

CueTip

Integrating Other Software Tools and Proprietary Code into a GeoCue Workflow



Applies to GeoCue, versions 7 and above.

GeoCue Group Support
February 8, 2007
Revision 2.0

Purpose:

This technical note describes the procedure for using GeoCue’s Environment Builder to integrate third-party software tools or the user’s own proprietary code into a GeoCue workflow.

Environment Builder is a powerful customization tool for GeoCue. It allows users to build their own production environments from the ground up, including creating new entity types, new layer types, new menus, and even complete new environments. It allows users to very easily ‘wrap’ their existing processes and integrate their existing workflows into GeoCue. This allows users to take advantage of GeoCue’s inherent production management capabilities without having to make major changes to their established procedures and tools. The full capabilities of Environment Builder allow GeoCue to be used as an integration platform to create an enterprise framework for other software tools, essentially using GeoCue to provide the GUI, file management, multi-user/multi-client locking, database management, workflow engine and related functions while the developers focus their efforts on their own unique code.

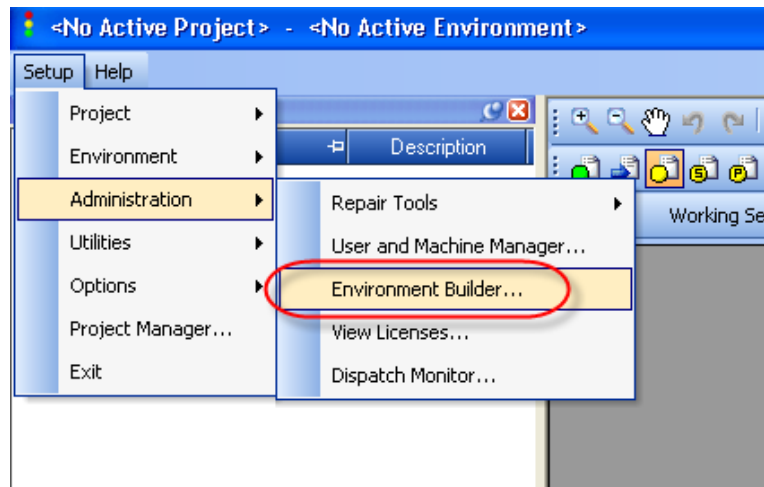
However, for most users with existing processes, workflows and tools already in place, only a subset of Environment Builder’s capabilities need to be used. Specifically, as we will discuss in this note, a client can quickly add Commands, create Checklist Steps that link to those Commands and build custom Checklists to attach to production entities (for example a LIDAR tile or working segment). Users can also easily create custom checklists or rearrange existing checklists based on the default commands that ship with GeoCue.

Environment Builder.....	2
Create a Command.....	5
Parameters	7
Create a Checklist Step.....	9
Create a Checklist.....	13
Assign Checklist	16

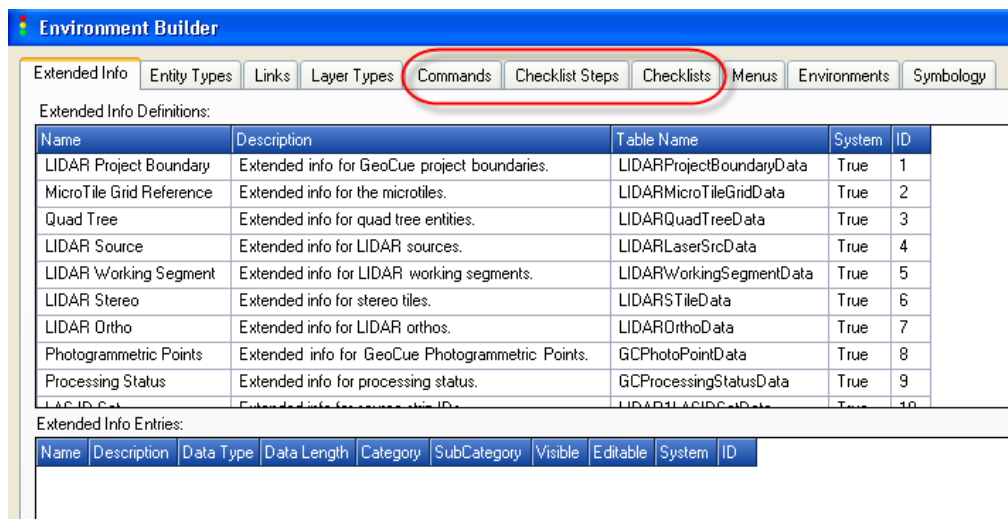
Integrating Other Software Tools and Proprietary Code into a GeoCue Workflow

Environment Builder

1. Environment Builder, included with both the Workstation and Enterprise versions of GeoCue, is an admin-level GeoCue utility accessible through the main GeoCue **Set-Up** menu:



2. Upon launching Environment Builder, a window will open and present the user with several customization options, indicated by the various tabs along the top of the screen. The three areas we are going to focus on in this note are '**Commands**', '**Checklist Steps**' or '**Steps**' and '**Checklists**':

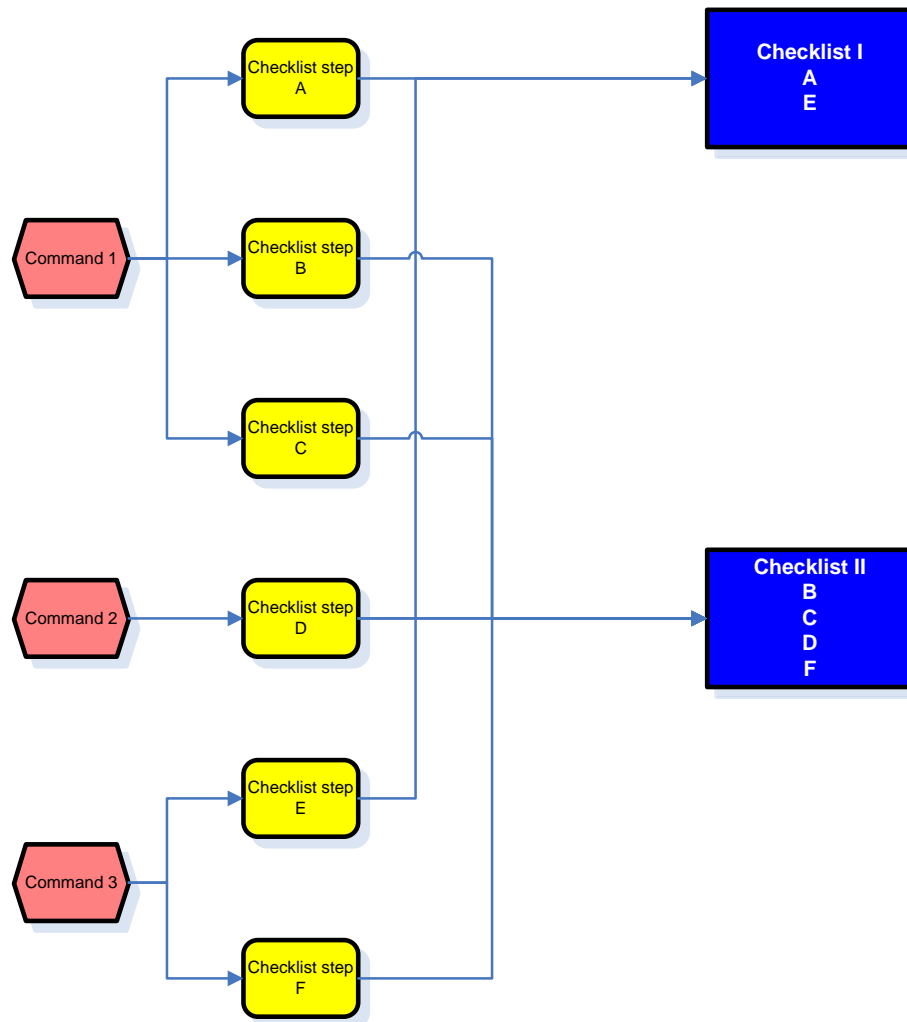


3. To provide some context for our example, keep in mind that in GeoCue, production work is managed using a **Checklist** tagged to each production entity, such as a LIDAR tile or working segment. This checklist can be thought of as a 'To Do' list. It is through the assigned checklist that production tools are launched by the user, completion status is tracked, production history (time/effort per task) is logged and work assignments are managed. A **Checklist** is

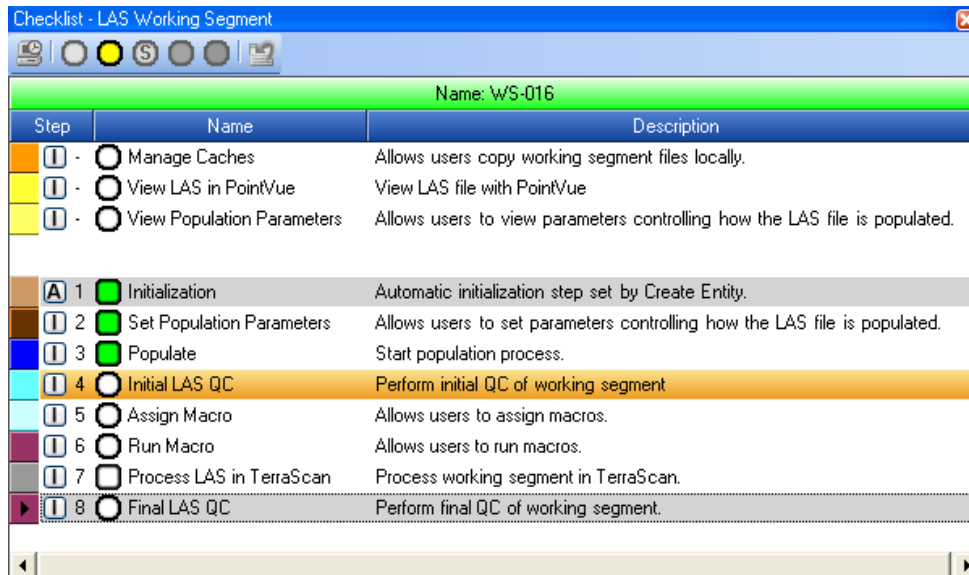
Integrating Other Software Tools and Proprietary Code into a GeoCue Workflow

simply a group of **Checklist Steps**. Steps are assigned to checklists from the complete library of all predefined, 'out-of-the-box' steps or from new steps created by the user using Environment Builder. A **Step** itself is essentially a link to an underlying **Command** that calls a specific production tool or executes a particular routine or piece of code. Note that GeoCue also supports the concept of **Checklist Steps** that do not launch commands. This is useful when you are using the checklist as a tracking tool for non-programmatic actions, such as a manual task like 'Shipped to Client'.

4. A library of predefined commands is included with GeoCue and users can create their own commands to link to other software tools or code not supported by GeoCue 'out-of-the-box'. In building checklists to manage geospatial data production, an existing command will often be assigned to a single checklist step, which is then assigned to one specific checklist. However, in general a single command can be called by many different checklist steps, and a single checklist step can be assigned to different checklists or multiple times to the same checklist, if necessary, thus allowing the user to customize the specific instance of each execution of the underlying command. The following diagram gives an example of this architecture:



5. The default checklist for LIDAR data production that ships with GeoCue is shown below. This is a typical, if somewhat basic, workflow for processing a LIDAR tile. The tile is initialized and then populated from the source data (usually a flight line strip or strips). These steps are marked as 'Complete' (green icon) on checklist shown below. The LIDAR data is then planned to go through the following steps: **Initial LAS QC, Assign(ing) a Macro, Run(ning) a Macro, Process(ing) (Editing) in TerraScan** (a point-editing tool from Terrasolid) and then a **Final LAS QC**.



As noted above, each of these steps on the checklist is linked to an underlying command in the GeoCue library. For instance, selecting and launching the **Process in TerraScan** step will invoke TerraScan, load the associated data file (LIDAR tile), track the time the process is open and then update the status when the user exits the tool (or the underlying command completes). The underlying commands can be interactive, in the case of point-editing in TerraScan, or automatic in the case of commands that simply invoke an external routine that requires no user interaction. GeoCue also supports **Dispatching** commands to remote machines and/or **Distributing** commands that act on multiple entities to multiple machines.

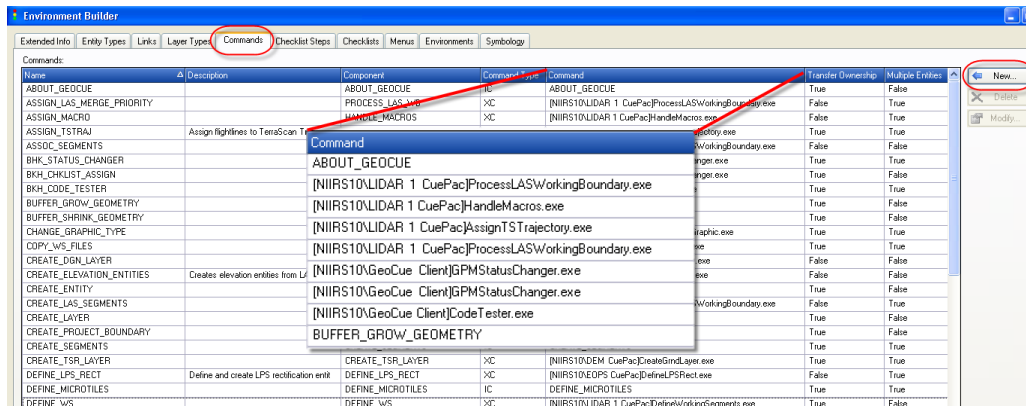
- For our example we will assume the user wants to add a step to the above checklist that allows them to call their own proprietary code, compiled as a command line executable, which takes the LIDAR tile and extracts utility pole heights to a CAD file. We will assume this step is currently done manually by the user, after ground classification is completed, by invoking the user's own code **pole_heights.exe** against each LIDAR tile.

Create a Command

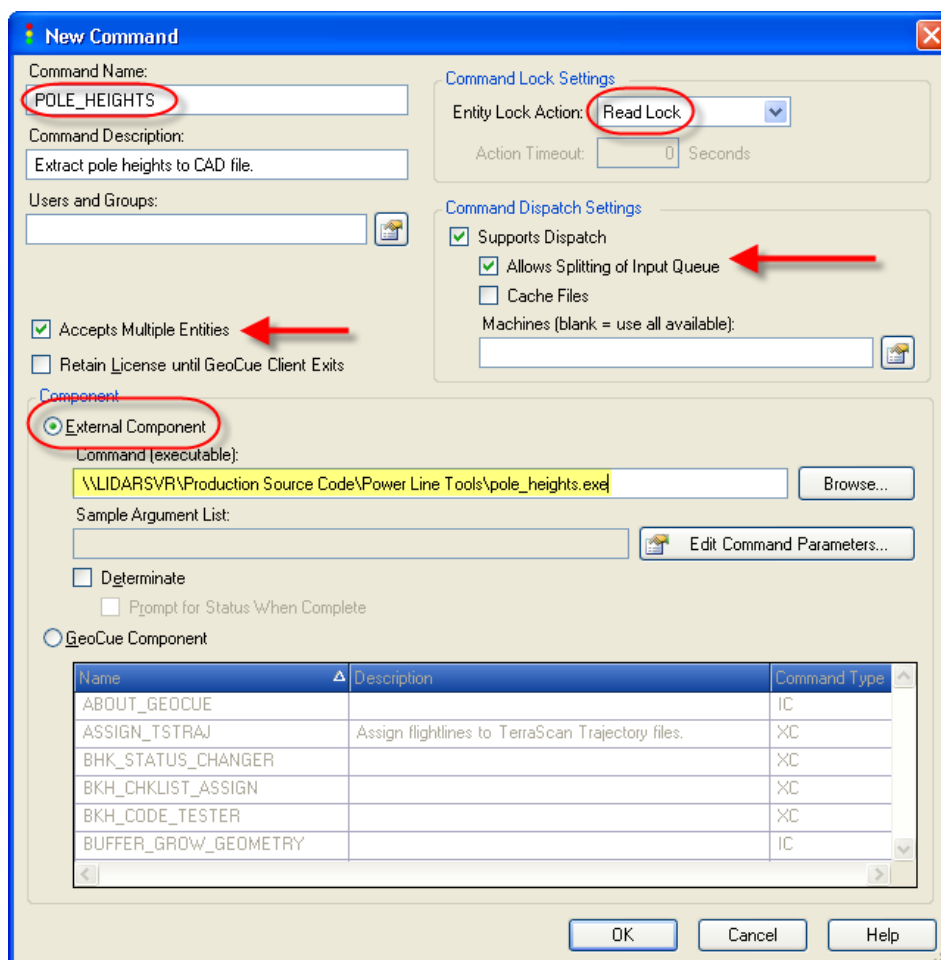
- The first step to customizing the checklist is to create a new **Command** in GeoCue that calls the user's **pole_heights** routine. This is accomplished in the **Commands** tab of Environment Builder. The main thing to remember here is that commands in GeoCue are really just pointers to code (.exe) that resides either in the GeoCue repository or elsewhere on the user's system.

CueTip

Integrating Other Software Tools and Proprietary Code into a GeoCue Workflow



8. We start creating the new command by invoking the **'New'** dialog on the **Commands** tab:

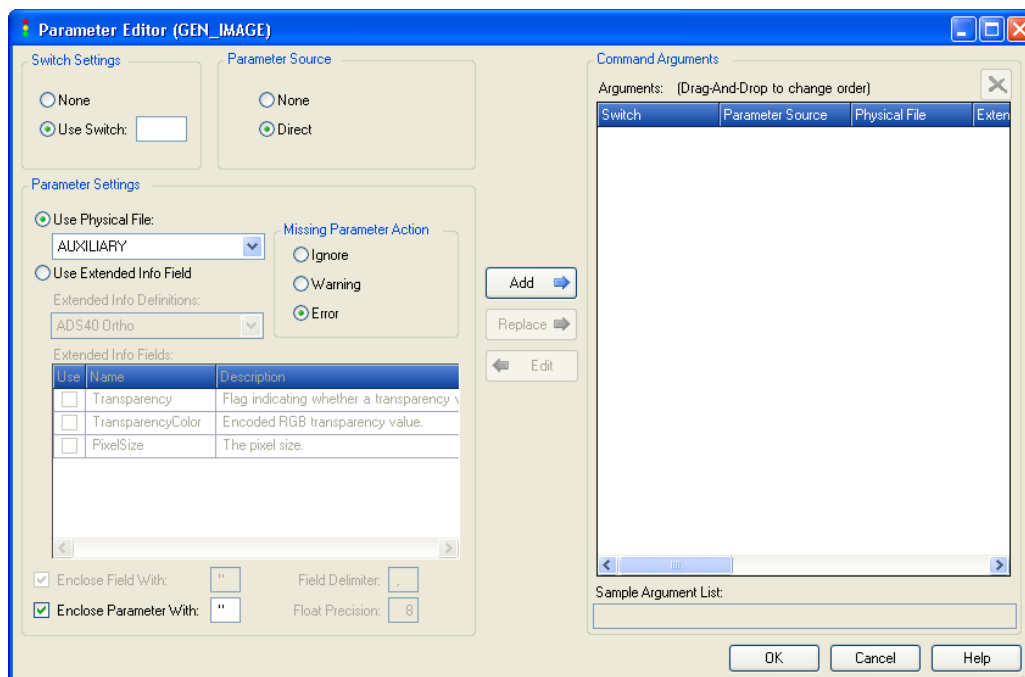


Completing the dialog as indicated above, we have assigned the new command a name **'POLE_HEIGHTS'** and set several other flags related to how we want this command to work. Under the **'Command Lock Settings'** we've set a Read Lock on the entity so when we invoke

this command, the LIDAR tile will be locked out from anybody else using it until the command finishes. Under the **'Command Dispatch Settings'** and the **'Accepts Multiple Entities'** check box, we have set-up this command so it can be **dispatched** to remote machines and a batch of multiple entities can be **distributed** across all available remote machines. Essentially, with just a few mouse clicks, we have enabled this command to accept a batch of tiles and use GeoCue's inherent distributed processing to assign those tiles across idle machines in our GeoCue network. The user is able to make their **'pole_heights.exe'** dispatchable and distributable without having to modify any of their own code. Finally, we have set the pointer to the actual executable, using a UNC path for visibility, in the **'External Component'** section.

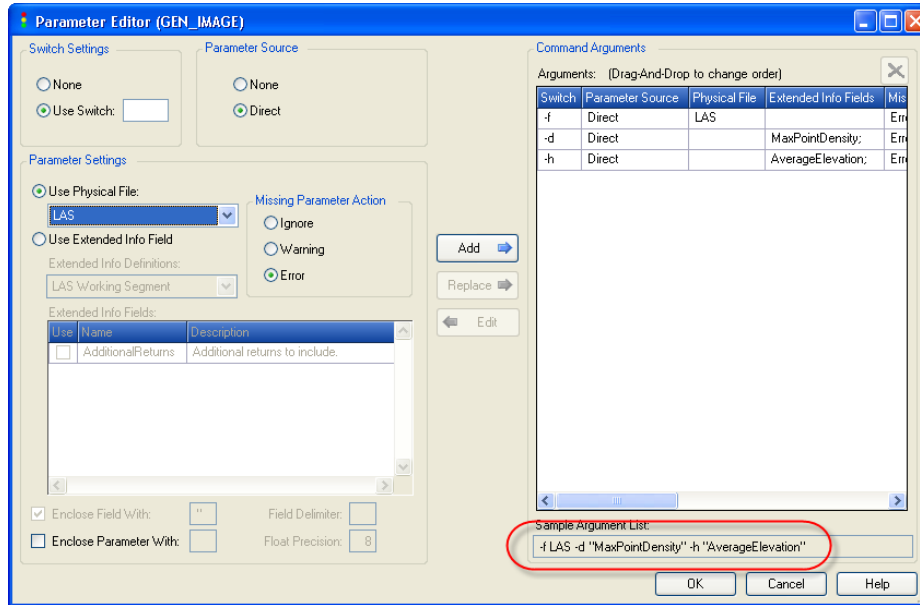
Parameters

- For purposes of this example we will also assume **'pole_heights.exe'**, like many third-party or custom-built code, needs several parameters to be passed to it to function properly. We can build detailed command lines in GeoCue, including passing variables and extended information about the entity to be processed – the working tile - by invoking the **'Edit Command Parameters'**:

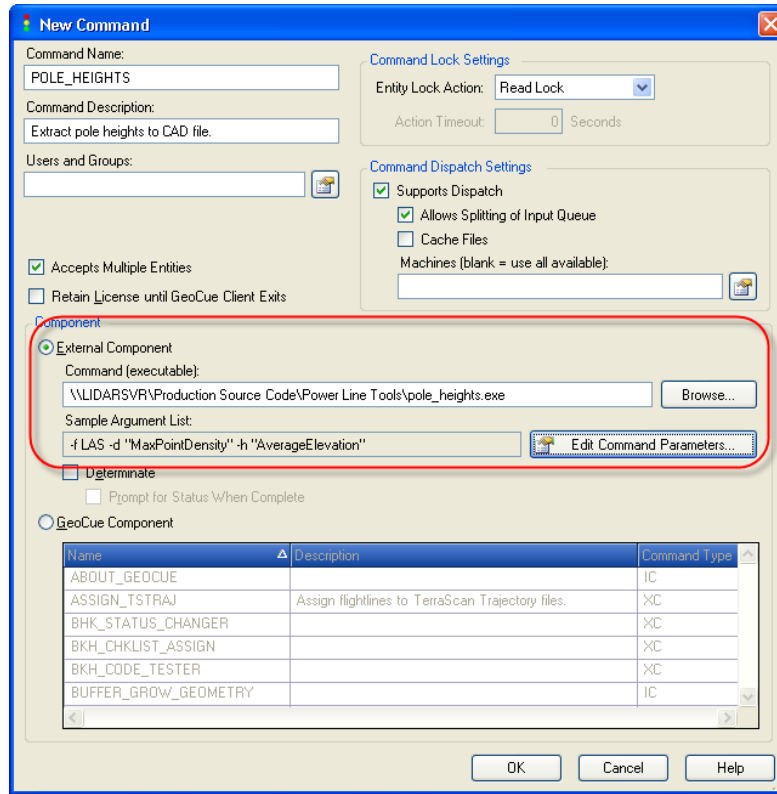


This dialog allows us to set multiple command line parameters and related behaviors of our new command. We can set switches, such as `-f` for file name, in the **'Switch Settings'** along with the source of the parameter for the switch; **'None'** or **'Direct'** from the entity in question. We can also choose **'Parameter Settings'** either from the entity itself or from extended information attached to the entity. To create the command line, we set-up each

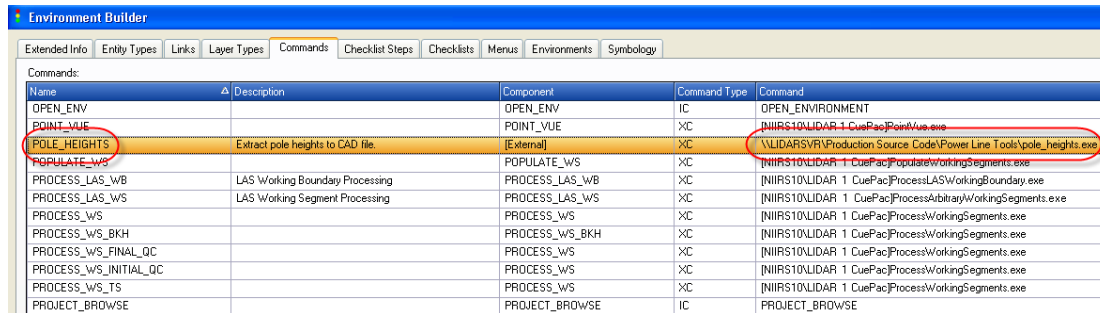
parameter and use the **'Add'** for each entry to create a compiled list of the command line we want to pass to the executable. For this example, we will pass **'pole_height.exe'** the LAS file name – the actual LIDAR point file for the tile in question - the maximum point density and the average elevation across the tile. The last two parameters are already defined as extended information for any LAS tile created in GeoCue. Users can define and pass their own extended information for entities as well, if necessary.



Adding the required parameters to our command line is simply a matter of adding each argument separately. A sample argument list is shown in the lower right of the dialog and updated as we add more parameters to the command line. When we are ready, **'OK'** will close the dialog and create the sample command line for our command.



We have now completed the definition for our new command. Selecting 'OK' will close the dialog and place it into the GeoCue command library.



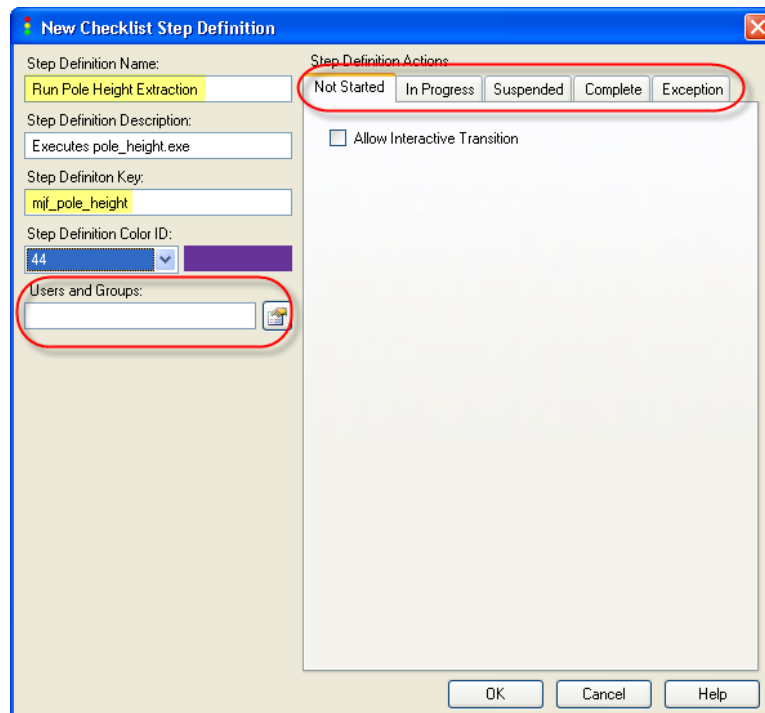
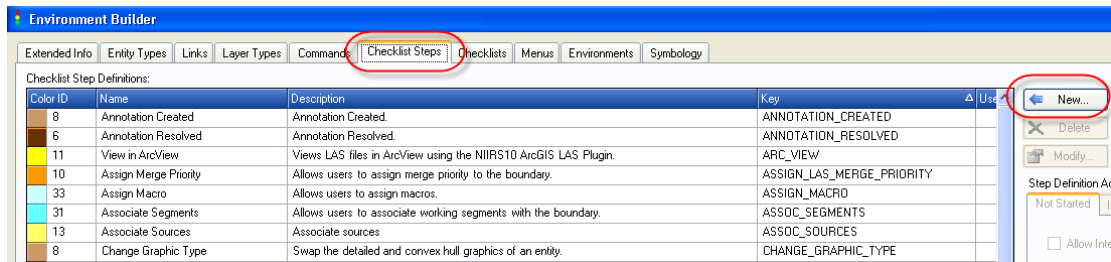
The next step we need to do is create a checklist step that calls our new **POLE_HEIGHTS** command.

Create a Checklist Step

10. We will create a single checklist step that calls the '**POLE_HEIGHT**' command. We start by selecting 'New' from the **Checklist Step** tab:

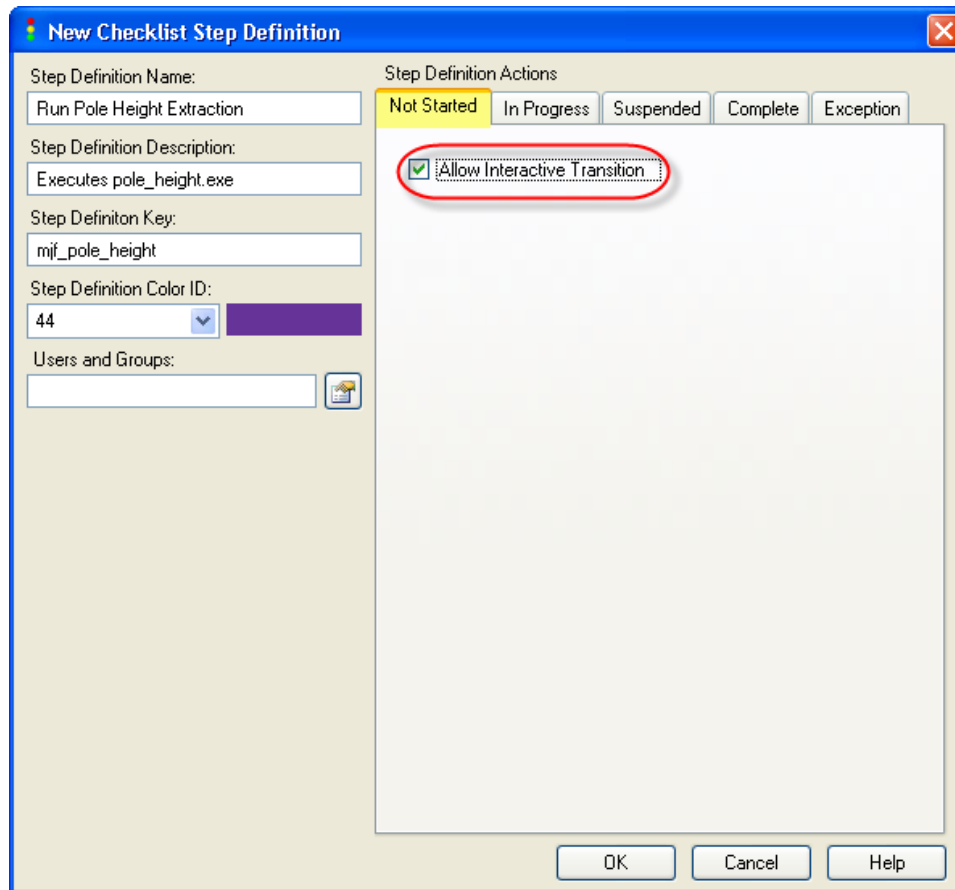
CueTip

Integrating Other Software Tools and Proprietary Code into a GeoCue Workflow

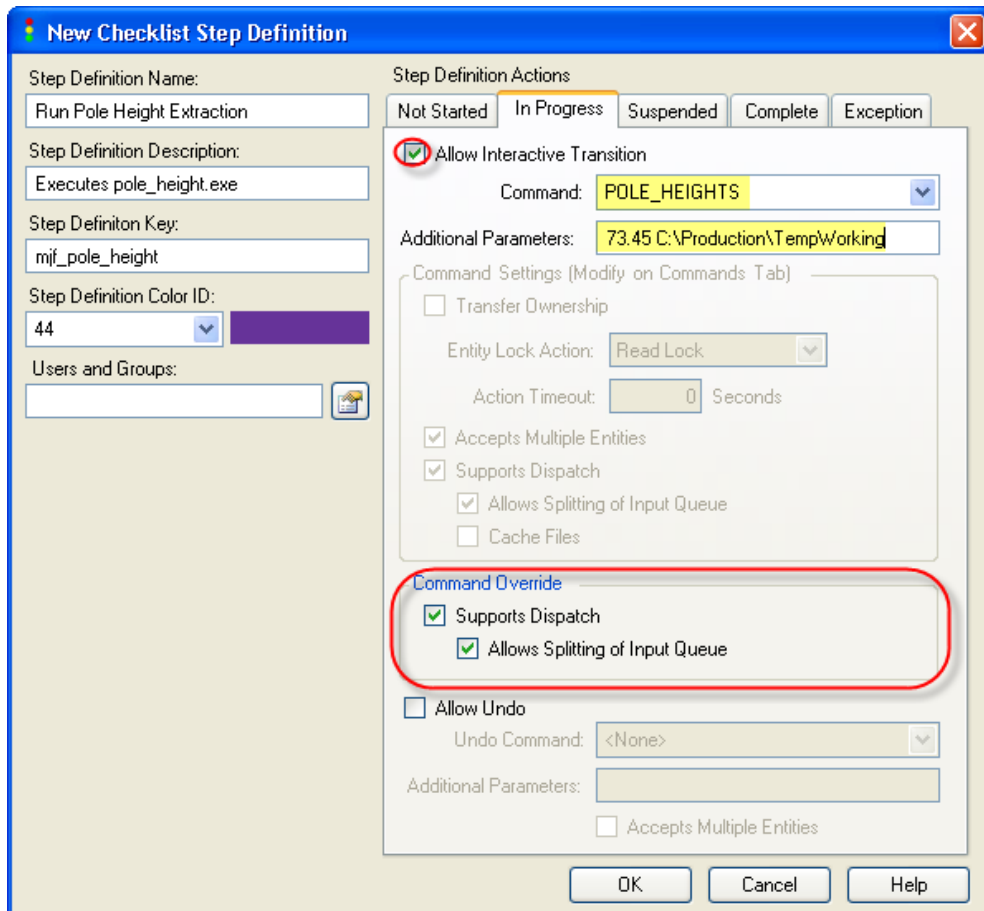


In the dialog above, we have assigned an informative **Name**, **Description** and a unique **Step Definition Key**. Each step must have a unique key, so it is useful to set-up a naming scheme for any custom steps created. Here we have used author/creator initials as a prefix. A consistent naming scheme makes it easier to track changes and find modifications in the future. Note that we could have also assigned a specific set of **Users and Groups**. This would limit the users, either by specific user or by assigned group (e.g. Transmission Line Group), who could execute this step in GeoCue. A blank **Users and Groups** field means that any user can execute the step (the desired option in this case). We can also assign **Users and Groups** at the individual checklist level as well.

- The next steps we need to complete follow the sequence of tabs across the top of the dialog. These let us define what GeoCue should do when a user changes the state of this checklist step; that is, we need to define what happens when the checklist step **'Run Pole Height Extraction'** is set to **'Not Started'**, **'In Progress'**, **'Suspended'**, **'Complete'** and **'Exception'**



For **'Not Started'**, we simply allow the step to be interactively selected by the user; the user can set any **'Run Pole Height Extraction'** step to **'Not Started'**.



The **'In Progress'** tab specifies what happens when we 'run' this step. Again, we have set it to interactive so it can be initiated by a user. We have linked the underlying command, the new **POLE_HEIGHTS** command we just created, to the step and we have indicated that this step can be dispatched to a remote machine for processing and that separate tiles (in batch mode) can be split across multiple remote machines. Also note that we can pass some additional command line parameters to this command if we need to; here we have specified a constant numeric value, perhaps a calibration coefficient already determined for the command, and a temporary directory for holding intermediate working files created by **pole_heights.exe**.

For the remaining three steps (tabs) we simply set to the default to allow interactive changes by the user if necessary. 'OK' then adds this checklist step to our library of available checklist steps:

Environment Builder

Extended Info | Entity Types | Links | Layer Types | Commands | **Checklist Steps** | Checklists | Menus | Environments | Symbology

Checklist Step Definitions:

Color ID	Name	Description	Key
26	Launch Notepad	Launch Notepad	mif_launch_notepad_03
26	Launch Notepad	Launch Notepad	mif_launch_notepad_04
26	Launch Notepad	Launch Notepad	mif_launch_notepad_05
26	Launch Notepad	Launch Notepad	mif_launch_notepad_06
26	Launch Notepad	Launch Notepad	mif_launch_notepad_07
44	Run Pole Height Extraction	Executes pole_height.exe	mif_pole_height
14	View in QT Reader	View LAS file with QT Reader	mif_qt_reader

Create a Checklist

12. The final step we need to do to complete our example is to assign this new checklist step to a checklist. In this example we are going to assume this step needs to be added to the existing checklist for LIDAR tile editing, after the ground surface has been extracted. To accomplish this, we move to the **Checklists** tab of Environment Builder, select the **LAS Working Segment** checklist and click **'New'**. In Environment Builder, whenever you click **'New'** with an existing entity selected, you will create a new entity using the selected entity as a template. Again, we assign a descriptive name and a unique key to this new checklist; **PL** (for power line) **LAS Working Segment**:

Environment Builder

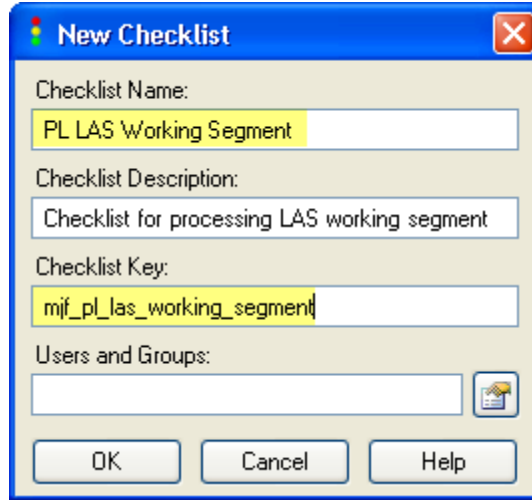
Extended Info | Entity Types | Links | Layer Types | Commands | Checklist Steps | **Checklists** | Menus | Environments | Symbology

Checklists:

Name	Description	Key	Users And Groups	System	ID
Imported Elevation	Checklist for managing imported elevation data.	IMPORTED_ELEVATION_BOUNDARY		True	24
LAS Boundary	Checklist for processing LAS working boundaries.	LAS_WORKING_BOUNDARY		True	15
LAS Working Segment	Checklist for processing LAS working segments.	LAS_WORKING_SEGMENT		True	20
LAS Working Segment (Extended)	Checklist for processing LAS working segments.	mif_ls_working_segment_extended		False	25
LIDAR Mission Boundary	Checklist for updating pre-processing steps in REALM	mif_ldar_mission_boundary		False	39
LIDAR Ortho	Checklist for processing LIDAR Orthos.	LIDAR_ORTHO		True	9

Assigned Checklist Steps: (LAS Working Segment)

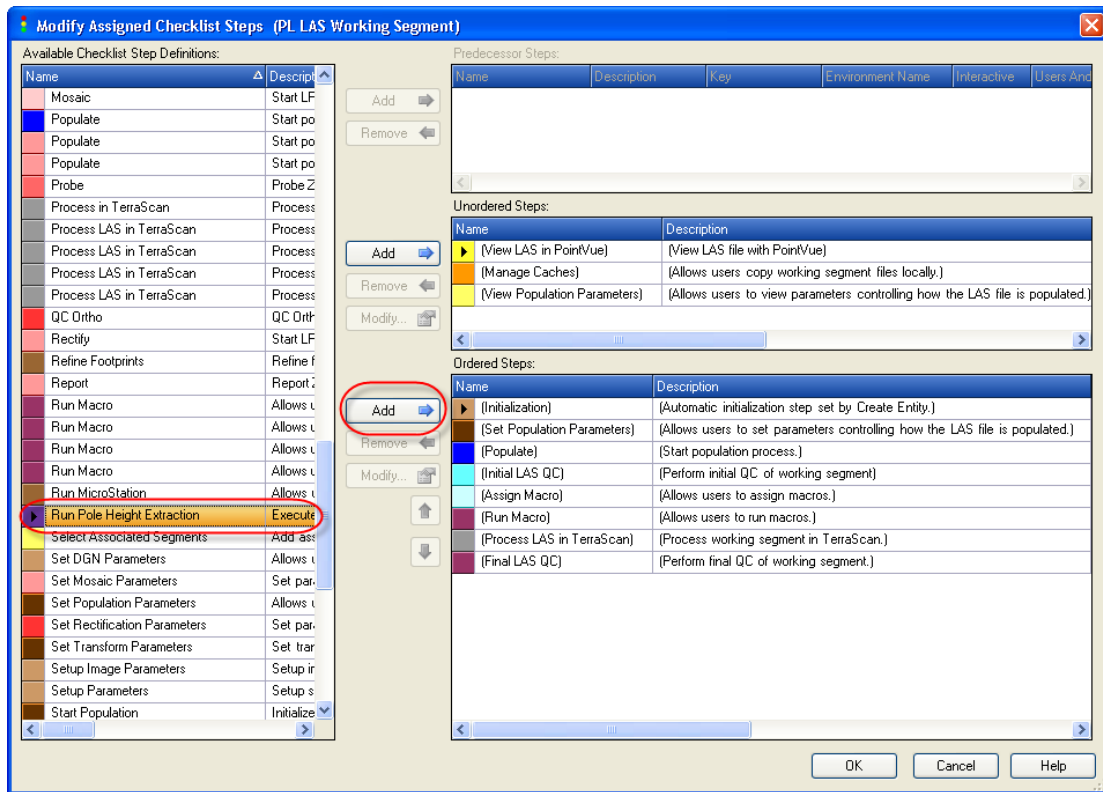
Name	Description	Key	Step Number	Color ID	Interactive	Optional	Users And Groups	Checklist Step Definition ID
View LAS in PointVue	View LAS file with PointVue	LAS_RUN_POINTVUE	0	12	True	True		733
Manage Caches	Allows users copy working segment files locally.	COPY_WS_DATA	0	10	True	True		702
View Population Parameters	Allows users to view parameters controlling how the LAS file is populated.	VIEW_LAS_SEGMENT_PARAMS	0	13	True	True		743
Initialization	Automatic initialization step set by Create Entity.	INITIALIZE	1	8	False	False		712
Set Population Parameters	Allows users to set parameters controlling how the LAS file is populated.	SET_LAS_SEGMENT_PARAMS	2	6	True	False		734
Populate	Start population process.	START_LAS_SEGMENT_POPULATION	3	38	True	False		735
Initial LAS QC	Perform initial QC of working segment	INIT_OC_LAS_WS	4	31	True	True	road1	736
Assign Macro	Allows users to assign macros.	ASSIGN_MACRO	5	33	True	True		700
Run Macro	Allows users to run macros.	RUN_MACRO	6	53	True	True		701
Process LAS in TerraScan	Process working segment in TerraScan	TS_LAS_WS	7	57	True	False		737
Final LAS QC	Perform final QC of working segment.	FINAL_OC_LAS_WS	8	53	True	True		738



13. We need to customize this new checklist by using the **'Modify Group'** option. This allows a user to assign new checklist steps, rearrange the order of existing checklist steps, or override the default names and colors of existing steps:

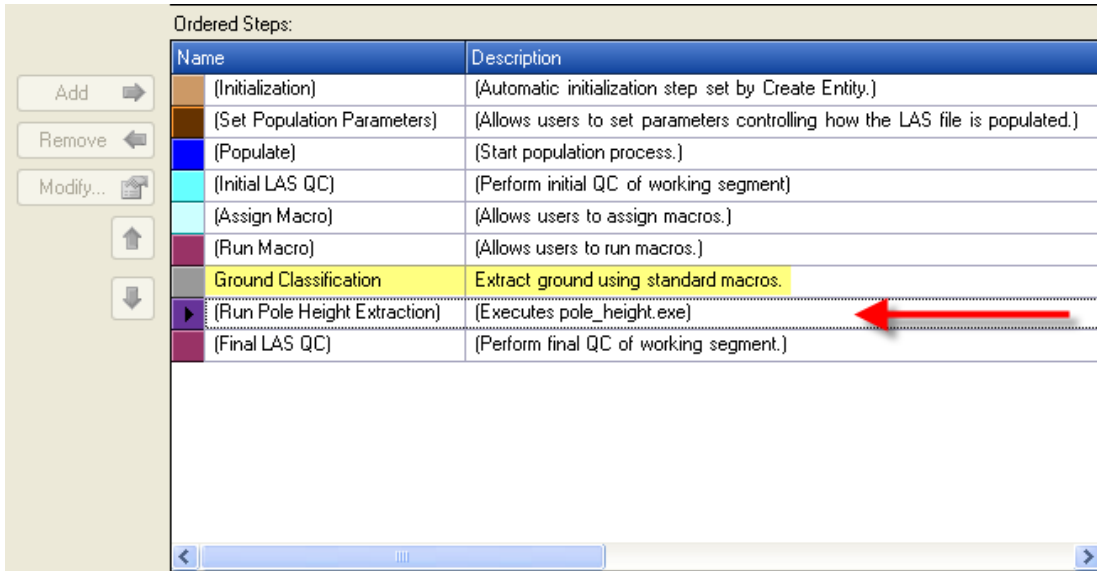
Name	Description	Key	Step Number	Color ID	Interactive	Optional	Users And Groups	Checklist Step Definition ID	S
View LAS in PointVue	View LAS file with PointVue	LAS_RUN_POINTVUE	0	12	True	True		733	F
Manage Caches	Allows users: copy working segment files locally.	COPY_WS_DATA	0	10	True	True		702	F
View Population Parameters	Allows users: to view parameters controlling how the LAS file is populated.	VIEW_LAS_SEGMENT_PARAMS	0	13	True	True		743	F
Initialization	Automatic initialization step set by Create Entity.	INITIALIZE	1	8	False	False		712	F
Set Population Parameters	Allows users: to set parameters controlling how the LAS file is populated.	SET_LAS_SEGMENT_PARAMS	2	6	True	False		734	F
Populate	Start population process.	START_LAS_SEGMENT_POPULATION	3	38	True	False		735	F
Initial LAS QC	Perform initial QC of working segment	INIT_QC_LAS_WS	4	31	True	True		736	F
Assign Macro	Allows users: to assign macros.	ASSIGN_MACRO	5	33	True	True		700	F
Run Macro	Allows users: to run macros.	RUN_MACRO	6	53	True	True		701	F
Process LAS in TerraScan	Process working segment in TerraScan.	TS_LAS_WS	7	57	True	False		737	F
Final LAS QC	Perform final QC of working segment.	FINAL_QC_LAS_WS	8	53	True	True		738	F





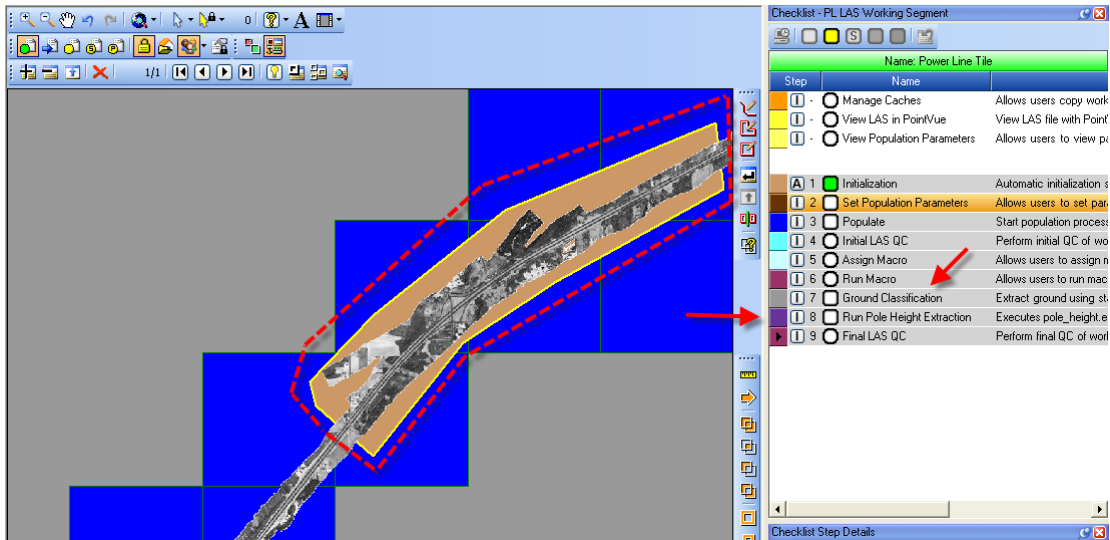
The list on the left side of the **'Modify Group'** dialog presents a list of all checklist steps in the library; we can select steps here and use the **'Add'** button to add them to the group. To set a new step so that it is interactive, we select **'Modify'** and check **'Interactive'**. We could also choose to make the new step **'Optional'** at the same time, however for this example we will assume the step is mandatory. We can also remove steps, alter the order of steps and modify existing steps. Note that in Environment Builder each checklist step has a set of default values for fields such as **'Name'** or the step color. The default values are indicated by being shown inside (brackets). The default values can be over-ridden on a checklist-by-checklist basis. A common use for this is to change the default name to provide more context-specific information for a step that is used many times across many different checklists (for example editing in TerraScan). To demonstrate, we complete our new checklist by renaming the **'Process LAS in TerraScan'** step to **'Ground Classification'** and adding our new **'Run Pole Height Extraction'** step after ground classification:

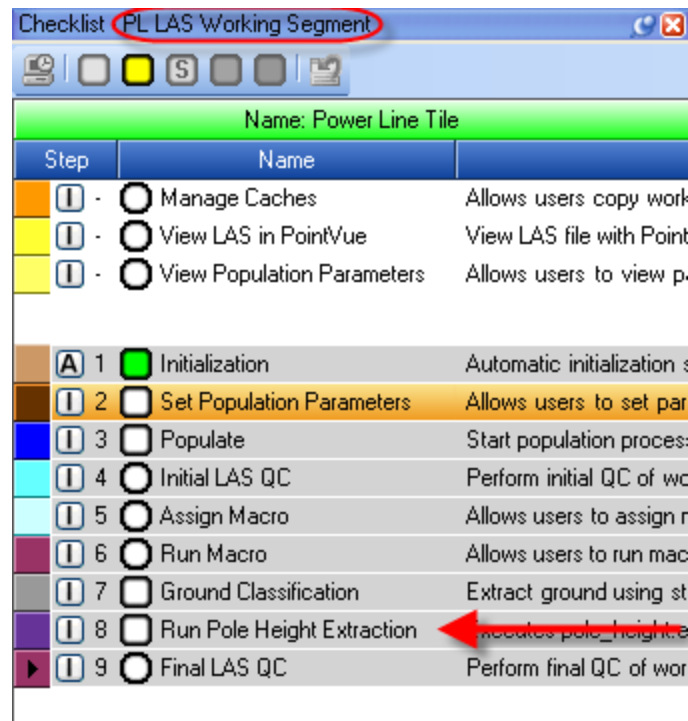
Integrating Other Software Tools and Proprietary Code into a GeoCue Workflow



Assign Checklist


14. The **'PL LAS Working Segment'** checklist is now available for assigning to any LAS working segment when they are created. Note that in the current version of Environment Builder, checklists already assigned to existing entities – LIDAR tiles - cannot be modified, so it is important to complete any customization of the workflow, that is create any necessary checklists, prior to starting work on the tiles in question. In our example project, an arbitrary tile drawn over the center of a corridor and with the new **'PL'** checklist assigned looks like this:






15. This new checklist, and the underlying command we created, allows any GeoCue user direct access to the *pole_heights.exe* custom code from within the GeoCue GUI. Selecting a tile (or group of tiles for batch processing) and executing the **Run Pole Height Extraction** step will call the command, pass the command line parameters we specified and execute the code. Integrating *pole_heights.exe* in this manner also has the added value for the user that any calls to this routine will now be part of GeoCue's normal production management environment; that is the time and effort as well as users invoking the command will be logged to the production history, the status will be included in any production reports, full multi-user locking is enforced, batch processing of multiple tiles is enabled, and transparency of file locations to users (no hunting around for source code or tiles to be processed) is maintained. Perhaps even more useful, this command is now fully dispatchable for remote processing and distributable across multiple remote nodes.

This type of Environment Builder customization can be easily extended to any third-party software or the users own proprietary code in exactly the same manner as outlined above.



For information on this CueTip, contact:

GeoCue Group Support
GeoCue Group, Inc.
9668 Madison Blvd., Suite 202
Madison, AL 35758
support@geocue.com
+1-256-461-8289



Find additional information and
participate in our GeoCue Group user
forums.

<http://support.geocue.com>