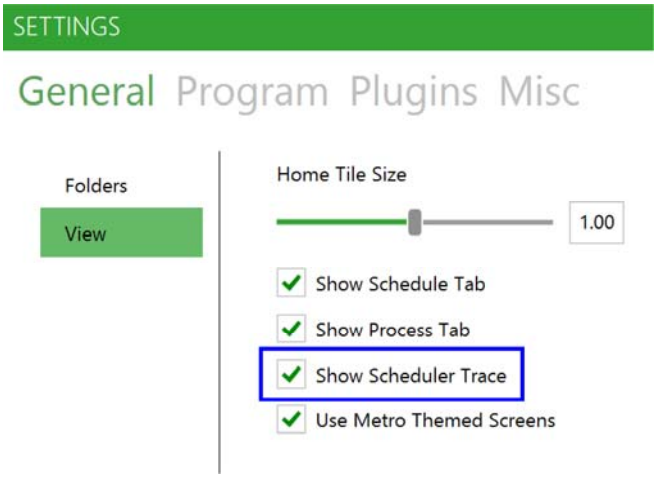


FIGURE 25. SETTINGS WINDOW

5.2. Advanced Details of the Scheduling System

At the heart of the scheduling algorithm, the scheduler command asks each process if it is ready to take or give a plate. There are many advanced calculations that are taken into account, but the general principle is that when a process says it is ready to give a plate, and the following process says it is ready to take one, the scheduler cues up all necessary components in order to move the plate and to run it on the appropriate instrument.

There are a few additional considerations that the user has some control over. These include the deadlock analysis and the Skip/Ready evaluations, all of which are configurable in the Process tab. The following list is a very general example of what gets cued up to run a single plate through a single process:

- Get Plate Procedure
    i. Set barcode in arm variable
    ii. Remove barcode from previous instrument variable
- Set end time in data table
- Place Plate Procedure
    i. Set barcode in new instrument variable
    ii. Remove barcode from arm variable
- Set start time on new process in data table

This list excludes any additional complications, such as getting a plate from storage, putting one back in storage, or using a Plate Mover class of arm, which typically incorporates storage. Sometimes a low-level modification of a procedure is the easiest way to introduce a desired adaptation. It is then vital to understand the order in which procedures will fire. To see the correlation between processes and procedures in your method, consult the Process/Procedure View.

Note

It is possible to run GBG without processes or the Run Scheduler command, in which case ALL method logic is defined in the Advanced tab. This should only be done by an advanced method designer, and only in special circumstances

Chapter 6. Storage

6.1. Storage Fundamentals and Run Time

The fundamental principle of storage in GBG is that the software needs to know where your plates are and how you want them accessed. For a more thorough explanation, see Appendix D: Biosero Universal Storage Guide, page 83.

The location of your plates is defined in data tables which can be edited manually, automatically by a device with a barcode scanner, or with a script pulling information from a file or LIMS system. How to access the plates is defined in the Process tab by dragging on, for instance, a Get From Hotel process and setting its properties. This is of course dependent on hardware and desired workflow.

At run time, the system typically asks the user either how many plates will be run or for the file path to a worklist defining the plate locations. Alternatively, the data tables can be manually edited at the start of a run. Examples of input include:

- manual input
- worklist input
- automatic shelf scanning in an incubator
- automatic shelf scanning with an arm-based barcode scanner.

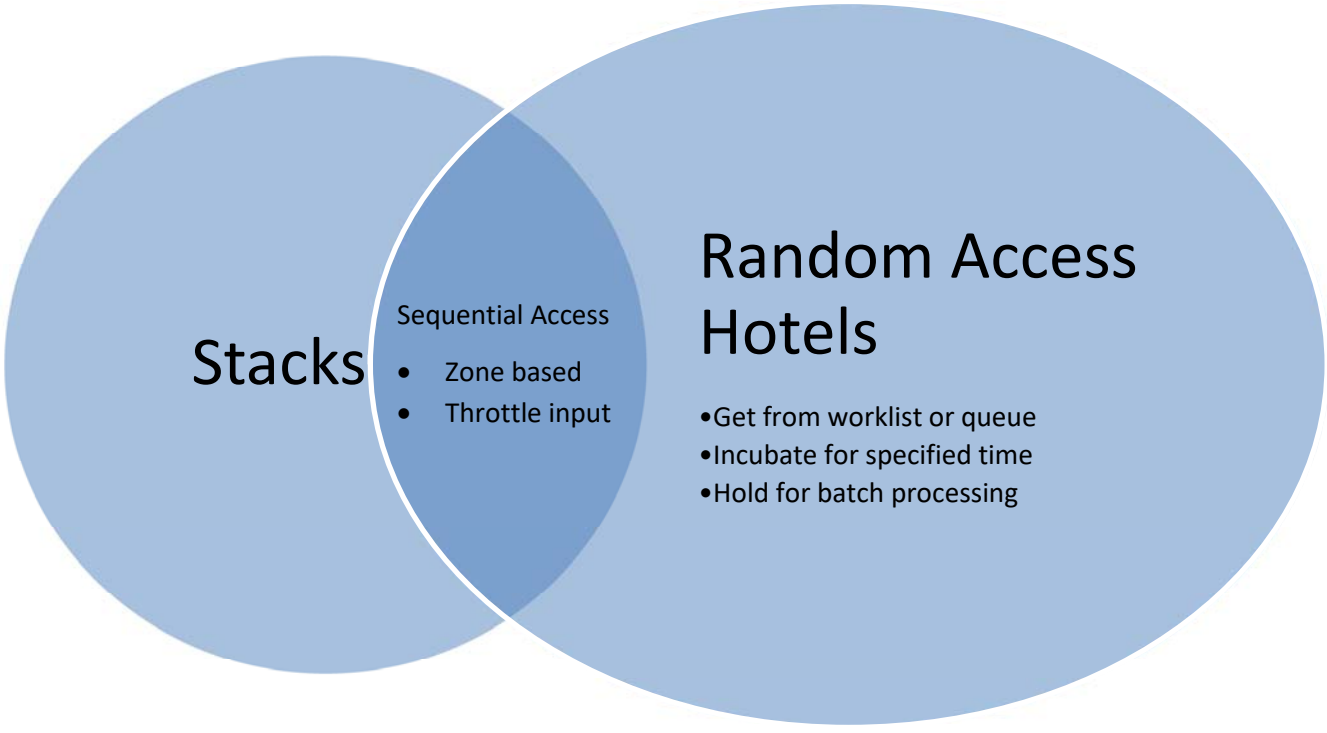


FIGURE 26. STORAGE DIAGRAM: OPTIONS FOR GETTING PLATES

As seen in Figure 26, either stacks or hotels can be used to get plates in a sequential fashion. Barcode-based storage logic can only be used with Random Access Hotels, Carousels, or Incubators.

6.2. Sequential

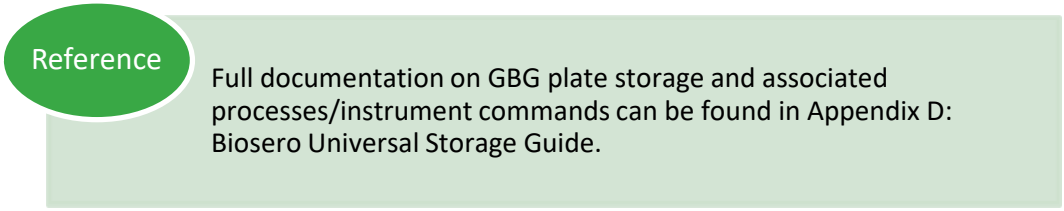
Getting plates sequentially is simple and can simplify user input. Sometimes the only input is the number of plates to get. Other properties include the zone name and a throttle number in case it is desirable to input plates into the system with a delay between each plate.

6.3. Barcode-Based

Getting plates based on their barcodes gives a lot more flexibility than a sequential get. The details of each option can be found in Appendix D: Biosero Universal Storage Guide, page 83.

The most fundamentally important concept is that plates are tracked by barcodes. The barcode can be a pseudonym (dummy barcode) that follows the plate through the whole system, a real barcode, or a pseudonym that gets changed to a real barcode when the plate reaches a barcode scan step. If real barcodes are used these can be scanned in by hand, read from a file, or (hardware permitting) automatically scanned at the start of a run. In any event, the storage needs to know what to call each plate, and will use the value in the Hotels data table under the Barcode column. This also signals the system that a plate is present in that spot.

Getting plates based on their barcodes typically means getting from a worklist that is either pre-generated or requested by another instrument like an ATS. Pre-made worklists sometimes include timing data such as the time a plate should come onto the deck for a read operation. Random access also allows for incubation and holds for batch processing.



Chapter 7. Teaching the Robot

GBG employs a Teach Pendant built by Biosero and implemented on most robot arm drivers. This gives you an equivalent experience between different arms used within GBG. The Teach Pendant can be accessed from the Process tab by clicking the arm icon in the top menu bar.

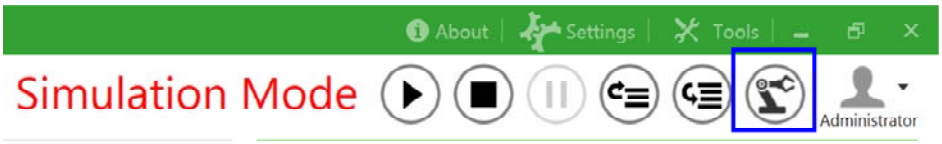


FIGURE 27. ACCESSING THE TEACH PENDANT

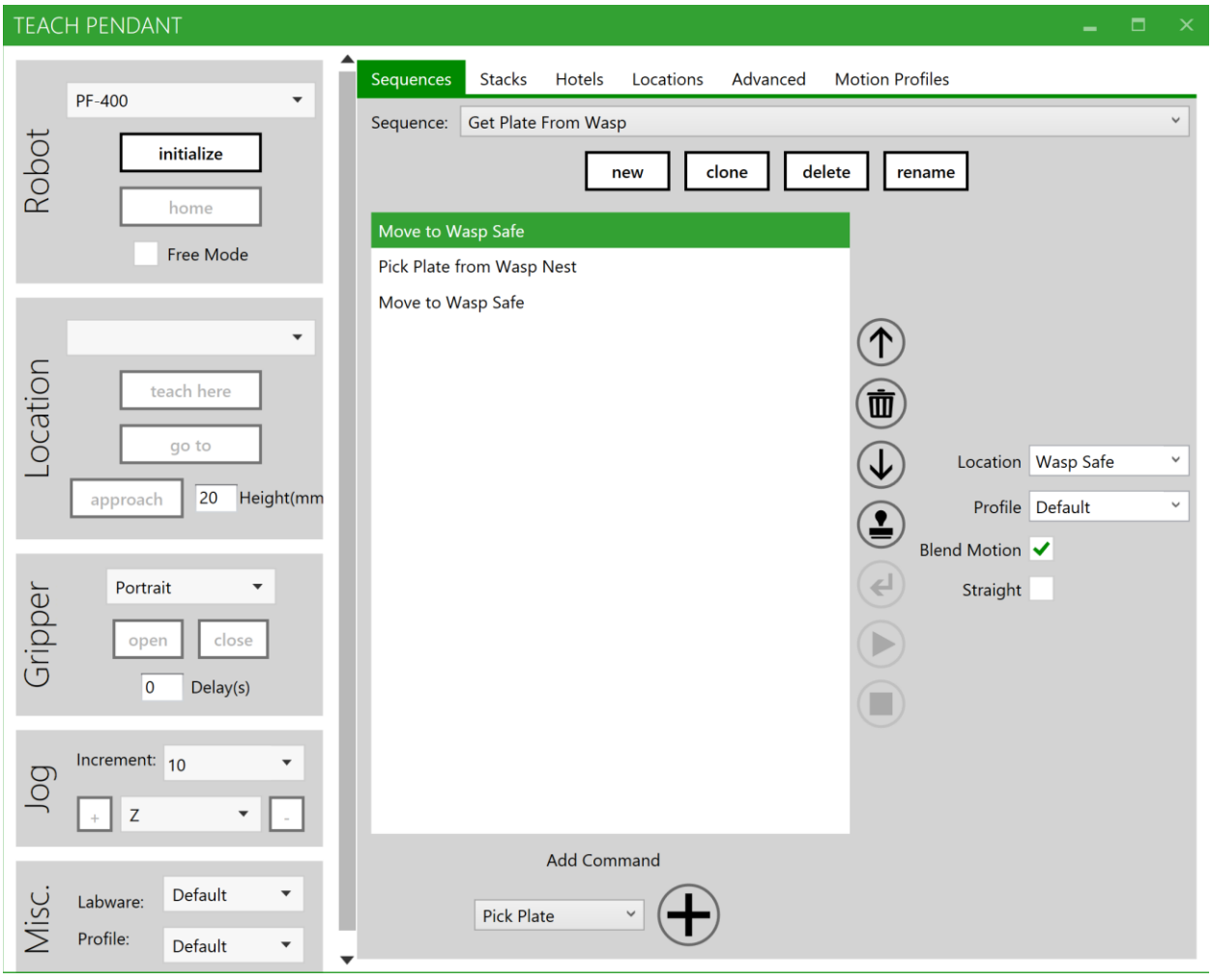


FIGURE 28. TEACH PENDANT

7.1. Basic Logic Structure

The teach pendant encompasses four logic structures:

1. Locations (i.e., for the Wasp a “Wasp Nest” and “Wasp Safe” location)
2. Hotels (Random Access)
3. Stacks (Sequential Access such as LIFO or FIFO)
4. Sequences

Locations

A location (i.e., teach point or position) is simply a place where the arm has been taught to go by name. Most of the necessary locations will be auto-generated. To teach these, go to the Locations tab in the teach pendant.

Note

Each GBG nest has two teach points auto-generated: a nest and safe location. These are automatically incorporated. For example, if the program has a sealer named "Wasp", it will have a "Wasp Nest" position that should be taught exactly where the plate should be grabbed/released. The corresponding "Wasp Safe" should be taught where the arm can retract to after getting/placing a plate.

Typical steps:

1. Have teach pendant open.
2. Click **initialize** button (left panel) to initialize the arm.
3. Make sure a plate is properly gripped by the arm.
4. Check the **Free Mode** (limp arm) box (left panel).
5. Physically place the arm where you want it.
6. Ensure correct location is selected and click **Teach Here** (left panel).
7. Repeat steps 5 and 6 as needed.

Note

You do not have to take arm out of free mode to activate the teach function.

Other tips:

- Make sure the arm is gripping a plate at the same location on the plate for every teach point.
- It is usually best to select one nest as a reference nest. This nest is taught first, and all subsequent teach points are taught after having the robot pick the plate from this reference nest.
- Usually locations and storage need to be taught but for most scenarios the sequences that reference those locations will work just the way they are auto-generated.

- Safe points should be set up such that the arm can move unrestricted from any safe point to any other safe point.

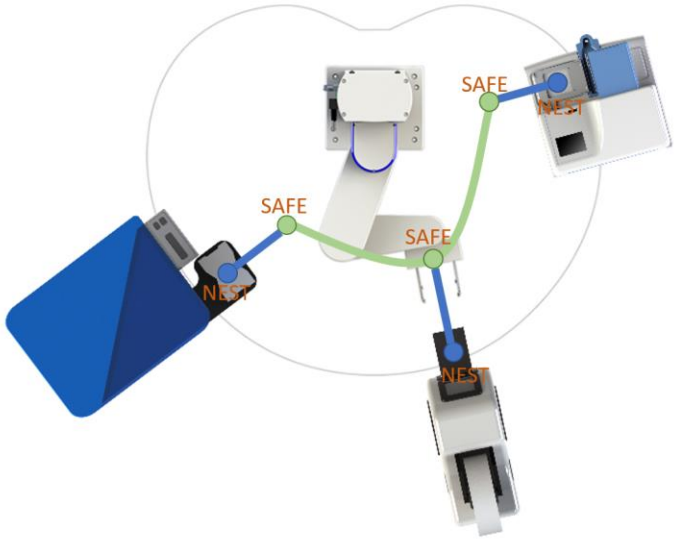


FIGURE 29. SAFE POINT ILLUSTRATION

Hotels

For each hotel defined in a storage instrument's configuration, there should be a corresponding hotel created in the Teach Pendant.

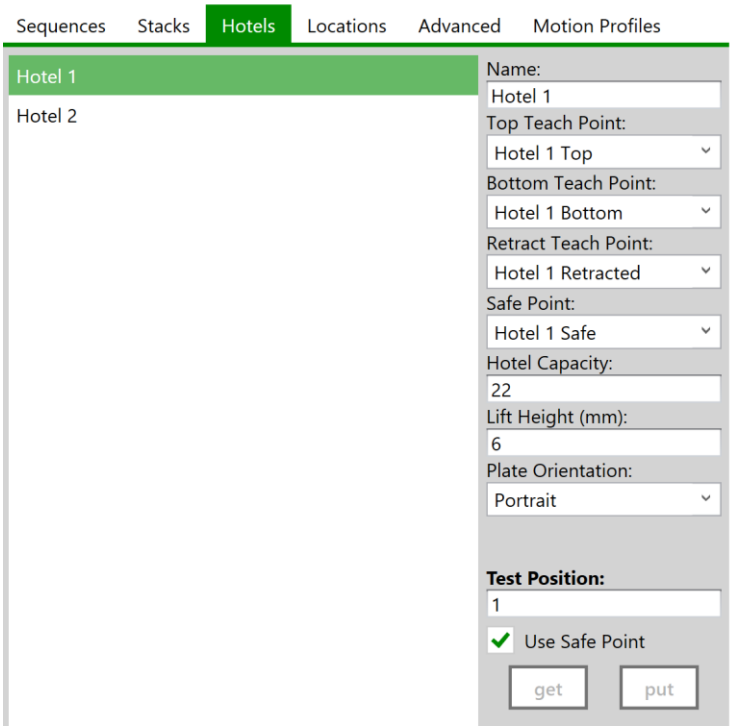


FIGURE 30. TEACH PENDANT: HOTELS TAB

Set the name of the hotel.

1. Select the necessary locations.
    - Top Teach Point is the top plate position.
    - Bottom Teach Point is the bottom plate position.
    - Retract Teach Point defines how far back the arm needs to pull the plate and is typically taught at the same height as the Top Teach Point, but with the plate pulled fully out of the nest and slightly away from the edge of the Top shelf.
2. Set capacity to the number of shelves. This figure is used to interpolate the height of a given shelf.
3. Set the lift height to determine how high a plate is lifted before being pulled out of the shelf.
4. Define whether plates in the hotel are portrait-or landscape-oriented.
5. Test using controls at bottom.

Stacks

Teaching stacks is in principle the same as teaching hotels.

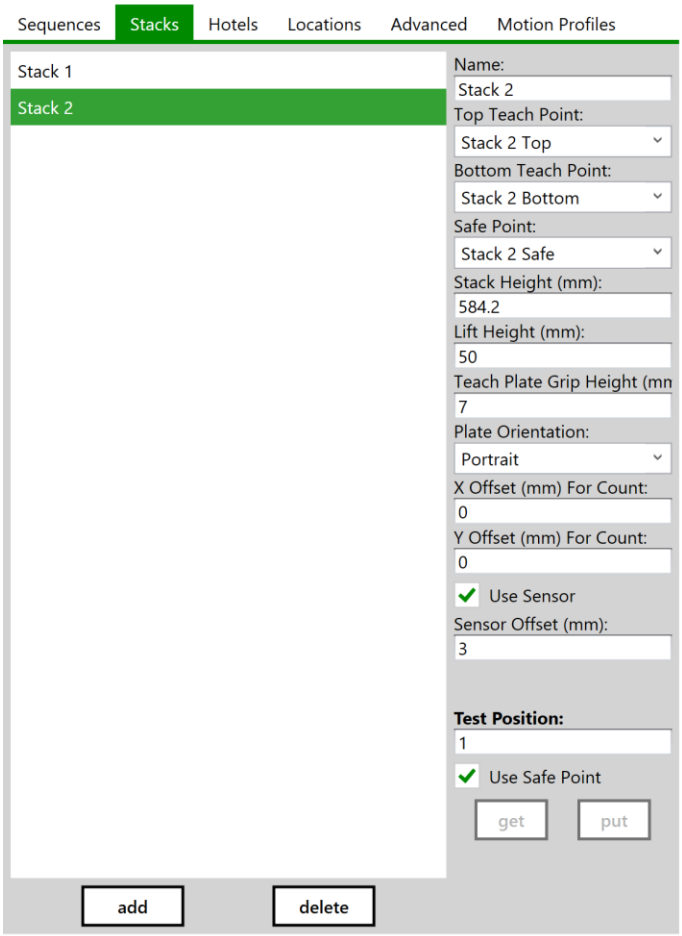


FIGURE 31. TEACHING PENDANT: STACKS TAB

Notes on the parameters:

- Top Teach Point should be equivalent to gripping the highest plate in the stack.
- Bottom Teach Point should be equivalent to gripping the lowest plate in the stack.

Stack height is used to interpolate the height of a given plate number. If the stack is not oriented perfectly in line with the axes of the arm that is OK: interpolation happens in the horizontal axes as well if the top and bottom points are not perfectly on top of each other.

Sequences

GBG uses sequences to define a series of robot motions and actions. Sequences are automatically generated for getting and putting plates into all of the instrument nests in a GBG program.

FIGURE 32. TEACHING PENDANT: SEQUENCES TAB

The typical setup for a sequence is show in Figure 32. The arm will start and end at a safe point. The command **Pick Plate** will open the gripper and move to a position directly over the nest using the specified lift height. It will drop to the nest location, close the gripper, and come back up to the lift height.

Note

Many modifications can be made using sequences. Use the drop-down menu at the bottom of the tab to see possible commands to add.

Chapter 8. Scheduling

8.1. Overview

The Schedule tab allows the user to see the times that past runs were executed and whether or not they completed successfully. It also allows the user to set up runs ahead of time, to run multiple workcell processes at the same time, and to schedule workcell processes to run statically using a static schedule.

8.2. Implementation

In order to schedule a run, the variables for that run should be marked in the Process tab. When the run is scheduled those variables will be presented allowing the user to determine what their values will be when the run actually starts. The main factor with setting variables is that the user must be able to set variable values ahead of time, but those values cannot become active right away since another run may be going. Likewise, multiple runs may be scheduled ahead of time and each can have separate variable values assigned.

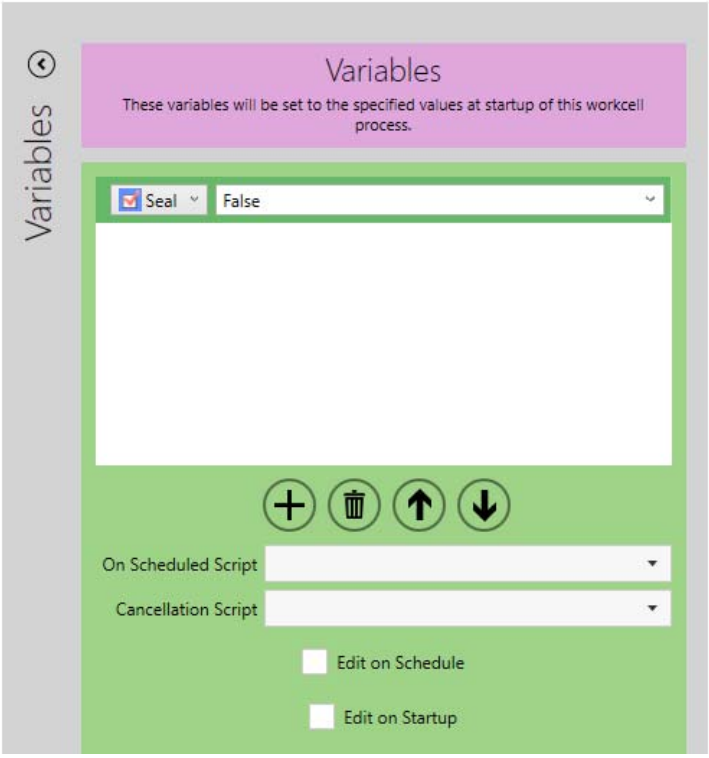


FIGURE 33. PROCESS TAB: VARIABLE SETTER

One additional setting is required in the Process tab if two workcell processes are to run simultaneously. **Allow Simultaneous Execution** must be checked in a workcell process' properties in the Process tab (Figure 34).

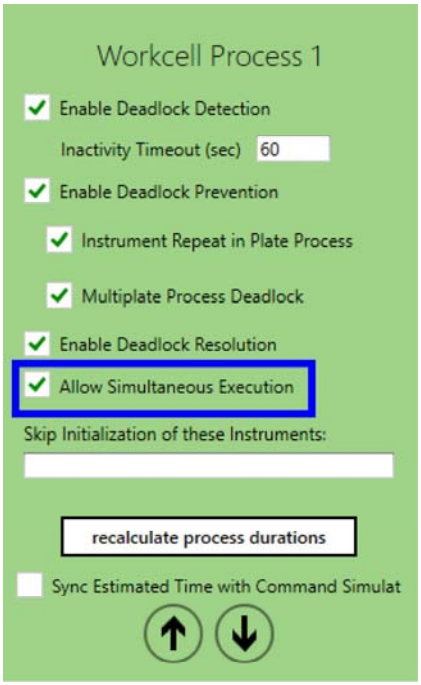
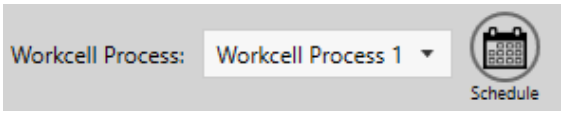


FIGURE 34. ALLOW SIMULTANEOUS EXECUTION

Once the workcell processes are configured in the Process tab as desired, the Schedule tab can be used to run them in parallel or to schedule them. On the upper left of the Schedule tab select the workcell process you wish to schedule and click the calendar button:



A window will appear that allows timing to be set for the run (Figure 35). Set a start time or use the **Run After** option to designate a running workcell process after which the desired run will start. Note that if **Allow Simultaneous Execution** was not checked on all workcell processes involved, the start time may be delayed until all running processes complete.

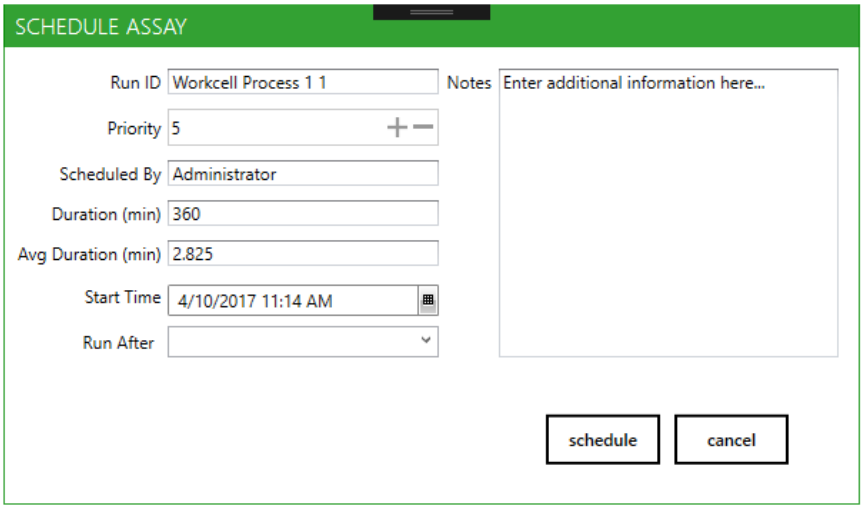


FIGURE 35. SCHEDULE WINDOW

Note

GBG will only automatically interleave multiple workcell processes for parallel execution if the **Allow Simultaneous Execution** property is checked on each workcell process. Otherwise execution of each process will occur after the currently running process has completed.

Chapter 9. Simulating a Run

9.1. Overview

GBG can simulate a run by entering full simulation mode (Figure 36) in which all instruments are simulated, or by selectively simulating certain instruments. Either way results in the method being run as it is designed; commands belonging to simulated instruments will run in simulation mode, meaning that they typically delay for a specified amount of time and then complete.

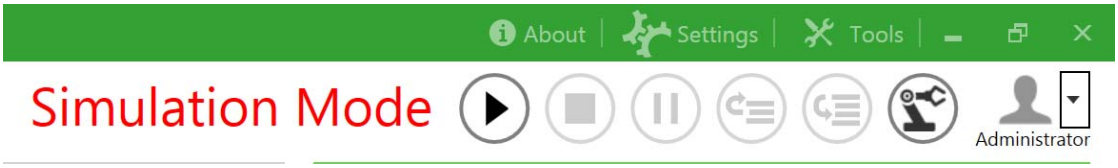


FIGURE 36. FULL SIMULATION MODE INDICATED ON TOP RIGHT IN TOOLBAR

9.2. Full vs Partial Simulation

Full simulation can be activated by a checkbox. This can be found in multiple settings windows, but it is typically selected in the Run tab at the bottom of the window (Figure 37).

FIGURE 37. FULL SIMULATION CONTROLS (LOWER RIGHT OF RUN TAB)

Additional simulation controls allow the speed of a simulation to be accelerated. Alternatively, each instrument can be selectively simulated in its properties. While this can be done in several tabs or interfaces, a common place is in the Run tab in the right-click menu (Figure 38). See page 54 for more information on simulation speed.

Figure 37 shows the options for full simulation:

1. Fast Simulation: skip logging to speed up the simulation (only use if throttling up).
2. Run timing: Start time, elapsed time, speed magnification.
3. Reset Sim Time: Resets the clock (item 2).
4. Throttle: Increased throttle will cause the system to run faster. All simulated commands will execute for a shorter amount of time based on the throttle value. The clock will update accordingly, and the Gantt chart will still attempt to display accurate predictive times rather than the compressed real time of the run.
5. Simulate: This activates/de-activates the full simulation setting.

If not running in full sim mode, you can still simulate an individual instrument by accessing the instrument's properties or by right-clicking on the instrument's image in the Run tab (Figure 38).

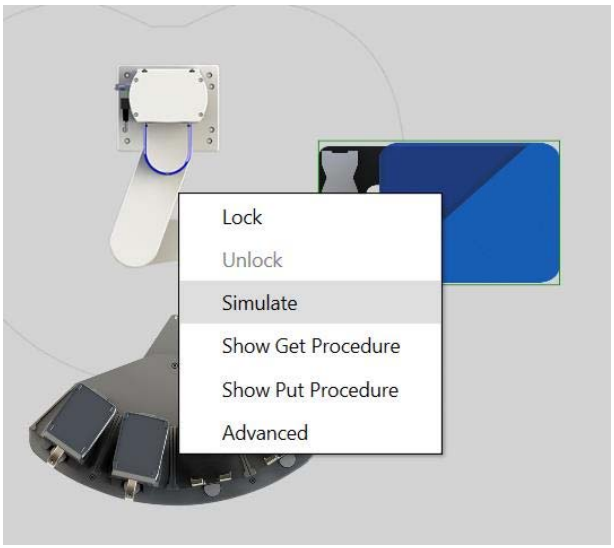


FIGURE 38. SELECTIVE/PARTIAL SIMULATION

9.3. Simulated Times

Specify times for each process in the Process tab. The preferred way to do this is with the tool shown in Figure 39. However, it is also possible to edit these times in the advanced properties of individual instrument processes or by accessing a similar tool to the one shown below using the clock button on a plate process' properties. The most comprehensive solution is the one shown below, where a workcell timing window is opened using the clock button on a workcell process:

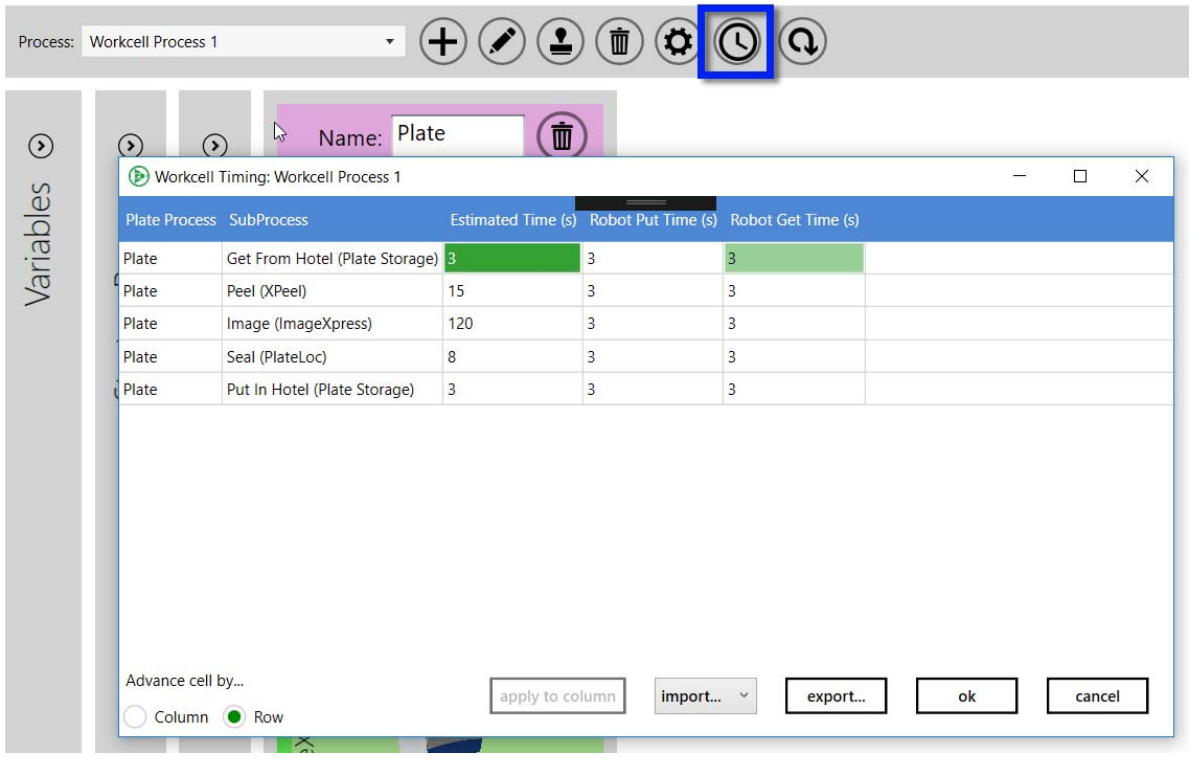


FIGURE 39. WORKCELL PROCESS TIMING WINDOW

The timing interface in Figure 39 allows all process times to be edited. It also allows them to be imported from past runs and to export the times of the current run.

Note that individual commands are auto-generated with simulation properties that will apply the times marked in the Workcell Process Timing window. These properties can be edited to adjust the amount of time each individual instrument command takes in simulation. See Figure 40.

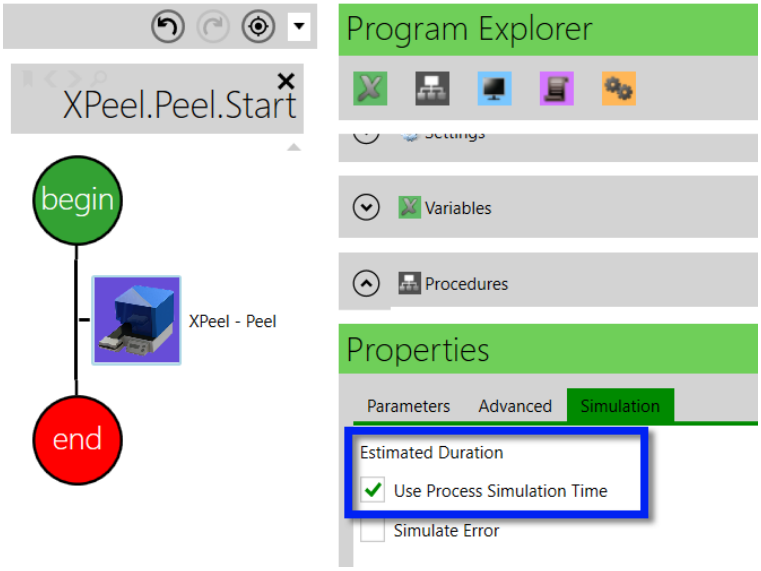


FIGURE 40. INSTRUMENT COMMAND SIMULATION PROPERTIES

In Figure 40, the asynchronous command XPeel – Peel has been auto-generated and will implement the simulation time set for its associated process. If this default action is undesirable, the link can be broken by unchecking this property. In this case a text box becomes available in which a custom sim time can be specified.

9.4. Simulation Speed

See Figure 37 for an image of the sim timing options. To simulate in real time, check **Simulate** (item 5 in that image), but keep the throttle (item 4 in that image) on the far left. Often it is desirable to simulate a run as quickly as possible to ensure changes work or to try timing optimizations. To do this, check **Simulate**, slide the throttle all the way to the right, and optionally check **Fast Simulation** for maximum speed.

9.5. Simulating Errors

If a method designer needs to simulate an error, this should be implemented in the Advanced tab. Any instrument command that can be simulated can also simulate an error. Access the Simulation properties for the desired command (Figure 40), check **Simulate Error**, and fill out the textbox that will become visible with the error message you wish displayed.

9.6. Commands That Do Not Simulate

Not all instrument commands can logically be simulated. For instance, many instruments reference data tables to track worklists. Even in simulation mode these worklists must be used to create an accurate run simulation. Such commands, therefore, typically run the same logic in either live or simulated scenarios. They do not offer any simulation settings since they cannot be simulated. Regarding timing, such commands are typically very short (<1 second) with no actual instrument movement.

Chapter 10. Tools Menu and Managers

The following windows can be accessed from the Tools menu in the top menu bar.

10.1. Archive/Restore Program

This window will back up your program as a zip file and allow you to restore from that file.

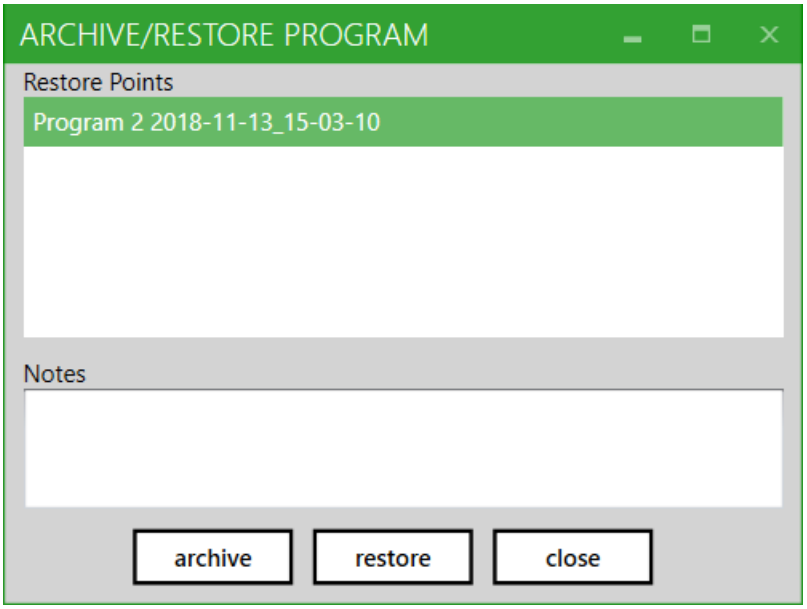


FIGURE 41. ARCHIVE/RESTORE WINDOW

10.2. Error Manager

The Error Manager (Figure 42; also discussed in Chapter 14. Advanced Error Handling) lets the user define a default action for a specific error:

- Ignore: Error will always be ignored.
- Retry: One retry will always be attempted for this error. If the second attempt fails the user is notified with a standard error handling window.
- AlertUser: Default for all errors. Brings up the standard error handling window.

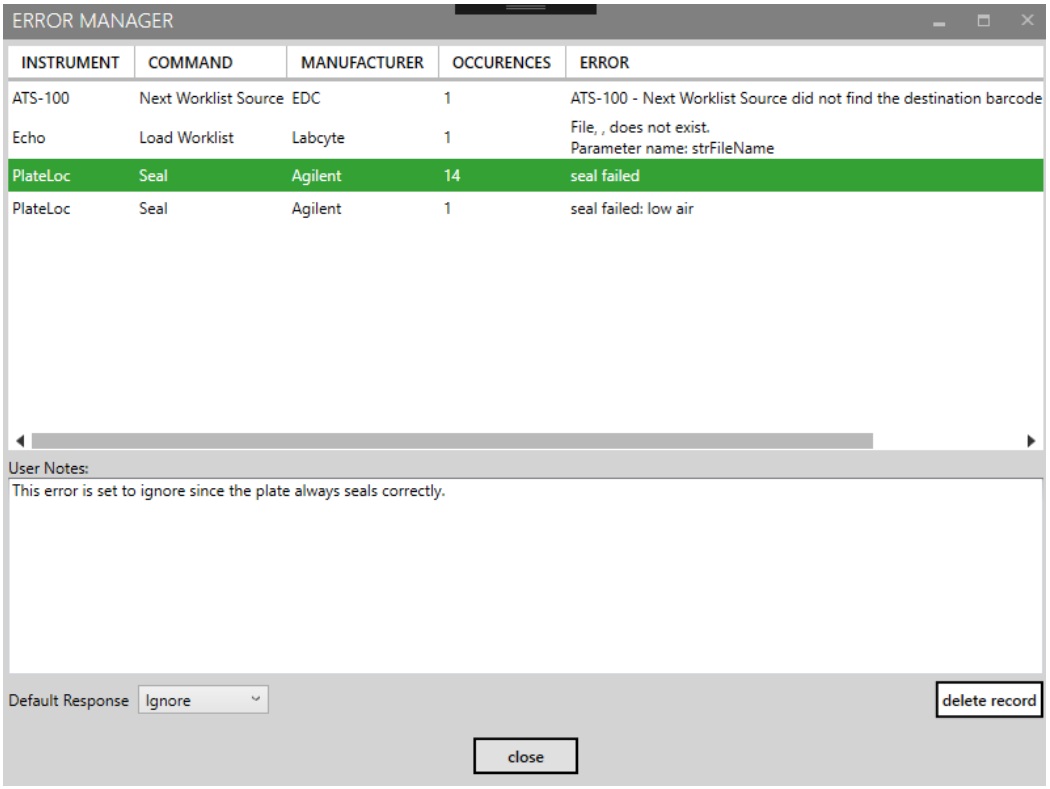


FIGURE 42. ERROR MANAGER

10.3. Archive Manager

The Archive Manager allows the user to view data from past runs.

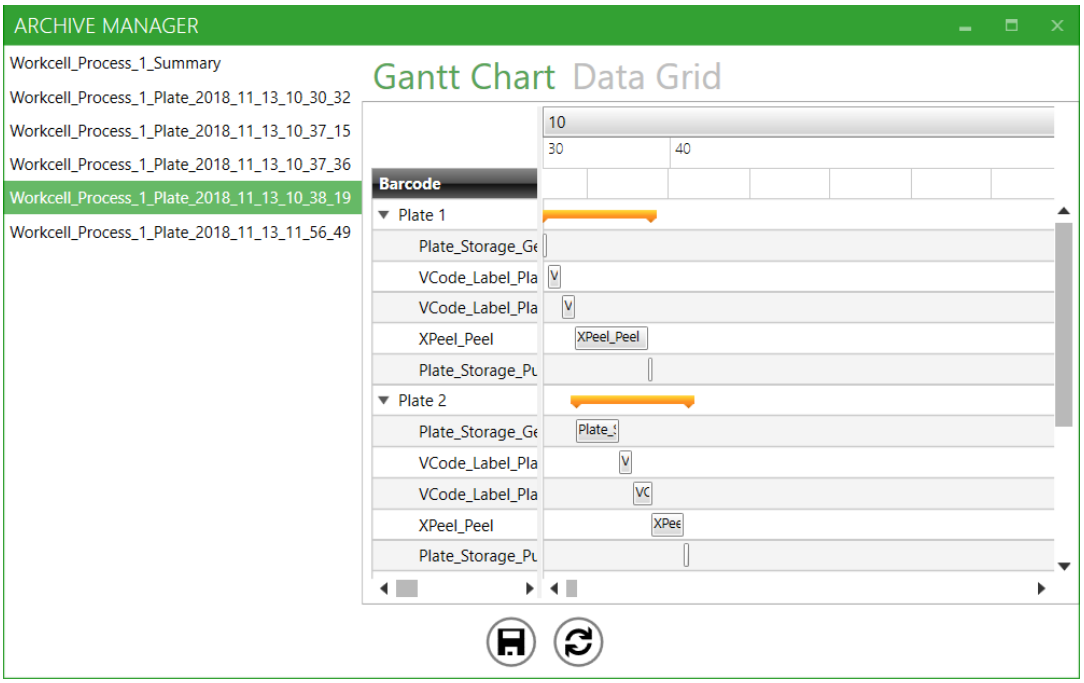


FIGURE 43. ARCHIVE MANAGER

10.4. Process Status View

The Process Status view shows the status of each process. Used primarily to understand the scheduler logic when making method adjustments.

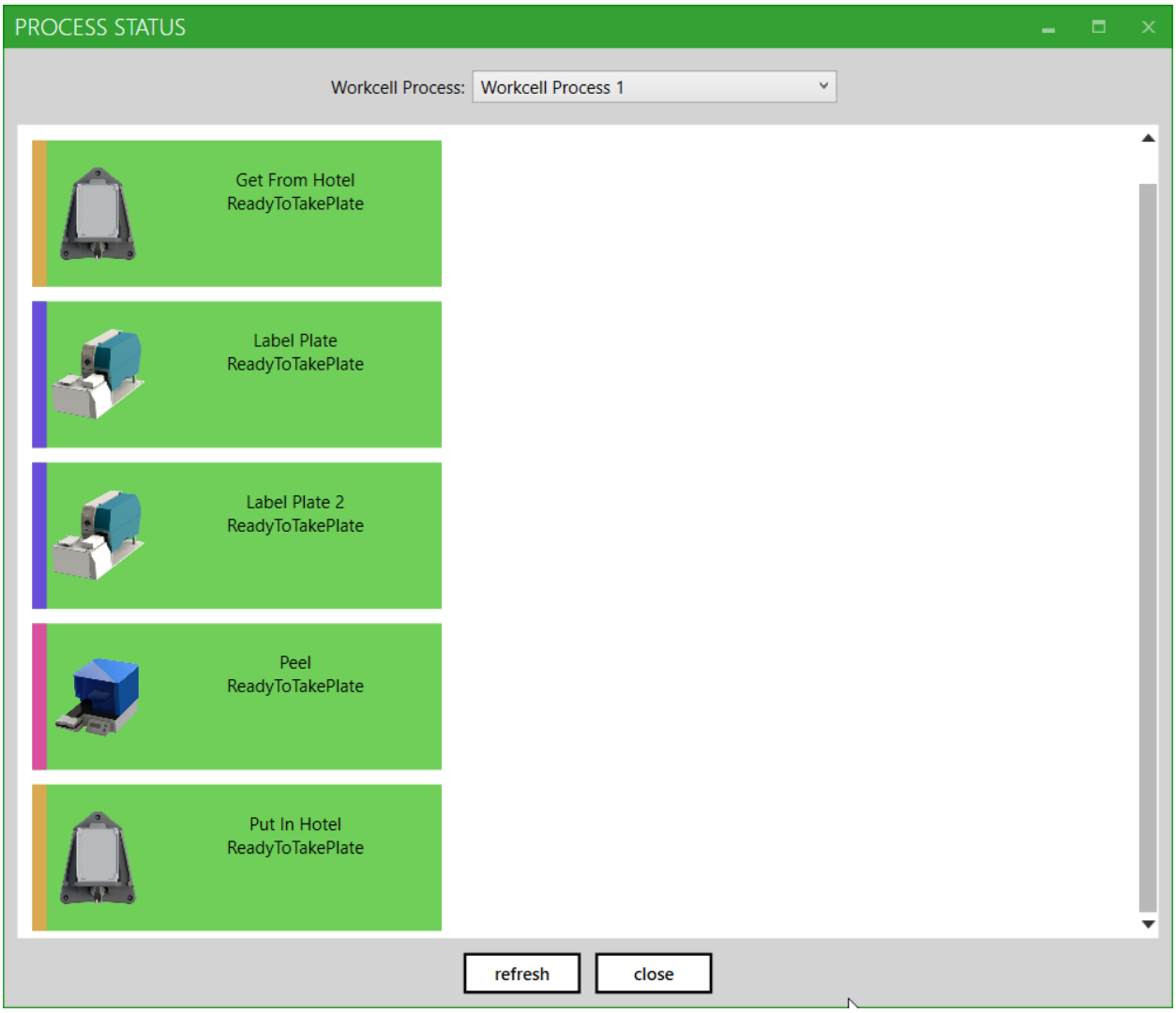


FIGURE 44. PROCESS STATUS VIEW

10.5. Save Runtime Changes

This tool saves the program, overwriting the current state of all files. If changes are made in the middle of a run, this should be used to ensure those changes are propagated.

10.6. Labware Editor

This command opens the labware manager. While some instruments allow for the import of labware, this manager allows others to be added manually. The fields are described below. Note that not all values are strictly required for all operations. If no lids or seals are used, then the three starred values are the only ones that need exact values. Other fields could be set to default values, 0, or false. If only

one type of labware is to be used, even the grip offset can be set to 0 and only the height values require an actual measurement.

- Part Number: Used for inventory and reference.
- Wells: Number of wells in this labware type.
- *Height in mm: Total height of plate.
- *Stack Height in mm: Height a plate takes in the stack, i.e., plate height minus the nesting effect of the skirt.
- Lid Height: Total height of the lid.
- Is Lidded: Deprecated, but present for backwards compatibility. Use lidded Boolean in storage table
- *Grip Offset: How much higher/lower to grab this plate type; based on the actual teach points (done with the default '0' labware), the Z offset used to grab/release this type of labware.
- Lidded Stack Height: Height a plate takes in the stack when lidded.
- Lid Grip Offset: How much higher/lower to grab the lid off the plate.
- Lid Placement Offset: How much higher/lower to grab the lid when placing on a shelf.
- Seal Thickness: Thickness of seal, particularly as it applies to stack spacing.
- Other: various instruments will add columns for their own use.

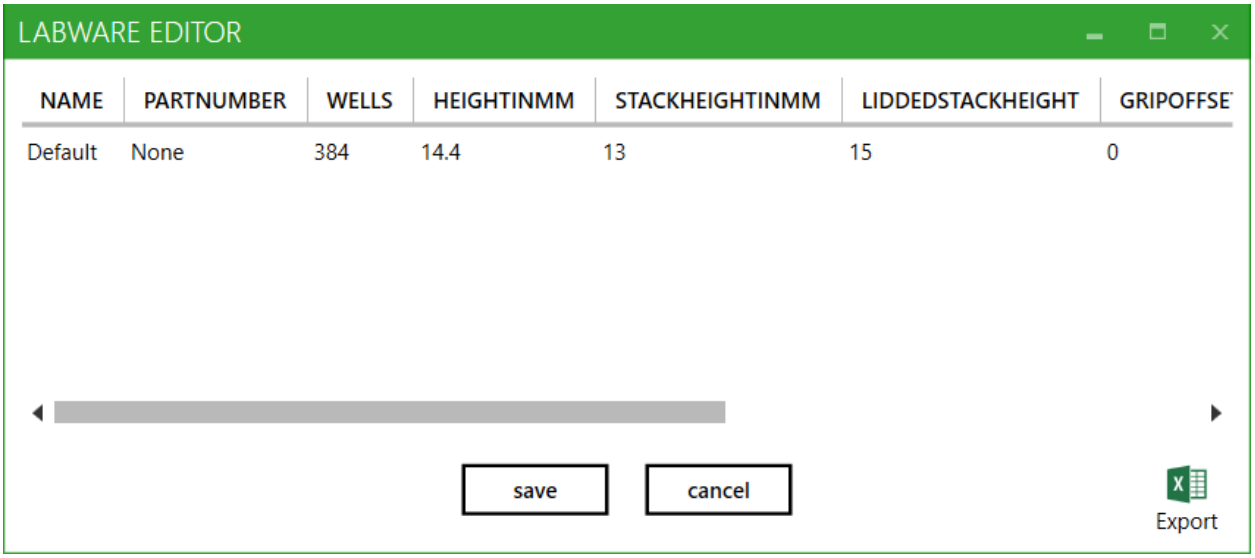


FIGURE 45. LABWARE EDITOR

Chapter 11. User Accounts

11.1. Overview

User accounts can be created and edited from the menu at the top right of GBG (globally available next to the Teach Pendant button).

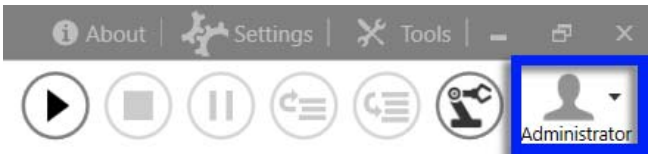


FIGURE 46. ACCESS/EDIT/SWITCH GBG USER ACCOUNTS

Selecting **Manage Users** from this menu allows user accounts to be edited (though this can only be accessed from an admin level account). **Add User** brings up the New User screen (Figure 47) and allows new users to be created.

11.2. User Types

There are three user types, but different permissions can be specified for each type:

1. Administrator: Full access, including the ability to alter other user accounts.
2. Editor: Full access to the program, but not to user accounts (except to alter personal password).
3. User: Program access limited to essentials and ability to alter personal password.

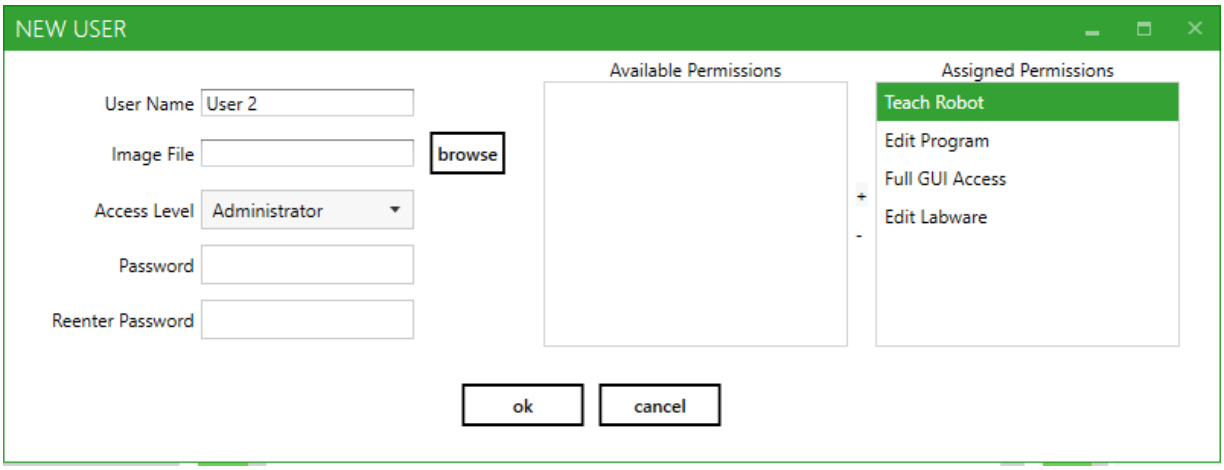


FIGURE 47. NEW USER SCREEN

As shown in Figure 47, there are four permissions that can be added or removed from each user:

- Teach Robot: Ability to access the Teach Pendant.
- Edit Program: Without this, the user is restricted to the Start, Schedule, and Run tabs.
- Full GUI Access: Without this, the user is restricted to the simplified basic user view (Figure 48).
- Edit Labware: Ability to access the Labware Editor.

11.3. Basic User View

If a user does not have Full GUI Access permission, then GBG will restrict that user's GUI to the basic screen in Figure 48.

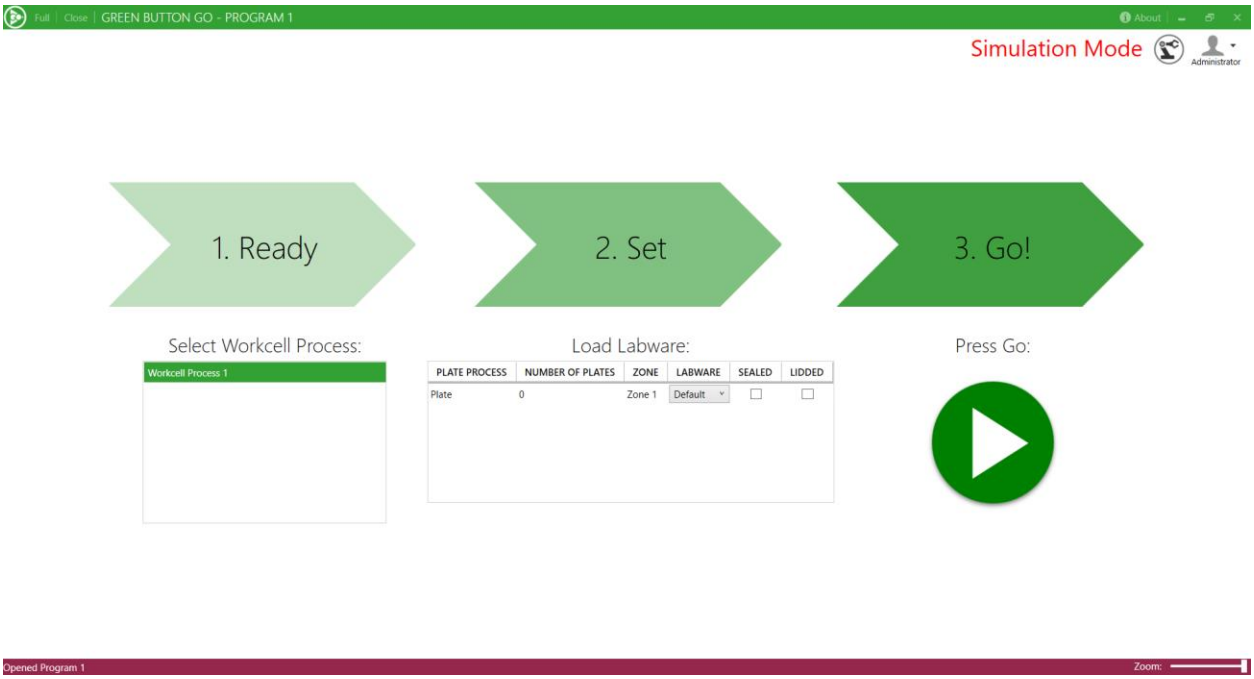


FIGURE 48. BASIC USER VIEW

This basic interface allows a user to input the number of plates to run and whether those plates are sealed or lidded. After the user clicks the **Play** button, the view changes to offer feedback on the progress of the run. This is a simplified interface for labs where users don't want or need detailed state information and only need to know basic timing information. This view is available to any user type by clicking **Basic** on the top left margin of GBG. If the full GUI is available, then **Full** will also show in the top left as a way to return to the full view.

Chapter 12. Common Tasks/Optimizations

Additional common tasks that may be performed in GBG are described below.

12.1. Determine Where Plates Enter/Exit the System

Plate Sources

Because GBG is largely plate-centric, the system must know where plates enter and leave the system. In addition, the system must know when a source has given all its plates and is finished. These details are handled in the driver code, and therefore vary depending on the instrument being used as a source. Fundamentally, most sources consist of a LIFO or FIFO stack or a random-access hotel. Required user input at design time will consist of some combination of instrument properties and process properties. Required user input at run time typically includes a pop-up window or some automated input of the contents (plates).

Design Time

The setup for a plate source requires a few different steps. First, you must determine what type of holders will be used (stacks vs hotels). For an instrument that has multiple source types, such as a carousel, this will be configured using the instrument properties in the Layout tab. Next set up the desired source process in the Process tab. At the top of the plate process you will put a Get process. For an incubator, which only has random access hotels, you will drag on a process called Get Plate. For a carousel which can take random access or LIFO, you will use either Get Plate From Hotel or Get Plate From Stack. You must then choose a strategy in that process' properties. For instance, you may choose to get all plates in a certain zone of the storage device. Conversely, you may choose to draw plates from random access storage based on a worklist.

Run Time

As mentioned above, there are numerous possible configurations for introducing plates to a run. Essentially, the Get process must know when it has gotten all plates. For a typical random-access hotel, the user must ensure that the barcodes to be used are recorded in the random-access table which will be presented at the beginning of the run. For a typical LIFO or FIFO stack, the number of plates in the stack must be entered in the stack table as well as the initial direction (input/output) of each stack. While these values are often entered by the user, it is also possible to set such values in a script or from some other data source, or to have the source or robot count the plates automatically.

Plate Destination

Destinations are configured in much the same way as sources. Again, much depends on the instrument. A trash chute, for instance, keeps no plate data and always reports ready to accept a plate. A carousel of stacks, however, will evaluate whether or not it has room to accept a plate. Much of the setup overlaps, since setting up the source also means inputting destination data (i.e., which stacks are input and which are output).

Zones

Most sources/destinations have a Zone column in their tables. On larger or more complex integrations, this allows part of a source to be set aside for one job while another section is used by other plate processes or even other workcell processes. Zones within a source instrument can be specified from

their configuration windows, reached through the instrument properties. Zones are specified for a plate process by clicking on the source process at the top of the plate process and adjusting its strategy and properties.

Starting on a Nest

Biosero's Plate Nest driver can be used to start a plate on a nest that would not normally be considered storage, but this should only be implemented by an advanced method designer and may require modifications in the Advanced tab to achieve the desired specifications.

Reference

For more on GBG storage, see Appendix D: Biosero Universal Storage Guide.

12.2. Low Level Execution

Running a procedure

Right-click on a procedure in the Program Explorer (Advanced tab) to bring up its context menu. Select **Run** to run the procedure. Note that this should be used to run a single procedure while the program is otherwise stopped.

Running a command that is in the workflow

Right-click on a command in the workflow to bring up its context menu. Select **Run** to run the command. Note that, similarly to running a procedure stand-alone, this is used for testing and is not part of normal, automated execution.

Running an instrument command that is not in the workflow

Right-click on an instrument in the Program Explorer (Advanced tab) and choose **Advanced**. This opens a window that gives you access to all commands for that instrument. From this window, you can connect (establish communication with the device), disconnect, and run commands. This allows low-level recovery steps in case of a complex error such as an instrument completely failing in the middle of a run, and is intended for use in conjunction with the **Modify** choice on the error popup. Note that this advanced command execution window is also available through controls in the Layout and Run tabs.

12.3. Setting Labware

Overview

If a single labware type is used, it may not be necessary to add any labware-specific details to the program. If multiple types are used, however, it is often useful to define those types in the labware manager. With a few simple tools, GBG will propagate the labware type name for each plate, allowing the system and individual instruments to make intelligent decisions. The best example is the GBG robotic arm architecture, which allows plates to be gripped differently based on their labware type.

Type propagation

The labware name is propagated through a run and stored in the storage tables and run tables. When a plate is active its labware type is indicated in the {Current Labware} variable. The type can be set or altered in three ways:

1. The name is set at the top of the Plate Process.
2. It is overwritten if a labware name is defined in storage.
3. It is finally overwritten if the Current Labware variable is manually changed before a plate is grabbed off an instrument, i.e., in a Get Plate From procedure.

Labware Editor

Labware can be specified in the Labware Editor, which is available both through the global Tools menu and through the Process tab.

LABWARE EDITOR

NAME	PARTNUMBER	WELLS	HEIGHTINMM	STACKHEIGHTINMM	LIDDEDSTACKHEIGHT	GRIPOFFSET
Default	None	384	14.4	13	15	0
384 Nexus/Aurora P/N: 32611 Source	?	384	12	11	0	0
384 Nexus/Aurora P/N: 32711 Source	?	384	12	11	0	0
384 Corning P/N: 3711 Source	?	384	12	11	0	0

If your integration uses a single plate type, it is typical to use the Default plate type. In this case, the Labware Editor will likely not be used. If there are multiple plate types, however, the Labware Editor can be used to define the labware in a number of useful ways:

- The labware name gets propagated throughout the run for each plate.
    - The name is set at the top of the Plate Process but can be overwritten as stated above.
    - This name is often used in user-defined method additions. (if {Current Labware} equals x then…)
- The GripOffset is a valuable tool that works in conjunction with the robotic arm.
    - At runtime, the system looks up the grip offset based on the {Current Labware} variable.
    - The arm adjusts its grip based on this offset.
    - For instance, an offset of 2 will cause the arm to grab the plate in question 2 mm higher than where the location was originally taught.
    - This provides a quick way of automatically adjusting for physically different plates.
- Lidded Stack Height
    - This allows the arm to automatically adjust for the height of lids within a stack.
- Driver specific columns

Some instruments, such as the ATS, require that GBG submit the labware type being used. In some cases, a driver adds a column to the Labware Editor table to allow the instrument to receive a specific alternate labware name. In other words, the instrument driver looks at the {Current Labware} variable and feeds the alternate name to the instrument.

12.4. Static Scheduling

Static scheduling can be used to establish exact times at which processes will run. This allows the timing for each plate to be precisely predicted and made consistent but sacrifices throughput in the process since timing optimizations are ignored. Often a good compromise is to set the throttle on a plate source's Get Plate process. This allows plates to be brought into the integration on a consistent schedule while still allowing all timing optimizations once a plate is on the deck. However, if further static timing is required, check **Use Static Schedule** on all desired processes (see bottom of Figure 49). Then use the

Static Scheduler button in the Schedule tab to start the run. Note that static scheduling should only be implemented by an advanced method designer and the method must be vetted for compatibility with the static system.

12.5. Priorities

Priorities can be defined in the Process tab. Select an instrument process and expand its advanced properties (Figure 49). One of these advanced properties is **Priority**. By default, all priorities are set to 5.

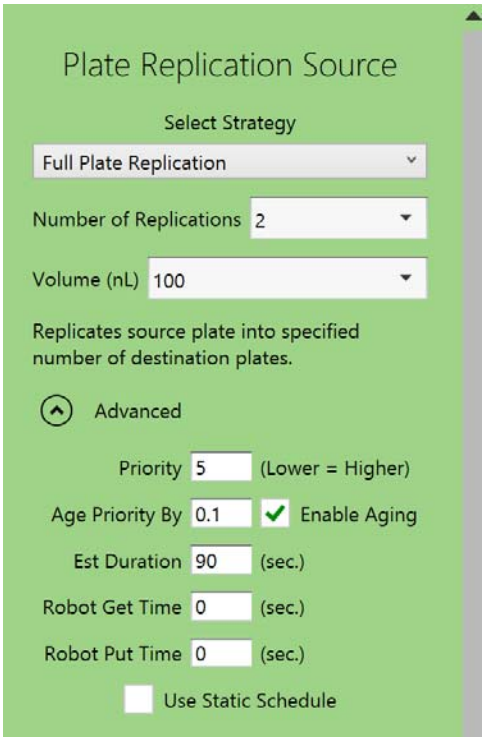


FIGURE 49. PROCESS TAB: INSTRUMENT PROCESS ADVANCED PROPERTIES

A lower number indicates a higher priority, so changing this to 1 means that this process' status will be evaluated before other processes. Because of the way the Run Scheduler command analyzes readiness, the process with the lowest number will give up its plate, or pass on its plate, first.
An example when this might be important is a method with an incubation in the middle in which the timing is very critical. The plates must be passed out of incubation and onto a reader at exactly an hour. A low-numbered priority must be set on the incubator in order to ensure that the scheduler picks this plate movement when it becomes available rather than choosing another available action.

Aging

Priorities also employ an "aging" mechanism which can be set in the **Age Priority By** field. This ensures that all processes will eventually be prioritized for execution. This happens inside the Run Scheduler command and prevents a common scheduler deficiency: starvation. Starvation would occur if there are too many high priority processes to run, so that there is never enough time for a lower priority process to be considered. Aging dynamically changes the priority each time a process is not evaluated. This way, a process will eventually be evaluated first. When it runs, the priority is reset to the default user-specified value.

12.6. Selectively Skip a Process, Selectively Wait Before Executing a Process

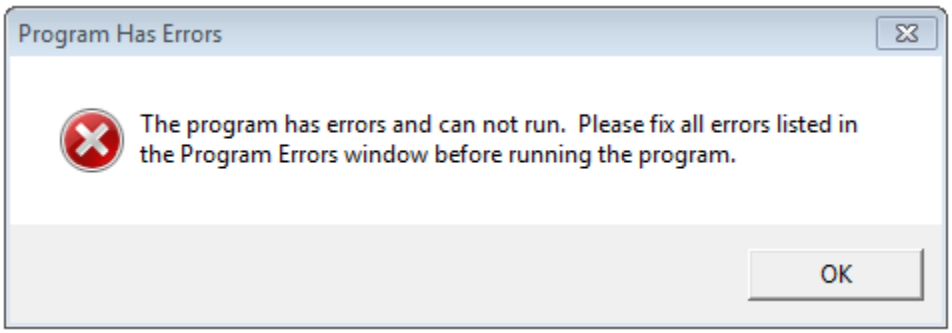
For information on these common tasks, see Appendix B: Instrument Process Status and Ready Evaluations, page 78.

Chapter 13. Basic Error Handling

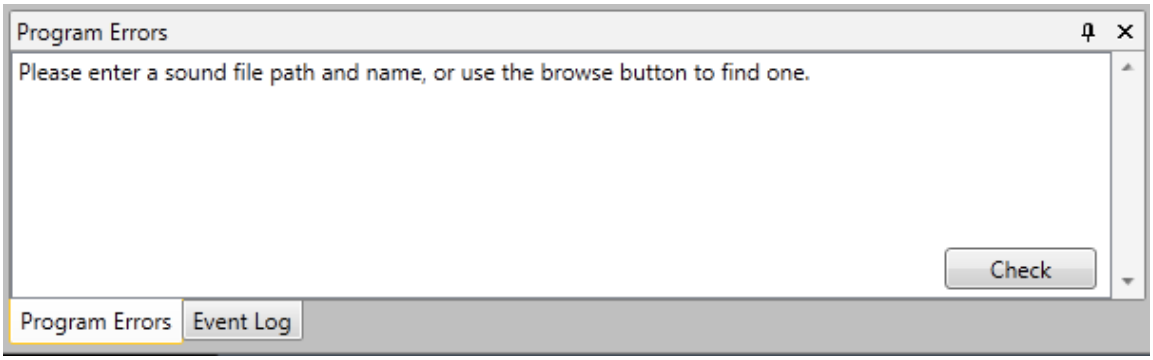
This chapter covers the most commonly used error handling interfaces: errors detected before a run (compile-time errors) and errors occurring during a run, called run-time errors. For advanced options for error handling, which includes errors arising due to method design, see Chapter 14. Advanced Error Handling.

13.1. Validation/Compile Time Errors

If you build a method, click **Run**, and get a prompt like the following, then you are seeing a compile-time error:



The message above indicates that GBG's validation system has caught an issue with the method design, such as a missing property. Go to the Advanced tab, and at the bottom you will see the Program Errors panel.



Double-clicking on a line in this interface will bring up the procedure, script, screen, or property set that is invalid. After making adjustments, click **Check** to re-evaluate for errors. When all errors have been resolved the program will be able to run.

13.2. Run Time Errors

These errors occur while the program is running and typically come up due to either a logic error in the method (i.e., an incorrect number of plates being provided to one swim lane) or an instrument failure.

The window in Figure 50 is presented when such an error occurs and allows the user to choose the desired response.

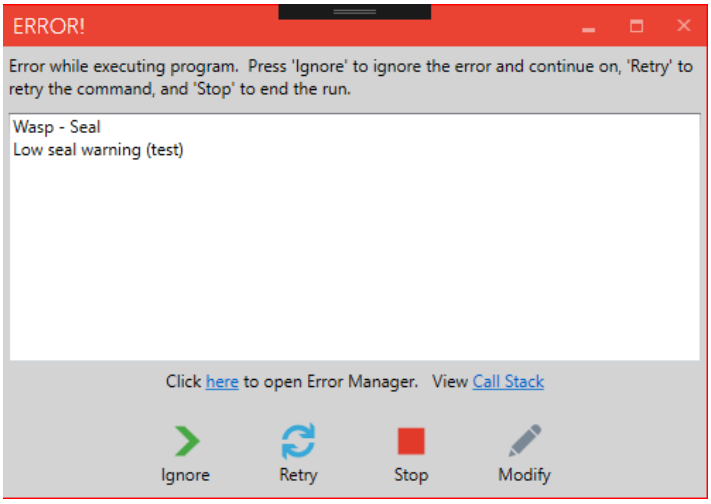


FIGURE 50. ERROR RESPONSE: BASIC ERROR HANDLING WINDOW

This window appears by default any time a command has an error. From here you can implement a simple one-click solution or make changes to the program to recover. Options within this window include:

1. **Ignore:** Continue execution at the start of the next command after the one which erred.
2. **Retry:** Continue execution by retrying the erred command.
3. **Stop:** Execute shutdown protocol for each driver and stop run.
4. **Modify:** Pause integration to allow changes to be made. Also allows Teach Pendant to be opened and used. Once **Play** is clicked, execution will continue with the command which erred.
5. **Error Manager:** Allows default action to be specified (i.e., auto-retry).
6. **Call Stack:** Shows code running at time of error (can be sent to programming team on request).

Modify

The **Modify** button in the Error Response window allows for advanced recovery. Many tools are available for evaluation of the problem. The normal method-building tools can then be used to adjust the method as needed.

Note

If a restrictive user account is being used, only a subset of the error handling options above will be available.

Chapter 14. Advanced Error Handling

"Failure is not an option."

For basic error handing, such as an arm crash or a low-seal error on a sealer, see Chapter 13. Basic Error Handling.

This chapter covers advanced options for error handling, which includes errors arising due to method design. For example, if a method includes an incubation, peel, and read, it might also have validation to ensure plates don't go overtime on the incubation. If an error arises out of this validation, advanced error handling would be needed to determine a resolution. Possible resolutions might include:

- Throttling the plate input at the start of the plate process to prevent a bottleneck.
- Pooling a second reader to allow increased throughput.
- Tracking reader availability and using it to gate the plates as they pass through an earlier step.
- If the problem is not related to incubation time but rather time sitting peeled on the deck, then checking **Wait Until Next Process Ready** on the peel process should resolve the issue.

Advanced error handling should be done by an advanced GBG method designer. Some of the available tools are highlighted below.

14.1. Process Status View and Situational Awareness

If a run-time error comes up that appears to require an advanced resolution, click **Modify** and attempt to assess the state of the system, both physically and in the software. What actually went wrong? Does GBG recognize the root issue? Is it in the correct state, or do some variables need to be adjusted?

The first place to look is the Run tab, which shows which instrument is in error as well as where plates are. If there is any confusion about why GBG moved a plate forward (or did not) consult the Process Status View found in the Tools menu (Figure 51).

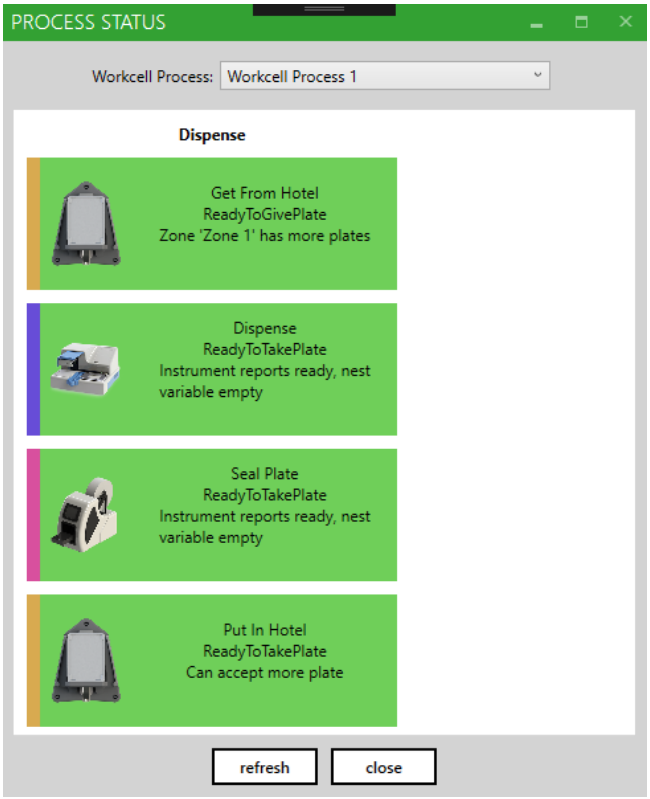


FIGURE 51. PROCESS STATUS VIEW

Edits

Once you understand what happened, fix it with some combination of changes. These can be done with the system stopped or paused, which allows for real time recovery.

- Modify an arm sequence in the Teach Pendant.
- Reteach Teach Points.
- Change variable values. Find variables in the Advanced tab. Double-click one to change its current value.
- Edit data tables. Like variables, these are found in the Program Explorer of the Advanced tab. While the system is paused, double-click one to open and edit it. For instance, you might find that the wrong plate got enqueued. Consider editing the data table to remove the faulty reference.
- Data tables, found in Program Explorer in the Advanced tab, are also editable. Use caution though; this should only be done by an expert method designer.
- Method changes: Add or remove instruments, processes, procedures, variables, logging, custom validation, or anything else you set up before running. With careful attention to the system state, it is almost always possible to continue a run without stopping it (though this is not always necessary or recommended).
- Adjust Ready Evaluations, Skip Evaluations, or Deadlock settings.

This is also a good time to use the Process/Procedure View tool. It is found under the Tools menu and allows you to see which procedure might be a likely place to input low-level changes.

14.2. Canceling a Plate

Occasionally it is necessary to remove a plate from a run because the plate has been fouled up or because of a mistake in the method design. At runtime, a plate can be canceled by right-clicking on its barcode in the Run tab's Gantt chart (Figure 52).

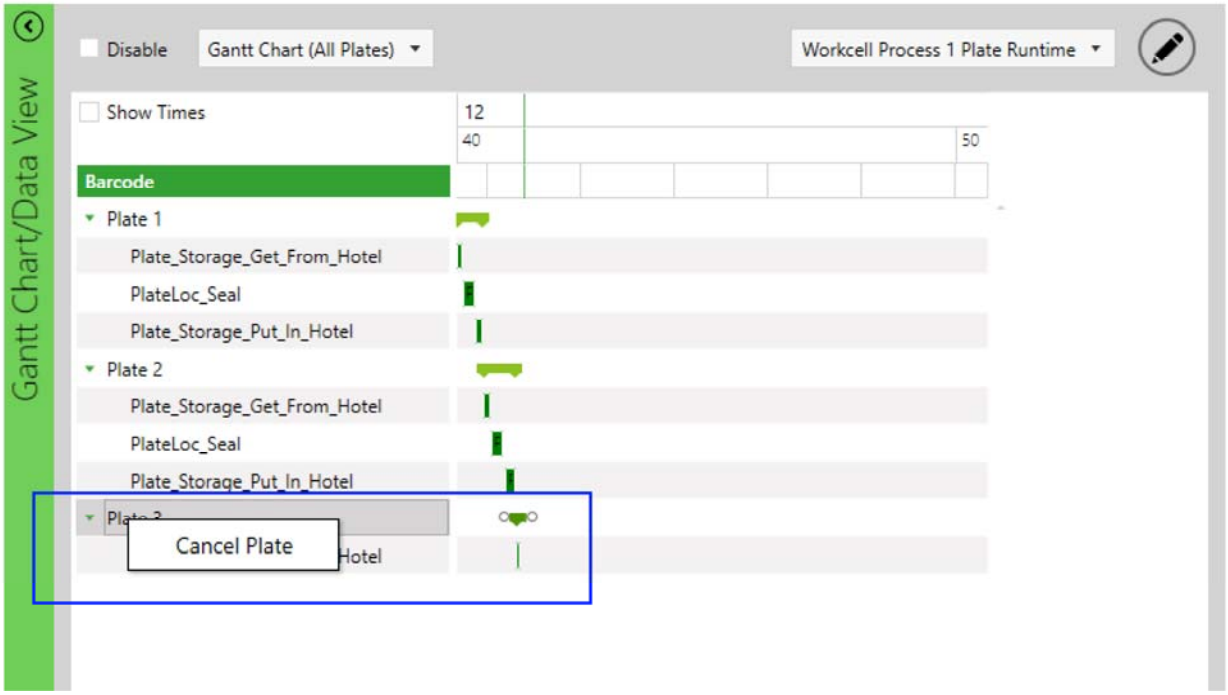


FIGURE 52. RUN TAB: GANTT CHART

Selecting **Cancel Plate** will remove the plate's barcode from any instrument nests and mark the plate as completed in the runtime table that backs the Gantt chart. Note that if the plate has already been placed in storage then the storage table may need to be manually edited. For instance, a plate in incubation should be removed from the storage table to prevent being brought out again.

14.3. Error Manager

The Error Manager (also covered in Chapter 10. Tools Menu and Managers) presents another option for responding to errors.

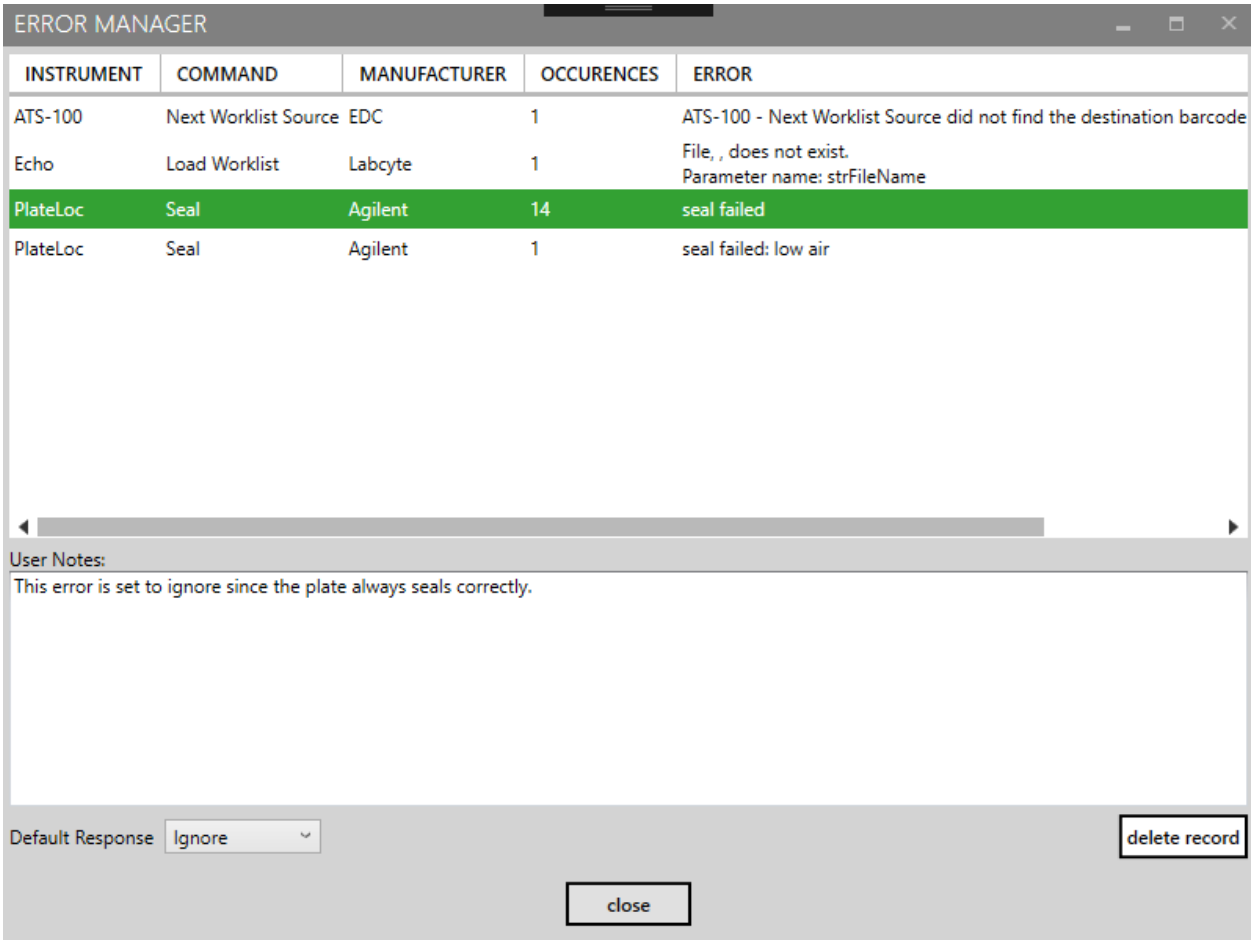


FIGURE 53. ERROR MANAGER

This tool shows all errors that have occurred in a program, allows the user to add notes for a given error, and allows a default action to be specified. All errors, by default, will be shown to the user unless this default action is changed. Options for default actions include:

- **Ignore**: No action is taken on this error. The program does not stop, and a popup is not presented. However, the ProgramErrorMessage still fires, meaning that tools and plugins that subscribe to GBG messages will still register the error. (This is similar to clicking **ignore** in the normal error popup).
- **Retry**: The command will be retried once before the user is notified. (This is similar to clicking **retry** in the normal error popup).
- **Alert User**: Brings up the normal error popup (default action for all errors unless otherwise specified).

In the example in Figure 53, the PlateLoc has been set to simulate two different errors. One of them has occurred 14 times, the user has registered a note, and the default action has been changed to **Ignore**. This error will no longer stop the program or be presented to the user. Alternatively, this could have been set to **Retry**, in which case the sealer would be asked to seal one additional time after the error occurred. If this second seal iteration errs, then the user is notified. The best solution is totally

dependent on the specifics of the situation. Note that in the example in Figure 53 the sealer presents two errors that are similar, but undistinguishable to a computer. This highlights the care with which this tool must be used. It should only be used by advanced user.

14.4. Error Notification

A program can be configured for a number of error notification options. The most common ones include:

- Normal error notification screen
- Light stack signals
- Error sound (or song!)
- Email

14.5. Error Simulation

To test error responses on your simulation system, go to the Advanced tab and click on the command you wish to simulate as an error. In its Properties window, go to the Simulation tab (Figure 54) and check **Simulate Error**. Optionally, you may add a message.

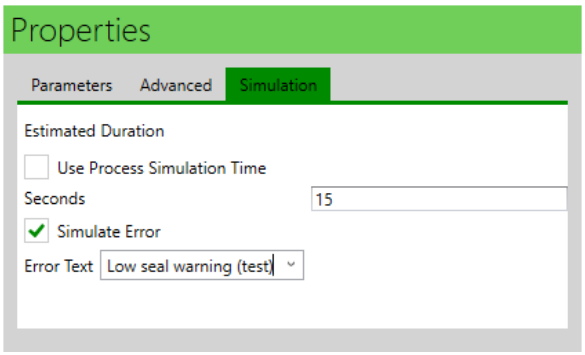


FIGURE 54. SIMULATION TAB IN INSTRUMENT PROPERTIES WINDOW

Chapter 15. Other Troubleshooting

15.1. Instruments Do Not Load

If an instrument will not load when attempting to use the **Add a New Instrument** command:

1. Navigate to the folder named Devices at the <<C:\Program Files (x86)\Green Button Go\Drivers>> on your computer.
2. Open the Devices folder and copy all of the files.
3. Create an empty folder on your desktop and paste the copied files into this folder.
4. Right-click on each device driver file and select **Properties** from the popup list.
5. Uncheck the **Hidden** check box, click **Apply**, and close the window.
6. Do this for all of the device drivers, then copy these files and paste them into the original Device folder at <<C:\Program Files (x86)\Green Button Go\Drivers>>.
7. Replace the current files if asked.

15.2. Deadlocks

Deadlock Detection

Deadlocks can be automatically detected when then integration is running but sits idle for too long. This check can be enabled or disabled, or the time changed, in the properties of a workcell process. For more information on the status and decision-making of the system go to the Tools menu in the upper right and select **Process Status Record**. The heart of software's automation is the Run Scheduler command, which asks processes for their status and then queues up tasks based on these responses. The Process Status Record tool will give you the latest status reported by each process, thereby allowing you to see where a deadlock or bottleneck has occurred.

Deadlock Prevention

Workcell Process Properties Related to Deadlocks:

FIGURE 55. WORKCELL PROCESS PROPERTIES RELATED TO DEADLOCKS

15.3. Crash Recovery System

The recovery system allows for run recovery after a computer crash or other system failure. (For physical robot crashes, refer to Chapter 13. Basic Error Handling). Recoveries can also be initiated after the user has stopped the integration in the middle of a run. In this way, a user may stop an integration that is operating incorrectly, resolve the issue, and resume the run with minimal interruption.

This feature should only be used by advanced users. This tool is dependent on the drivers being used and the method being run. It requires more user interaction on larger integrations, when using multiplate processes, when using processes that require additional data tables, and when customizations have been performed on the system. Recovering a run in the middle adds a layer of complexity to automated scheduling, which is already a complex system. For this reason, users of the recovery system should read this section thoroughly and familiarize themselves with their integration.

Overview

The recovery system only saves data from the most recent run of a given workcell process. To initiate a recovery, go to the Schedule tab, select a line in the schedule chart (ensure that it is the most recent run of the given workcell process), and click the button below with the "O" shaped life-saver symbol.

GBG will then run the recovery logic. It will automatically go to the Run tab and show a window with a few instructions (Figure 56). This window is non-modal, but best practice is to close it before proceeding. The integration will be in a started, but paused, state. On the left, observe the integration overview (the images of the instruments). You will see an overlay showing where the system expects plates to be.

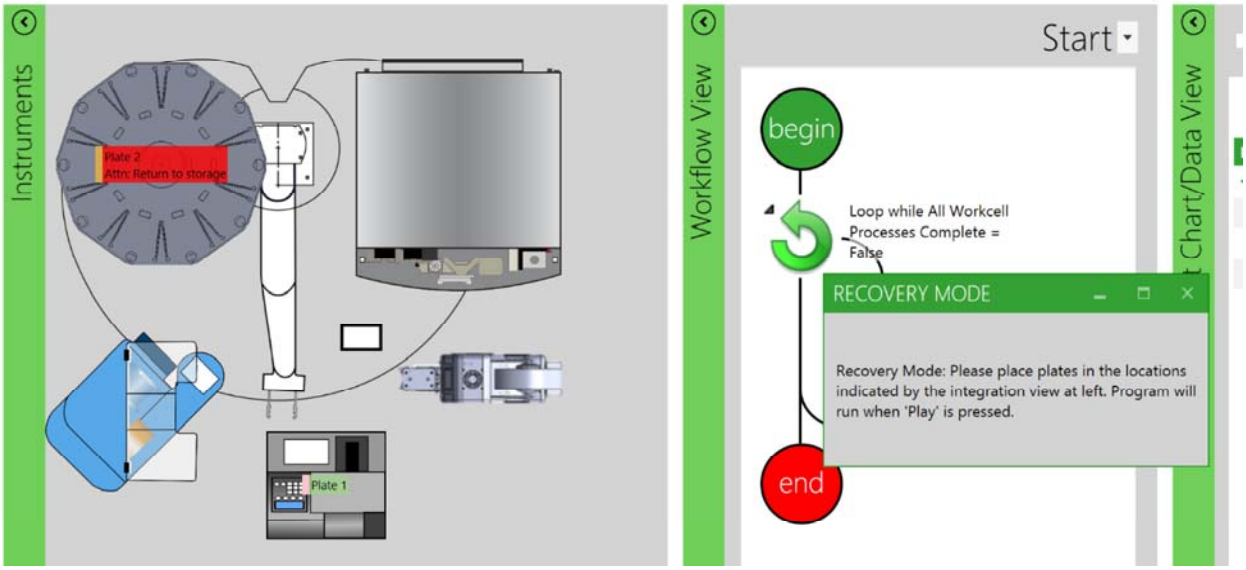


FIGURE 56. RUN TAB: RECOVERY MODE

The overlay showing plate locations is color-coded in two respects: The small colored strips on the left of each flag are based on the same color system used throughout the program for instruments. In other words, the carousel in the above image is set to orange and the ELx405 is set to a peach color, and those colors are representative of their respective instruments. This ensures that the user can identify to which instrument a flag belongs. The main part of the flag is colored green by default and is set to red if extra user interaction is required. The barcode of a plate is shown in front of the colored flag.

For all plates, a user should make sure that the plate is in the location shown. For single nest instruments, the flag will be centered over the appropriate instrument. For multiplate instruments, the flag is still centered, but additional information will show inside the flag to clarify on which nest to place the barcode(s).

Most plates on the deck during a recovery will be labeled with a green flag, meaning that no user interaction is required aside from ensuring that the plate's location is as shown in the flag overlay. For these plates, the system has determined all necessary information and set all data tables appropriately.

While most plates should have a green flag, there will often be one or more red-flagged plates which require some additional interaction by the user. The following paragraphs explain why this happens and what the required steps are.

Understanding the Recovery System

The system is currently configured to use the runtime data tables that track plate movements as well as the table that tracks variable states. Even with this information, it is impossible for the system to know for certain whether or not some tasks completed before the program halted. While further optimizations are being developed, current recovery requires that all plates be considered "completed" on the process that was last started. In other words, Plate 1 in the example above should be on the ELx405 and the user should ensure that the ELx405 protocol already ran.

This concept must be understood for optimal use of the recovery system. The Run Scheduler command is at the heart of the GBG software (see Chapter 5. Run Scheduler Command). This command looks for the next process to run and cues up everything necessary to run that process. An integral part of this is that it waits for processes to report complete. When one of these processes is stopped in the middle, the recovery mechanism is typically able to figure out approximately where the process was stopped and can even alter data tables accordingly to help get the system back to a known and stable state.

However, if a plate has just been moved to the third instrument in the method and is known to have not started yet, it will be sent back to the second instrument because that is where the scheduler expects to find the plate before cueing up all necessary commands for the third instrument's process. Since those do need to be re-cued after the crash, the plate must be placed on the second instrument. Therefore when you look at the flag overlay during a recovery, and it shows where plates are expected, *you may need to move plates in order to match that expectation even if all flags are green*. You should NOT need to adjust data tables for those plates UNLESS the flag shows red.

User Interaction and Red-Flagged Plates

Because plates must start from certain known states, plates that have just come out of storage will be sent back into storage. Such plates will be shown with a red flag. If the plate has been added to the runtime data table already, the recovery system will erase that reference so that it is as though the plate had never been removed from storage.

However, the user is responsible for ensuring that the contents data tables are correct. For stacks, this may require that the stack count be incremented. For hotels, simply ensure that the barcode shows on the correct shelf. As previously explained, the reason for this is that the system does not attempt to determine whether the program halted before or after the contents were updated. That step is driver dependent, which makes it simpler and more reliable for the user to manually check this step.

A red flag will also be used for plates on a multiplate process/multinest instrument. The first step required here is to ensure that the plates “ran” just as the user does for other, green-labeled, plates. The user should also verify any tables used by these processes. Specifically, worklists should correctly show whether or not a task was completed, and barcode queues should be verified. The exact edit needed depends, again, on exactly where the halt occurred.

The last type of red flag will show when a barcode is listed for multiple processes/instruments and it cannot be determined why. In this case, it is up to the user’s discretion to resolve the conflict.

Finally, inspect the deck to ensure that all plates are accounted for. There should be no plates on the deck unaccounted for; any plate in the flag overlay must have a corresponding plate on the deck (or returned to storage) and likewise, any plate on the deck must show in the overlay.

Once the plates have been placed and the tables verified, the user should click **Play** and the run will proceed from this known state. Because of the complexities involved in starting a run in the middle, it is highly recommended that a user watches or periodically checks on the integration for a few minutes after a recovery. Incorrect recovery procedures will often show up as a deadlock, and it is important that the user be able to identify which plates failed to move.

Appendix A: Auto-Generated Components

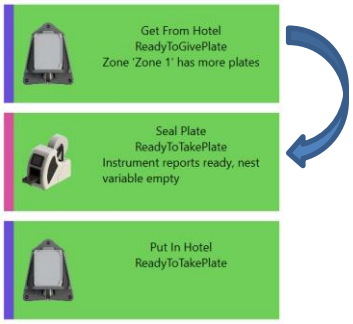
Component	Added When…
Teach Points	Instrument Added
Get and Put Procedures	Instrument Added
Sequences	Instrument Added
Other Procedures (i.e., Wasp.Seal.Start)	Processes Compiled
Nest Variables	Instrument Added
Other Variables	Processes Compiled
Storage Teach points	Button in the Storage Configuration Window is clicked
Edit Content Process	Added to Start Process when a Get From Hotel or Stack is added.
Data Tables	Process Compiled or Run (i.e., Get From Hotel with a strategy of From Worklist will generate the Barcode Queue table at runtime).

Appendix B: Instrument Process Status and Ready Evaluations

Instrument Process Status Explained

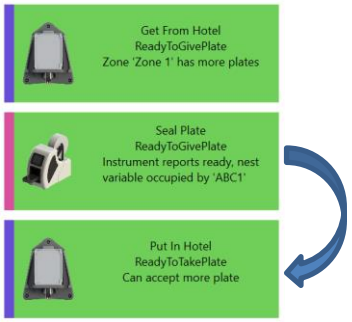
Instrument process status is constantly evaluated by the scheduler in order to determine if a plate is ready to be moved from one instrument process to the next. The most important status evaluation results are ReadyToGivePlate, ReadyToTakePlate, and Busy. A plate can only be moved from one instrument process to the next if the status evaluation of the first is ReadyToGivePlate and the status evaluation of the next is ReadyToTakePlate.

Storage to Sealer Transition



Sealer is Operating

Sealer to Storage Transition



Basic process status evaluation for instruments is usually based on whether a plate is in the nest and the status of the instrument. If the instrument is operating and there is a plate in the nest, then the status evaluates to Busy. If the instrument is idle and there is a plate in the nest, then the status evaluates to ReadyToGivePlate since the instrument has just completed its task. If there is no plate in the nest and the instrument it idle, then the status evaluates to ReadyToTakePlate.

Ready Evaluation

Instrument process evaluation can be overridden using the Ready Evaluation feature on an instrument process. A variable can be used to set the evaluation, and that variable can be set during the program workflow or in a script which is run prior to each evaluation.

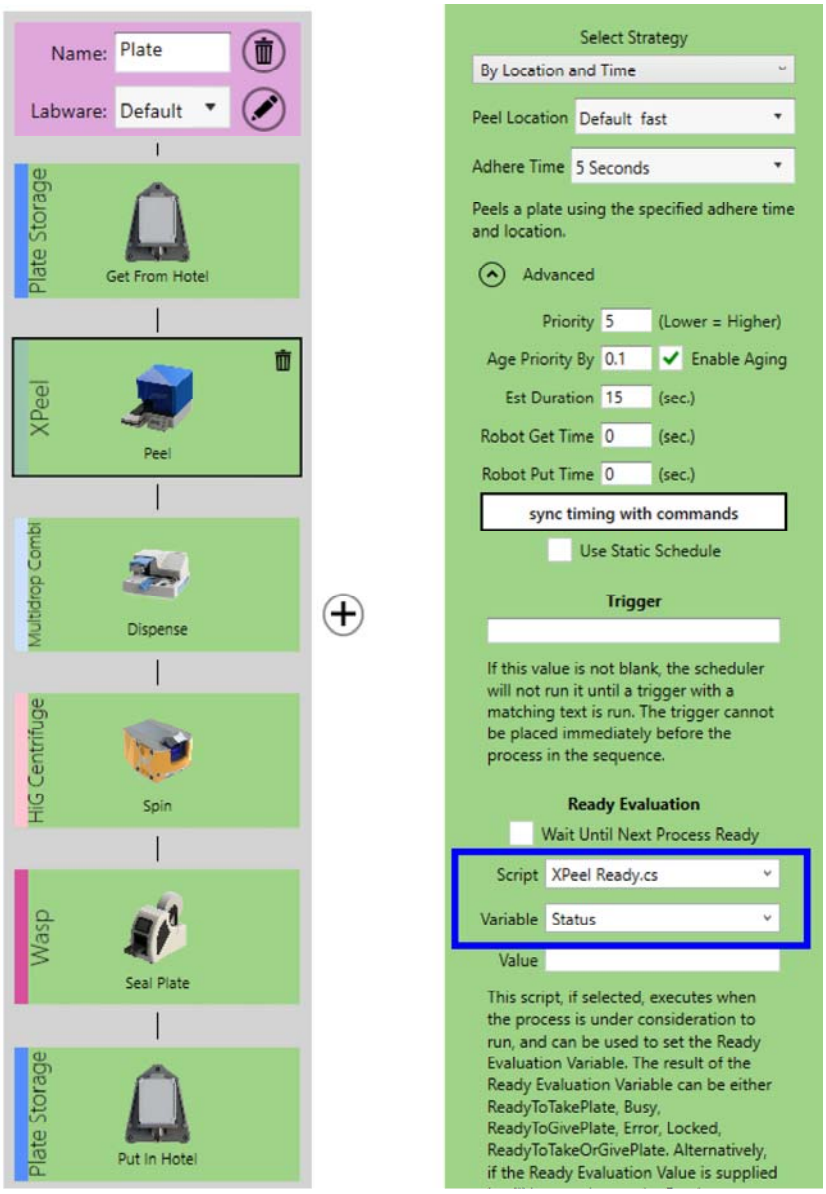


FIGURE 57. READY EVALUATION FEATURE

Note that the **Value** field is not set in the Ready Evaluation. The **Value** field can be used to compare binary conditions for ReadyToTakePlate or ReadyToGivePlate vs Busy.

Example Script

The following script ensures that all the instruments after the Xpeel are free of plates before allowing a plate onto the Xpeel.

```
using System;
using System.IO;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Linq;
```

```
using BioSero.GreenButtonGo.Scripting;

namespace GreenButtonGo.Scripting
{
   public class Xpeel_Ready : BioSero.GreenButtonGo.GBGScript
   {

      public void Run(Dictionary<String, Object> variables, RuntimeInfo runtimeInfo)
      {
         // this script will only allow a plate onto the XPeel if the deck is clear (Xpeel,Combi, HiG, Sealer)

         string plateOnXpeel = variables["Plate On XPeel"] as string;
         string plateOnCombi= variables["Plate On Multidrop Combi"] as string;
         string plateOnHiG = variables["Plate On HiG Bucket 1"] as string;
         string plateOnSealer = variables["Plate On Wasp"] as string;

         bool deckIsClear
= string.IsNullOrWhitespace(plateOnXpeel) && string.IsNullOrWhitespace(plateOnCombi) && string.IsN
ullOrWhitespace(plateOnHiG) && string.IsNullOrWhitespace(plateOnSealer);

         if (deckIsClear)
         {
            variables["Status"] = "ReadyToTakePlate";
         }
         else
         {
            if (!string.IsNullOrEmpty(plateOnXpeel))
            {
               variables["Status"] = "ReadyToGivePlate";
            }
            else
            {
               variables["Status"] = "Busy";
            }
         {

      }
   }
}
```

Appendix C: Referencing External DLLs in GBG Scripts

Introduction

It is sometimes beneficial to consolidate common code in an external DLL and to call methods therein from within GBG scripts. This can be accomplished by including reference lines at the top of the script that point the compiler to the location of the DLL.

Example

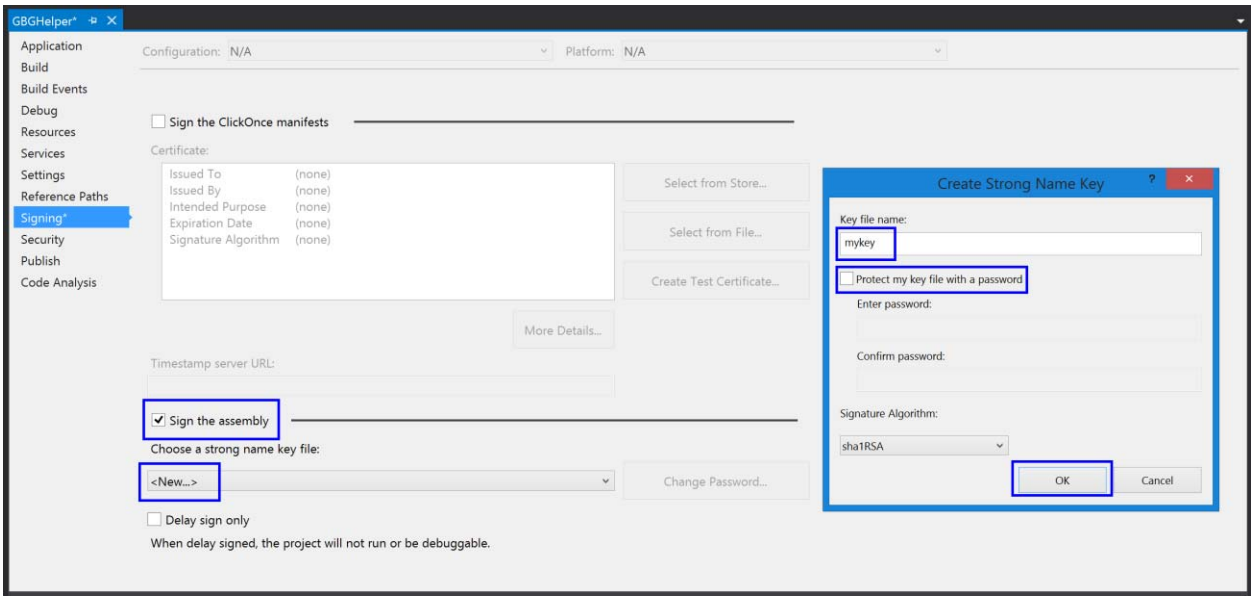
```
reference C:\Program Files (x86)\Green Button Go\GBGHelper.dll;
using System;
using System.IO;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Linq;
using BioSero.GreenButtonGo.Scripting;
using GBGHelper;

namespace GreenButtonGo.Scripting
{
   public class Script_1 : BioSero.GreenButtonGo.GBGScript
   {
      public void Run(Dictionary<String, Object> variables, RuntimeInfo runtimeInfo)
      {
         LimsPortal portal = new LimsPortal();
         portal.Open();
         string transferLog = variables["ATS-100 Log File"] as string;
         portal.LogTransferResults(transferLog);
      }
   }
}
```

Name Resolution

There are sometimes issues getting .NET to resolve the full name of the DLL. Security issues may prevent execution of code from a DLL not in the GBG program files folder unless it is strongly named and registered with the Global Assembly Cache (GAC) . This will result in an error message like the following:

If you experience issues with this it may be necessary to give your assembly a strong name. This can be easily achieved with a few clicks in Visual Studio. Right-click on the Project file in Solution Explorer, then select **Properties** and enable signing of the assembly as depicted below.



To register it with the GAC, use the Visual Studio Command Prompt and type the following command:

```
gacutil /i yourdll.dll
```

Appendix D: Biosero Universal Storage Guide

Introduction: What is the Universal Storage Driver (USD)?

The Need

The universal storage driver (USD) is a core component in standardizing Green Button Go (GBG) storage. Past storage drivers (including ambient static storage, carousels, incubators, plate loaders etc) have offered similar processes and tools, but these were not standardized with the same strategies, properties, and commands.

The Solution

To resolve this, the universal storage driver (USD) has been created to collate all previous storage options.

- This driver provides GBG with static, ambient storage options, ie hotels and stacks.
- To enable other storage such as a Liconic STX-44, the driver for this additional storage must be present, but all new or updated storage drivers will use the universal storage driver as a dependency meaning that the universal storage driver should be present as a driver even if static storage is not used.
- The processes and commands made available through the USD are consistent wherever possible with those provided by other updated storage drivers.

Processes

Process List

All storage that implements the USD improvements presents a standardized set of processes. These include:

1. Get From Hotel
2. Put In Hotel
3. Get From Stack
4. Put In Stack
5. Incubate
6. Hold
7. Edit Contents

Exceptions to the Process List

Note that an instrument such as an STX-44 will have a single “Get” process since stacks are not an option. These types of physical differences are the one type of deviation between different storage drivers.

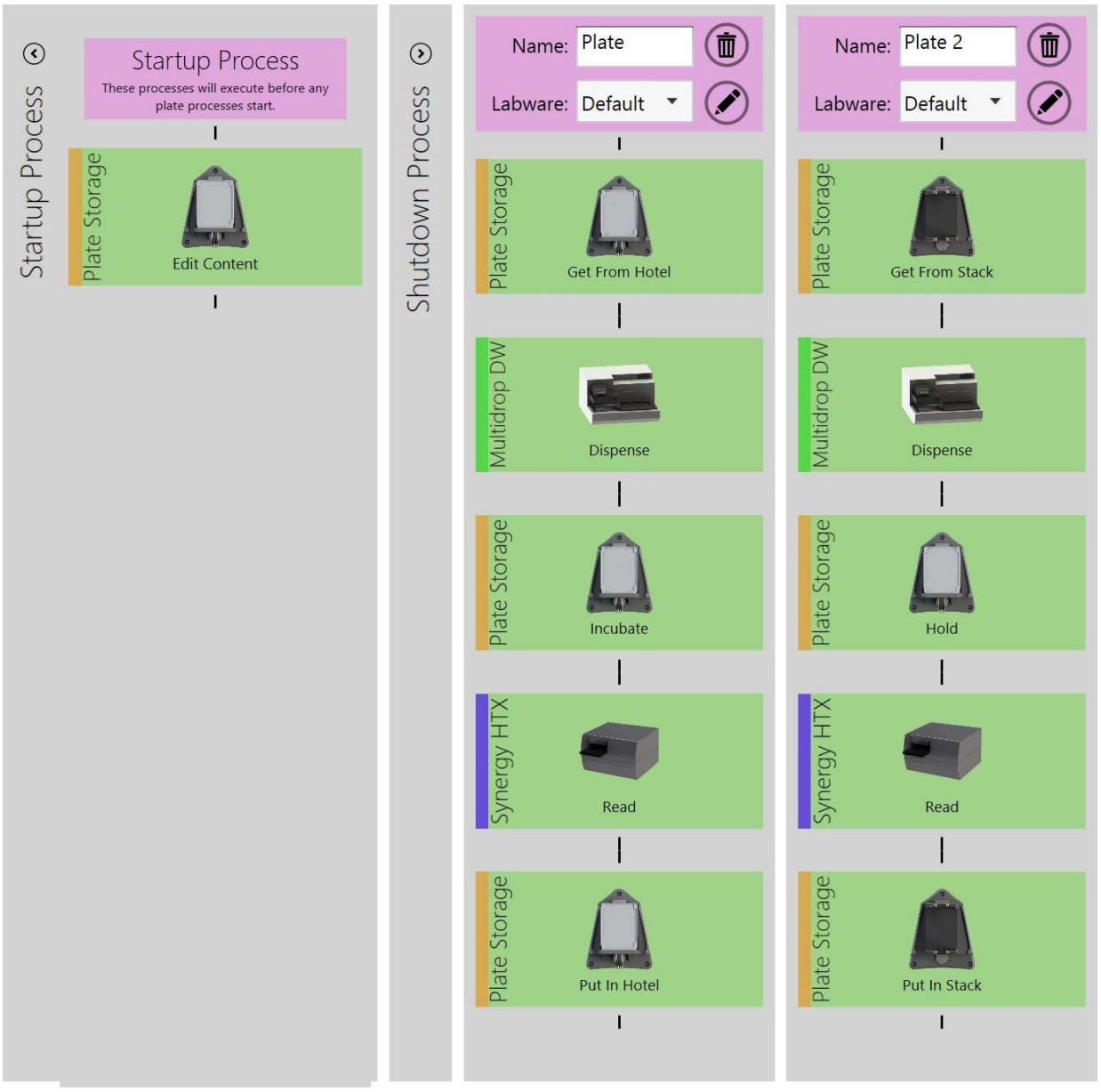
Standard Use

The image below shows a typical use case scenario for all of the USD processes. Details follow.

Basic Rules

- A plate process (column or swim lane) should start with a storage “Get” process
- A plate process should end with a “Put” process

- Incubate and hold processes should be placed in the middle of a run. Multiple incubate/hold processes are allowed in one plate process



Specifics

1. Get From Hotel
    a. Pulls plates from random access storage
    b. Only place this process at the start of a Plate Process
    c. Strategies
        i. From Zone
            1. Zone : The name of the zone from which to get plates
            2. Input Throttle (min): Delay between each plate

3. Require All Plates: Modifies the Finished evaluation. True means the process will not report finished until the zone is empty or all plates are marked as having run

ii. From Worklist

1. Table Name: The name of the table in the database that this process looks at for its list of plates to run
2. End If Empty: Modifies the Finished evaluation. False means the process will never report finished and an End Program command should be used somewhere in the program.
3. Batch Variable: If set, the value of the specified variable must match the Batch value for desired plates in the queue. This allows plates from different batches to be enqueued.
4. No Plate Strategy: Defines what happens when the desired plate is not present or not at the front of the queue.
    a. Wait: Wait for the plate to be present
    b. Error: Throw error if plate is not present
    c. Reshuffle: Moves top plate to end of queue until top plate is available
    d. Next Available: Provides first available plate in queue without reordering queue

2. Put In Hotel
    a. Places plates in random access storage
    b. Only place this process at the end of a Plate Process. This is usually but not always used to return plates to the same storage location where they started
    c. Strategies
        i. Put In Hotel
            1. Zone : The name of the zone from which to get plates
            2. Prefer Original Location: If true will try to select the shelf where the plate started marked by the Previous Barcode value. If False or if the shelf is unavailable the plate will go on the first available shelf.
3. Get From Stack
    a. Pulls plates from LIFO storage
    b. Only place this process at the start of a Plate Process
    c. Strategies
        i. Sequentially
            1. Zone : The name of the zone from which to get plates
            2. Input Throttle (min): Delay between each plate
            3. Require All Plates: Modifies the Finished evaluation. True means the process will not report finished until the zone is empty or all plates are marked as having run
4. Put In Stack
    a. Puts plates into LIFO storage
    b. Only place this process at the end of a Plate Process

c. Strategies
    i. Sequentially
        1. Zone : The name of the zone from which to get plates

5. Incubate
    a. Holds each plate for specified period of time before releasing to the next process
    b. Only place this process within a plate process, not at the start or end
    c. Incubation temperature depends on the hardware; for the USD it is ambient
    d. To get low variance on actual incubation times set process priorities to evaluate the incubation first
    e. Strategies
        i. Wherever
            1. Time (min): The amount of time the plate will be held in incubation before being considered ready to move on
            2. Zone: The name of the zone in storage where the incubation will take place. This zone must be present in the storage random access hotels
            3. Acceptable Deviation (min): While the "Time (min)" value determines the time at which the Incubate process reports the plate as ready, the actual move time depends as well on the following process and its status. The Acceptable Deviation (min) therefore represents the time window that GBG has to move the plate forward. If this is not accomplished the plate will automatically be marked as canceled and will not continue the run.
            4. Cancel Deviant Plates: If false then the Acceptable Deviation (min) will be ignored and no plates will be canceled

6. Hold
    a. Holds plates until a specified condition is met
    b. Only place this process within a plate process, not at the start or end
    c. Used for Batch processing
    d. Strategies
        i. Until Variable = True
            1. Zone: The name of the zone in storage where the hold will take place. This zone must be present in the storage random access hotels
            2. Trigger Variable: The name of the variable to watch. When the variable is set to false, plates will be held. When it is set to true, they will be released
        ii. Wait For Time
            1. Zone: The name of the zone in storage where the hold will take place. This zone must be present in the storage random access hotels
            2. Time: The time at which plates will be released to move forward in the process

7. Edit Content
    a. Allows the user to edit the storage data

b. This process provides input options defined below, but storage can also be edited in a number of other ways such as scripting or direct editing of the tables through the GBG Advanced Tab.
c. Strategies
    i. Launch Content Editor (See image and description below)
    ii. Load From File
        1. Hotel File Name: Loads the designated file into the hotels table
        2. Stack File Name: Loads the designated file into the stacks table

Storage Table Import/Export

Hotel	Shelf	Barcode	Previous_B	Incubation	Incubation	Zone	Labware	Has_Been_Run	Is_Lidded	Is_Sealed	Placed_By_Process	Hotel_Physical_Position
Hotel 1	1	Source 1	Plate 1		0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	2	Source 2	Plate 2		0:00:00	Sources	Default	FALSE	FALSE	FALSE		1
Hotel 1	3	Source 3	Plate 3		0:00:00	Sources	Default	FALSE	FALSE	FALSE		1
Hotel 1	4	Source 4			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	5	Source 5			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	6	Source 6			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	7	Source 7			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	8	Source 8			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	9	Source 9			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	10	Source 10			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	11	Source 11			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	12	Source 12			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	13	Source 13			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	14	Source 14			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	15	Source 15			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	16	Source 16			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	17	Source 17			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	18	Source 18			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	19	Source 19			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	20	Source 20			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	21	Source 21			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 1	22	Source 22			0:00:00	Sources		FALSE	FALSE	FALSE		1
Hotel 2	1	Destination 1			0:00:00	Destinations		FALSE	FALSE	FALSE		2
Hotel 2	2	Destination 2			0:00:00	Destinations		FALSE	FALSE	FALSE		2
Hotel 2	3	Destination 3			0:00:00	Destinations		FALSE	FALSE	FALSE		2
Hotel 2	4	Destination 4			0:00:00	Destinations		FALSE	FALSE	FALSE		2

The above csv file is an export of a storage hotel table. This file (or a subset of its columns) can also be used as an import in the Edit Content process' Load From File strategy.

Content Editor

The Content Editor shown above will appear if Edit Content has the Launch Content Editor strategy selected. In this example four hotels are present and the labware specific options are expanded on the right side of the window. Shelves can be selected (multiple or in singles), and various properties can be set including the barcode, zone, labware, lidded status, and sealed status. Switching to the Advanced tab here allows the backing data table to be edited as a classic grid.

Commands

Processes are high level representations of extensive low level commands. These auto-generated commands exist in procedures in the GBG Advanced Tab. The image below shows the procedures generated for each process. Note that the method depicted matches the one shown above that demonstrates each of the seven storage processes.

Procedure Conventions and Exceptions

The normal procedure convention, as seen in the image below, is to have auto generated “Get Plate From Nest” and “Put Plate On Nest” procedures. These allow the arm to get a plate or place on with minimum setup work on the part of the integrator. It’s very straightforward with single nest instruments. Storage, however, is a departure from this framework. Storage typically includes dozens of nests or stack positions. Storage often requires a plate to be moved out of storage before it can be grabbed or necessitates the shelf/position information to be output by the storage before the arm can get/put the plate. Below the following image you will find a short chart of each storage procedure and what it does.

Get From Hotel/Stack OR Get Plate (Incubator)

1. Present
    a. Either delivers plate to storage nest (ie on an incubator) or outputs the plate location into variables so the arm can access the plate. In the latter case will include arm movement as well
2. Get
    a. Physically gets the plate (ie from an incubator)
    b. Exception: if the plate was physically grabbed in the “Present” step then this procedure will be blank or deleted

Put In Hotel/Stack OR Put Plate (Incubator)

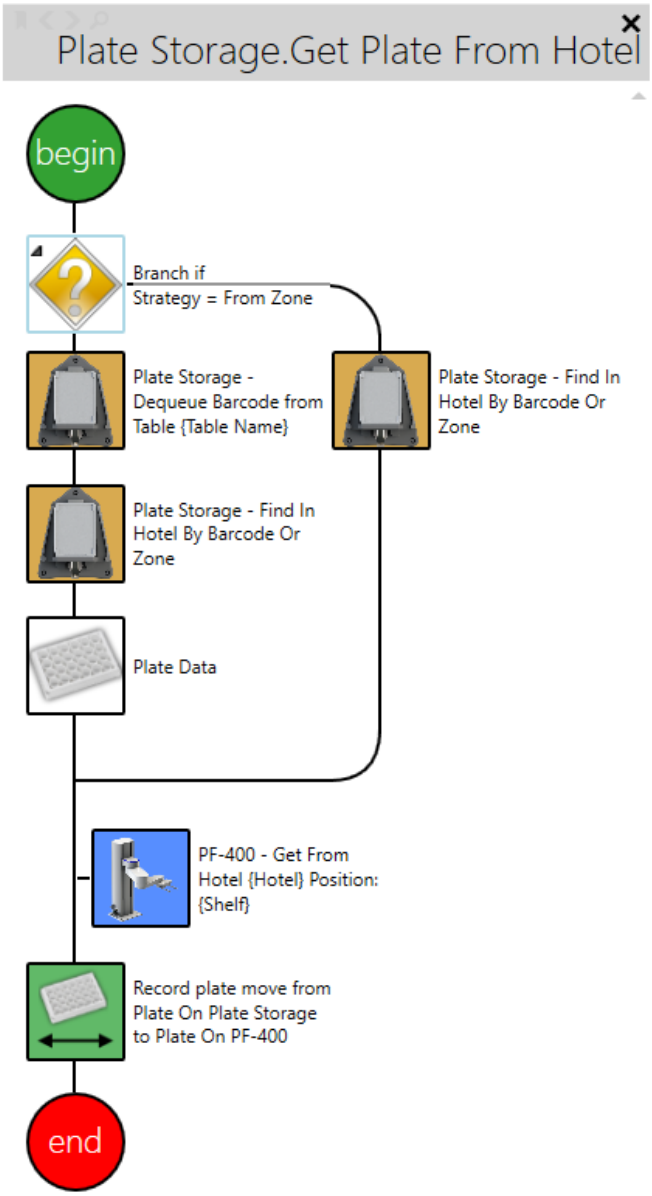
1. Put
    a. Physically places the plate on a nest (ie on an incubator)
    b. Exception: if the plate will be physically moved in the “Start” step then this will be blank or deleted
2. Start
    a. Either puts plate away (incubator) or outputs the plate location into variables and calls the arm move that puts the plate away

Incubate/Hold

The incubate and hold processes use a similar setup to the Get and Put defined above whether it's on an incubator or static storage.

1. When a plate goes into incubation the system will call the Put and Start procedures. When it comes from incubation the system will call the Present and Get procedures

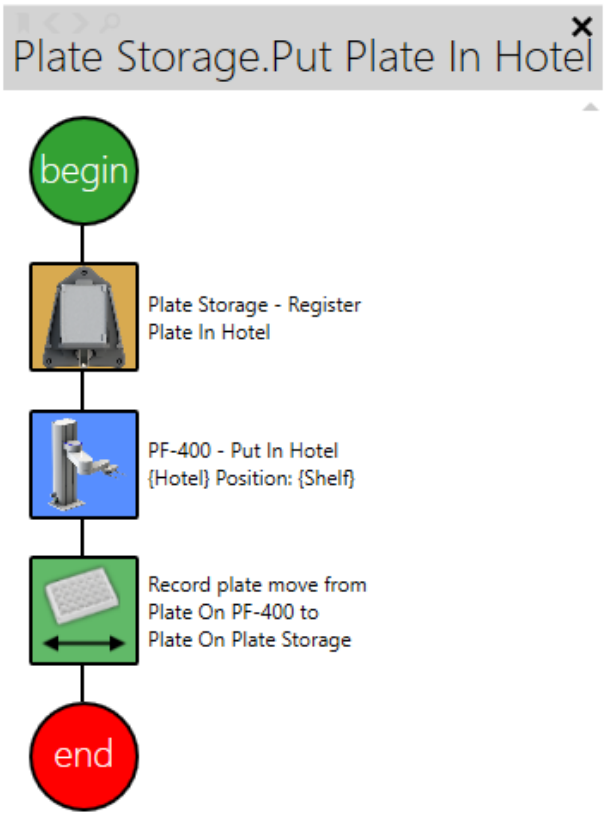
Get From Hotel



This procedure implements the logic from the storage point of view that's necessary to present the appropriate plate. In an incubator, this procedure would physically move the plate to the nest. Here in the USD it tells the arm where the plate is. Note that for the USD the auto generated code for this includes the arm grabbing the plate, so the procedure "Get Plate From Plate Storage" will be empty or removed.

1. A branch determines whether the strategy dictates getting from a zone or from a worklist
2. If a worklist is used then the barcode is found by the “Dequeue…” command. The “Find…” command registers the plate being removed. The Plate Data command writes some of the plate’s extra data into the runtime data table. Note that this extra info comes from the worklist and is unavailable if the plate comes sequentially from a zone.
3. The arm physically gets the plate
4. The “Plate On…” variables are set by the “Record Plate Move…” command

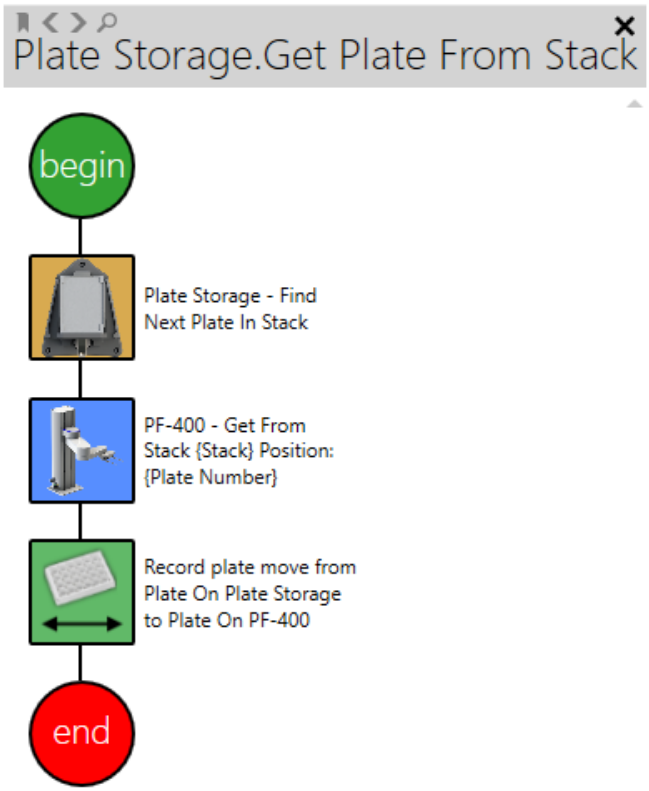
Put In Hotel



This procedure implements the logic from the storage point of view that’s necessary to receive the appropriate plate. In an incubator, this procedure would physically move the plate into storage from the nest. Here in the USD it tells the arm where to put the plate. Please note “Procedure Conventions and Exceptions” above regarding unique storage conventions.

1. With only one strategy this is a little simpler than the get procedure
2. A storage command registers the plate being put away and outputs the correct location
3. The arm physically gets the plate
4. The “Plate On…” variables are set by the “Record Plate Move…” command

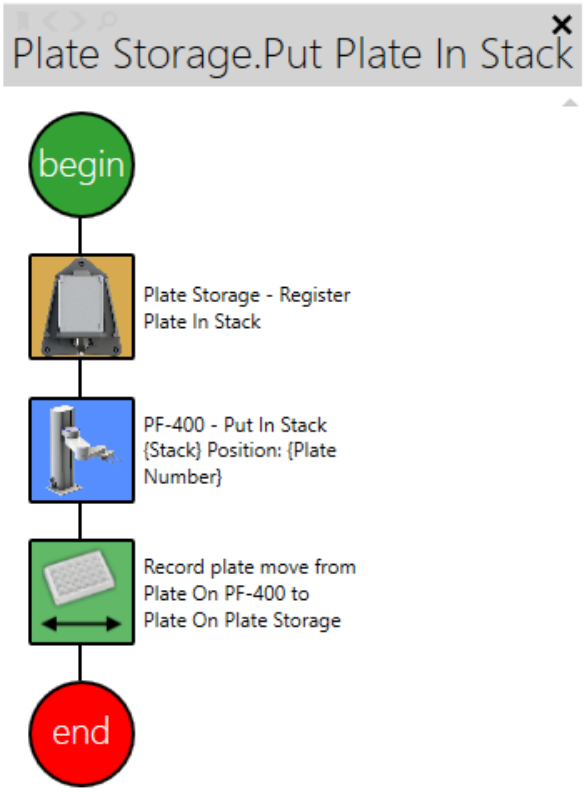
Get From Stack



This procedure implements the logic from the storage point of view that's necessary to present the appropriate plate. Please note "Procedure Conventions and Exceptions" above regarding unique storage conventions.

1. A storage command registers the plate being grabbed and outputs the correct location
2. The arm physically gets the plate
3. The "Plate On…" variables are set by the "Record Plate Move…" command

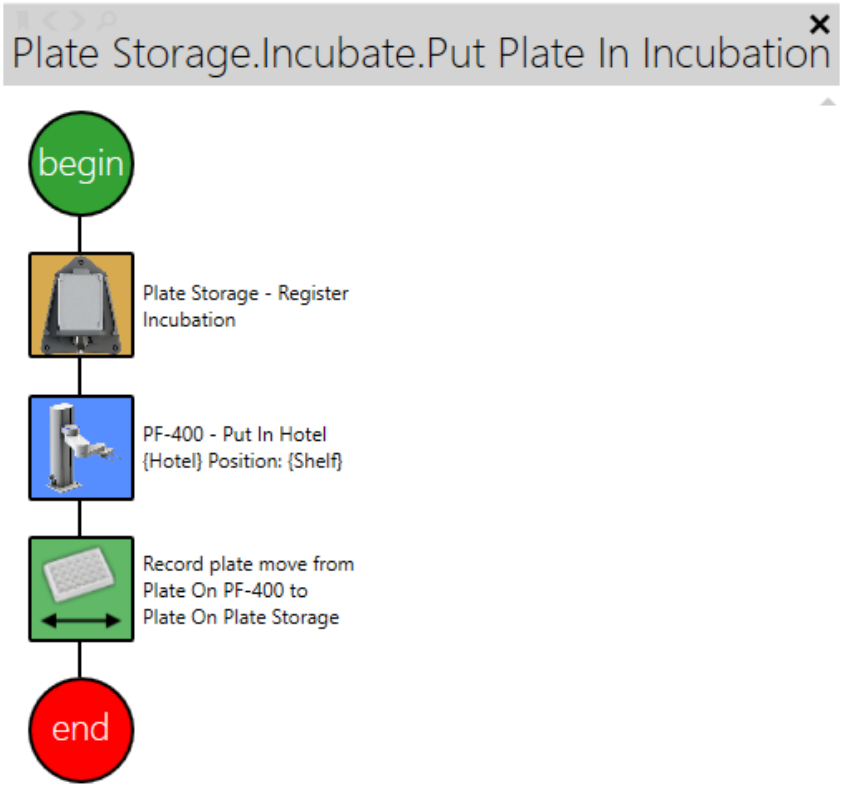
Put In Stack

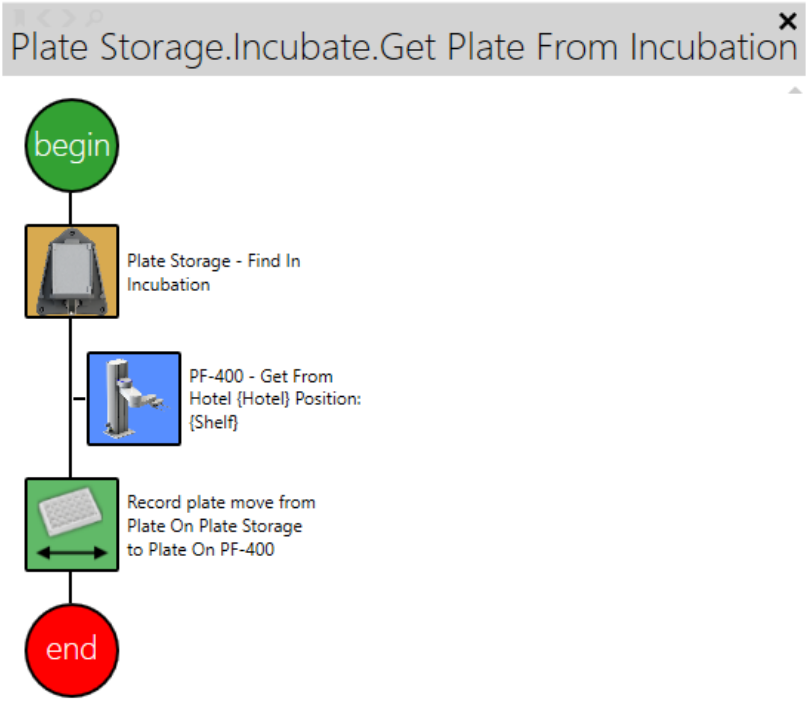


This procedure implements the logic from the storage point of view that's necessary to allow the arm to get the appropriate plate from the stacks. Please note "Procedure Conventions and Exceptions" above regarding unique storage conventions.

1. A storage command registers the plate being put away and outputs the correct location
2. The arm physically gets the plate
3. The "Plate On…" variables are set by the "Record Plate Move…" command

Incubate OR Hold



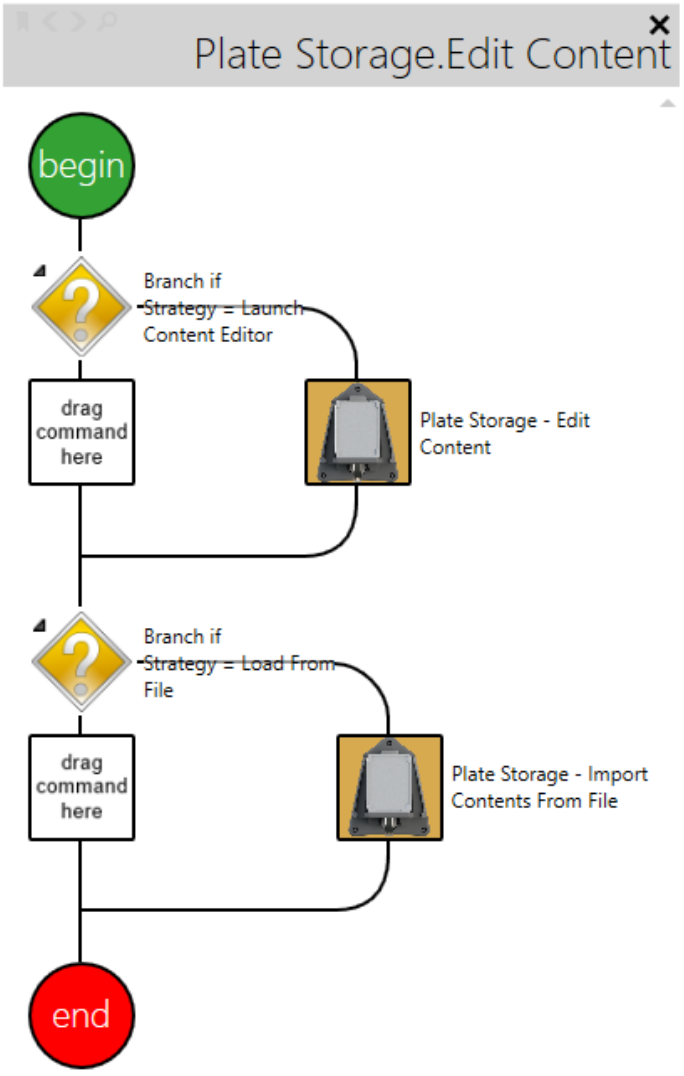


Incubate uses the same basic logic as put in hotel and get from hotel, except that a worklist is not an option for incubations. This simplifies things slightly, so both the Put Plate In Incubation and Get Plate

From Incubation employ three commands. The Hold process is roughly identical in its procedures and commands.

1. A storage command registers the plate being grabbed or put away and outputs the correct location
2. The arm physically gets or puts the plate
3. The “Plate On…” variables are set by the “Record Plate Move…” command

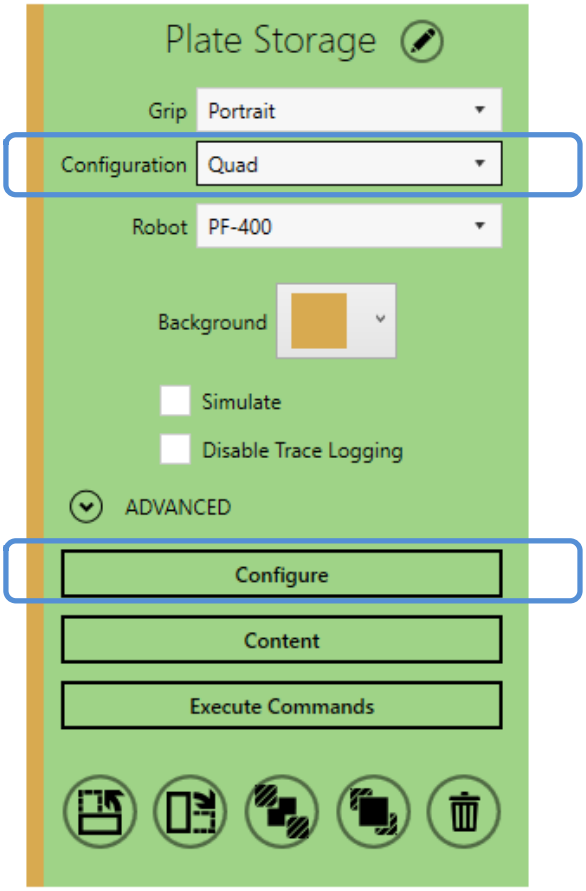
Edit Content



Edit Content is slightly different in that it doesn’t move plates. It has no get or put logic, but it does have two strategies available. The content editor and input files are covered earlier in this document.

Changing Configurations

Storage units often come in multiple configurations. This section describes the tools that allow an integrator to define the physical characteristics of storage on a system.



The instrument properties shown above can be accessed in the Layout Tab. The Configuration drop down allows the graphic to be changed for clarity. Note that this doesn't change the physical properties of the storage as perceived by the system. These must be set up using the Configure button indicated above.

Storage Configuration Window

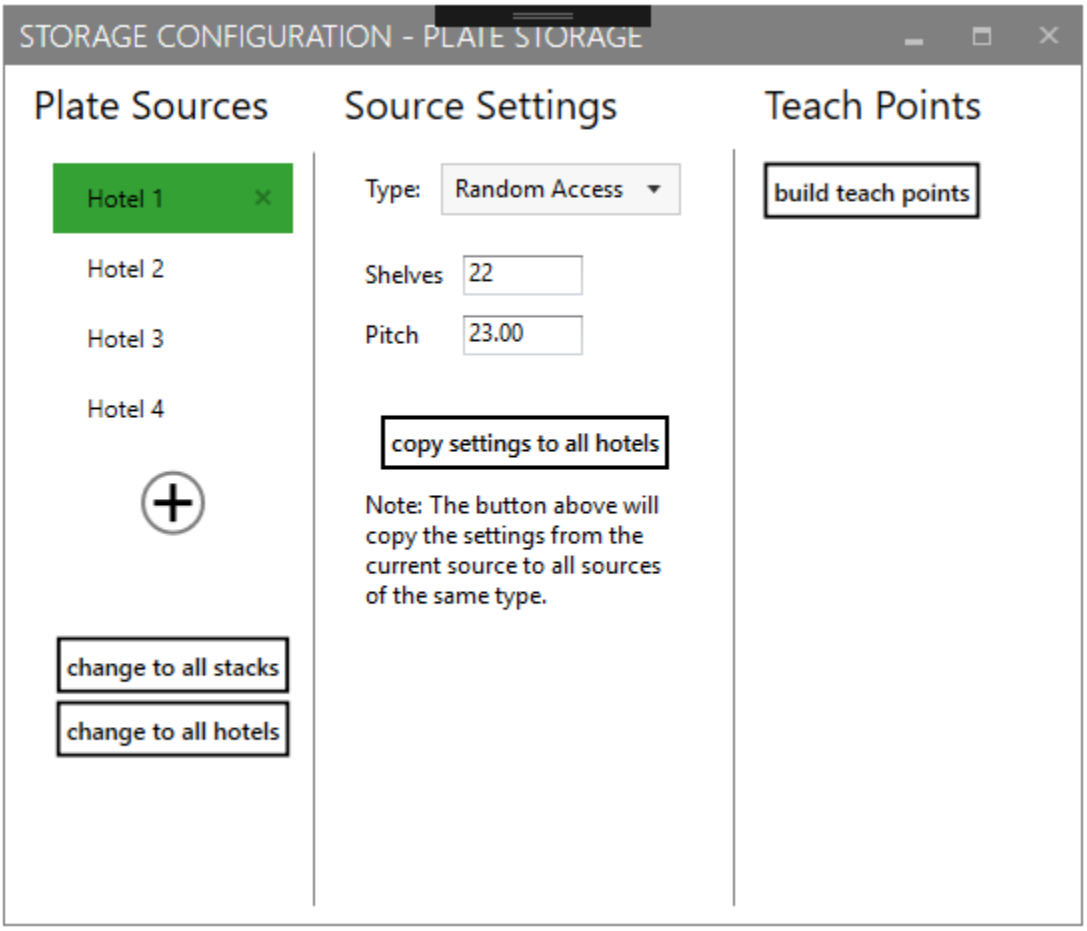


Plate Sources

These are the stacks and hotels present in the storage. For a 10 stack carousel, for instance, one would add six more plate sources and hit the “change to all stacks” button.

Source Settings

These vary slightly between hotels and shelves. Define the physical characteristics so that the system knows how many plates will fit.

Teach Points

Click “Build Teach Points” to generate the necessary teach points (name them) for newly added storage.

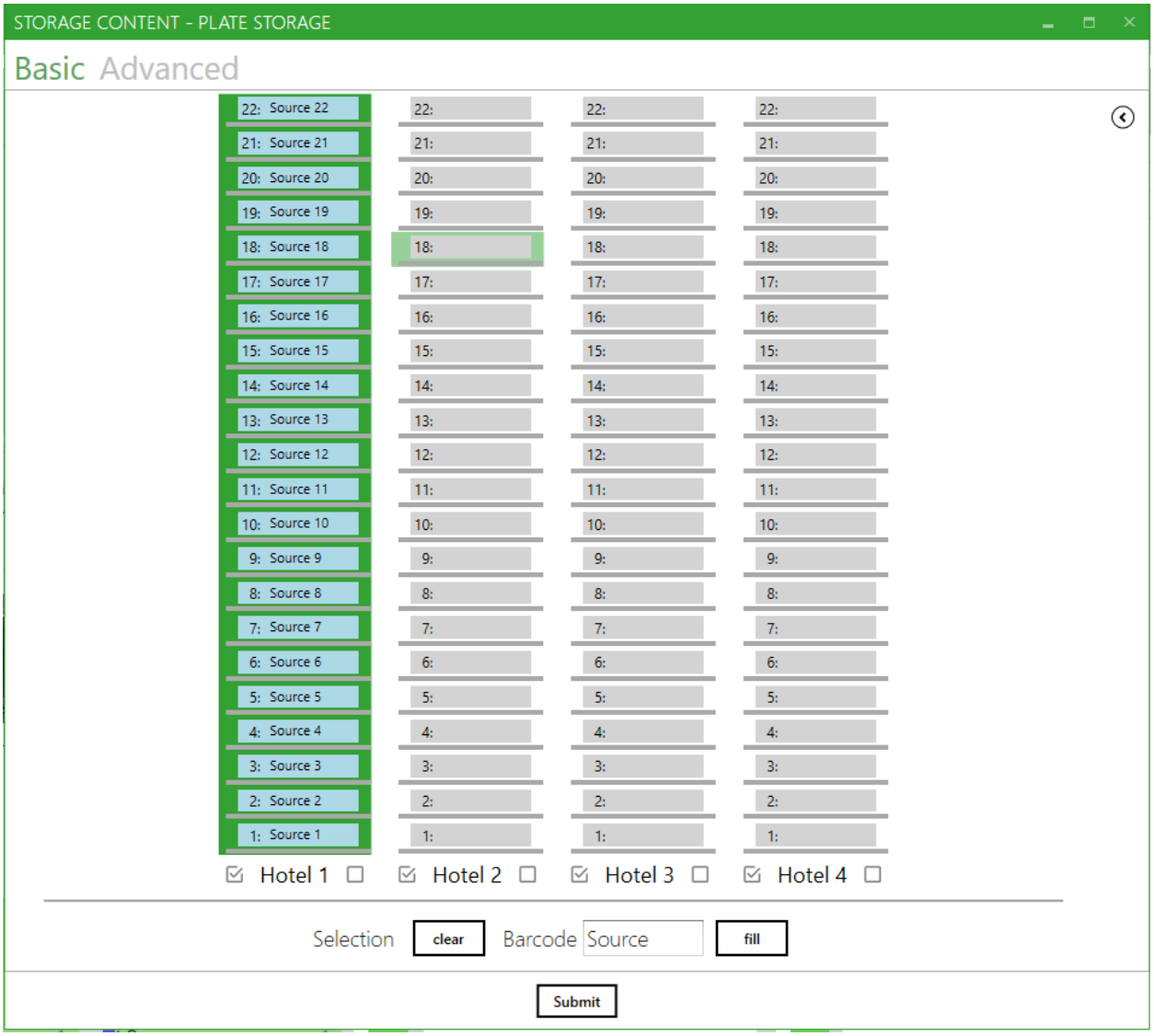
Setting Contents

Green Button Go uses an explicit storage architecture that allows for maximum control over where plates are stored in a system. At its root, this concept means defining zones where plates come from and are returned to and then telling the system which plates are there. There are many options for populating these values, including having an arm scan the barcodes present or count plates in a stack. This document covers the default manual input window. For information on additional options consult the Biosero wiki or a Biosero representative.

Content Editor

The Content Editor window can be access from the storage instrument's properties in the Layout, Run, or Advanced tabs.

Basic View Tab



The Basic view allows a user to quickly input the basic data to start a run.

Default Controls

1. Clear: Clears all selected shelves
2. Fill: Fills all selected shelves with a sequence of plates using the provided text as the seed for the barcodes

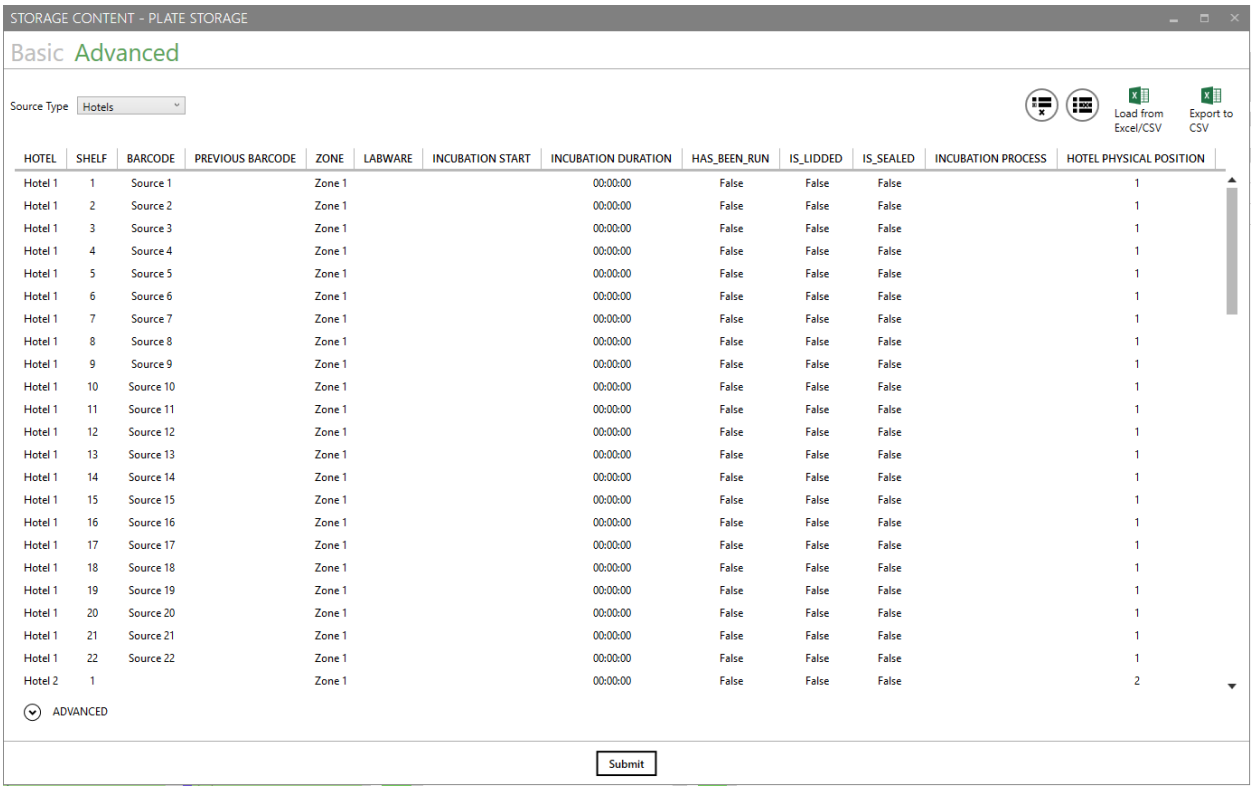
Advanced Controls

The arrow at upper right expands the advanced control panel.

1. Zone: define a new zone name for the selected shelves or selected stack. A stack cannot be split into multiple zones; a stack must be uniform in regards to zones
2. Labware: Identify the labware on selected shelves or stacks. If this is set its value overrides the labware value provided for a plate process. This value is optional
3. Lidded: Identify whether or not selected shelves or stacks have lidded plates
4. Lidded: Identify whether or not selected shelves or stacks have sealed plates

Advanced View Tab

The Advanced Tab View lets the user set the same properties as the Basic View Tab in a classic datatable format. It also exposes a number of other storage properties that are typically used by GBG without any user interaction.

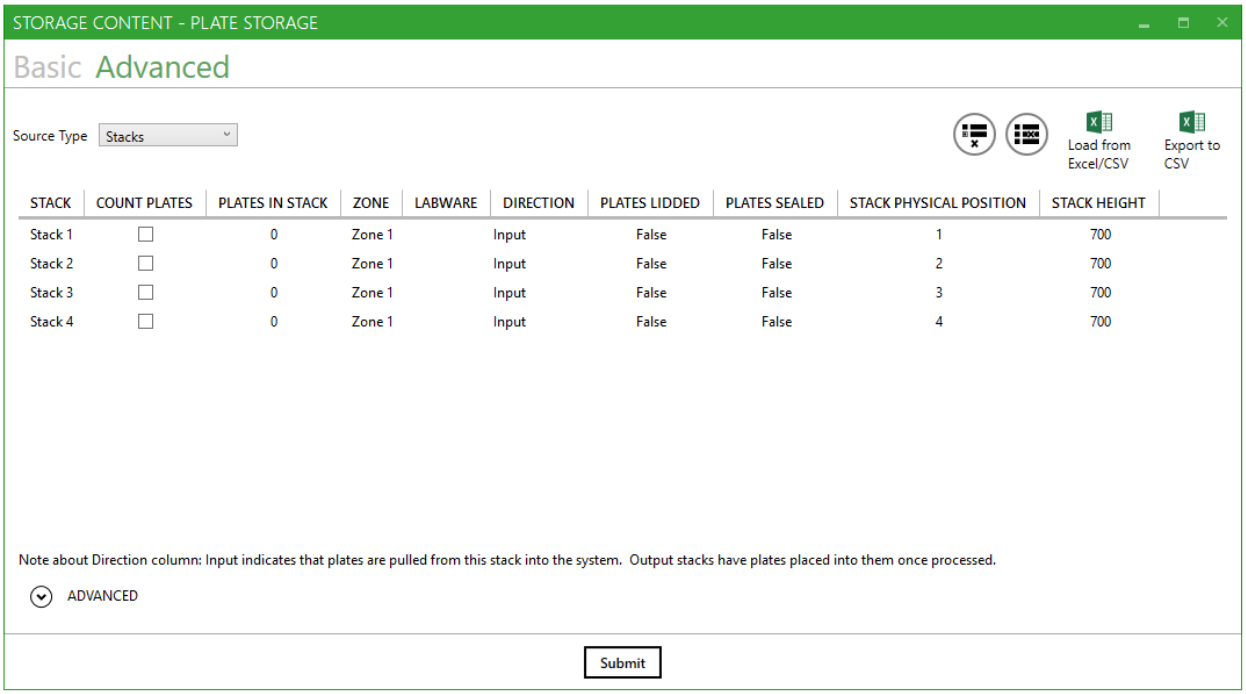


HOTEL	SHELF	BARCODE	PREVIOUS BARCODE	ZONE	LABWARE	INCUBATION START	INCUBATION DURATION	HAS_BEEN_RUN	IS_LIDDED	IS_SEALED	INCUBATION PROCESS	HOTEL PHYSICAL POSITION
Hotel 1	1	Source 1		Zone 1			00:00:00	False	False	False		1
Hotel 1	2	Source 2		Zone 1			00:00:00	False	False	False		1
Hotel 1	3	Source 3		Zone 1			00:00:00	False	False	False		1
Hotel 1	4	Source 4		Zone 1			00:00:00	False	False	False		1
Hotel 1	5	Source 5		Zone 1			00:00:00	False	False	False		1
Hotel 1	6	Source 6		Zone 1			00:00:00	False	False	False		1
Hotel 1	7	Source 7		Zone 1			00:00:00	False	False	False		1
Hotel 1	8	Source 8		Zone 1			00:00:00	False	False	False		1
Hotel 1	9	Source 9		Zone 1			00:00:00	False	False	False		1
Hotel 1	10	Source 10		Zone 1			00:00:00	False	False	False		1
Hotel 1	11	Source 11		Zone 1			00:00:00	False	False	False		1
Hotel 1	12	Source 12		Zone 1			00:00:00	False	False	False		1
Hotel 1	13	Source 13		Zone 1			00:00:00	False	False	False		1
Hotel 1	14	Source 14		Zone 1			00:00:00	False	False	False		1
Hotel 1	15	Source 15		Zone 1			00:00:00	False	False	False		1
Hotel 1	16	Source 16		Zone 1			00:00:00	False	False	False		1
Hotel 1	17	Source 17		Zone 1			00:00:00	False	False	False		1
Hotel 1	18	Source 18		Zone 1			00:00:00	False	False	False		1
Hotel 1	19	Source 19		Zone 1			00:00:00	False	False	False		1
Hotel 1	20	Source 20		Zone 1			00:00:00	False	False	False		1
Hotel 1	21	Source 21		Zone 1			00:00:00	False	False	False		1
Hotel 1	22	Source 22		Zone 1			00:00:00	False	False	False		1
Hotel 2	1			Zone 1			00:00:00	False	False	False		2

1. Hotel: The name of the hotel for a given shelf
2. Shelf: The shelf's number
3. Barcode: The shelf's current barcode
4. Previous Barcode: The shelf's last registered barcode (set when a plate is removed and used to return a plate to its original shelf)
5. Zone: The zone assigned to a shelf. Note that most storage drivers support comma separated zone values meaning that a shelf can have multiple zones
6. Labware: Optional Labware Definition name
7. Incubation Start: Start time if the plate is in incubation
8. Incubation Duration: Timespan if the plate is in incubation

9. Has Been Run: True if the plate was run and has been returned. Very important for sequential get/return
10. Is Lidded: True if the plate is lidded
11. Is Sealed: True if the plate is sealed
12. Incubation Process: Records the name of the process that put the plate into incubation. Important for runs with plates being stored in the same spot for multiple incubations
13. Hotel Physical Position: Used internally to identify a hotel (or stack)

Stacks



Stacks use a separate datatable from hotels but have similar options. This document will only highlight the differences:

1. Count Plates: True/False value that tells an arm whether or not to automatically count the stack if the number of plates is indeterminate
2. Plates In Stack: Number of plates in the stack. Check “Count Plates” to make this indeterminate (Identified by “?”)
3. Direction: Input means plates are entering the system; finished plates are placed into Output
4. Stack Height: This height is used to determine how many plates will fit in a stack (should be the same as in the Configuration window)

Contact Information

Biosero, Inc.
9560 Waples Street
San Diego, CA 92121

Phone: 858-880-7376
Fax: 661-284-7583
Email: helpdesk@biosero.com

For Service: service@biosero.com

January 2019