

BOSS Tile 'Done' Algorithm

Gary Kushner

January 2010 (v0.3.1)

0. Introduction

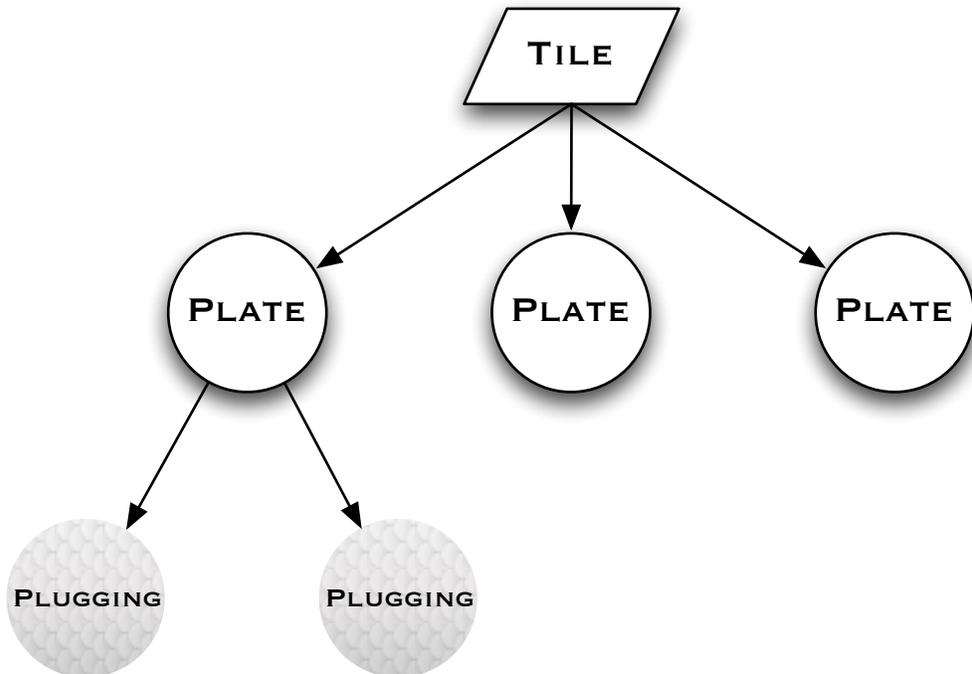
Each tile in the sky that BOSS observers will be declared done when the $(s/n)^2$ of each camera in a specific plate plugging reaches a threshold, or, the observers declare a plugging to be done. An example of when the observers might declare a plugging to be done, even though the thresholds are not met, would be when the $(s/n)^2$ in all cameras was extremely close to the required thresholds.

It is unclear at this point, but there might be an additional minimum exposure number requirements. This requirement would be to insure that there are sufficient exposures to perform cosmic ray reductions. The overhead to add this support for this requirement is slight at this point, and harder later, so it is included here. If it isn't needed, it can either remain unimplemented or the minimum exposure requirement could be set to one.

It would be useful to be able to automatically calculate whether each tile, plate and plugging is done from the information stored in plateDb.

This document describes the automatic algorithms to be used to determine when a tile, plate or plugging is done. It also describes the manual overrides that can be used to alter the automatic processing. Finally, there is a partial example design that is used to further define the problem and could be used as a basis for implementation.

1. Criteria



For any given tile in the sky, there will be one or more plates that are drilled to observe that tile. And, any given plate will have been plugged zero or more times for observations. Also, for each plugging, the plate will have taken observations zero or more times.

Criteria to determine a tile is done:

⇒ A tile is done if any plate observing that tile is done.

⇒ A plate is done if any plugging is done

⇒ A plugging is done if:

a) The observers have not manually declared the plate to be not done

b) -AND-

i) The plugging meets the $(s/n)^2$ and exposure number requirements

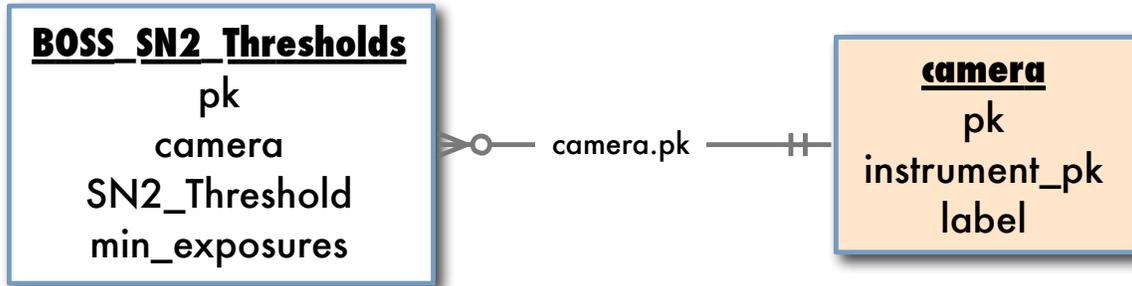
-OR-

ii) The observers declare the plugging to be done

2. Example Implementation

2A. Storing the $(s/n)^2$ and minimum exposure requirements (thresholds)

The $(s/n)^2$ requirements are currently stored in the file
 \$IDLSPEC2D_DIR/examples/opLimits.par. A table could be defined in plateDb to store these values:



In order to maintain the existing file, a python script would need to be written which would import and export the file.

The current values are 13 for the blue cameras and 24 for the red values.

2B. Storing the *plugging status flag*.

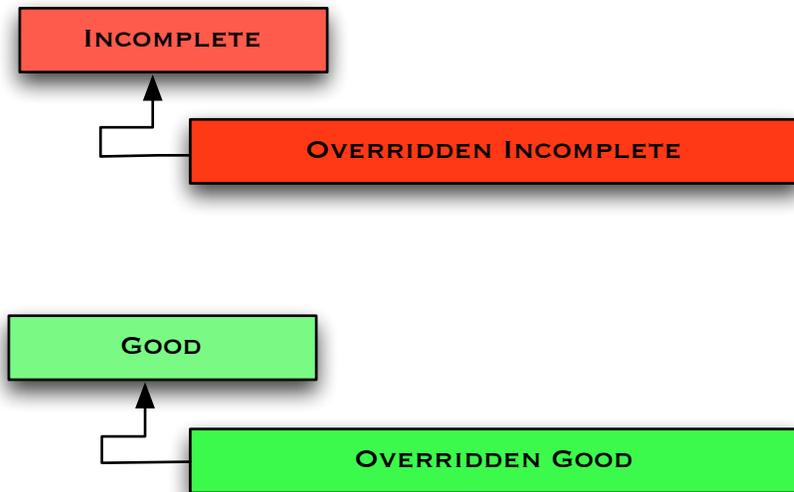
I'm not sure how tiles are stored in plateDb. It looks to me like the only reference is a tileId in the plate table. I'll make some guesses about the current design of plateDb, but instead of a presenting a detailed design, the relevant part of this section will be the definition and usage of the *plugging status flag*.

In order to store the plugging status flag, we would add two tables to plateDb. The first table would be store information about each plugging. The current information to be stored is the status of the plugging, for example done or not done; and, if the plugging has been included in a data release, then the first data release which included the plugging. The second table stores the possible status flags that a plugging could have.

BOSS_Plugging_Info		
plugging_pk	firstDR	status

BOSS_Plugging_Status_Flag
flag

The status flags would be:



The flag would be used to communicate the status of the plugging and each plugging would have exactly one state. Each lower related state in the diagram above implies the higher states. For example, Overridden Incomplete \Rightarrow Incomplete; and, Chosen For DR \Rightarrow Good. This scheme is chosen to prevent the possibility of conflicting flags.

The flag definitions would be:

Incomplete:

The plugging is not done because the $(s/n)^2$ or minimum exposures thresholds for any camera has not been met. This could be programmatically changed at any time.

Good:

The plugging is done because the $(s/n)^2$ or minimum exposures thresholds have been met for all cameras. This could be overridden by the observers at any time.

Overridden Incomplete:

The plugging is declared to be incomplete. This could be changed by the observers at any time. The observer could change the state to Overridden Good or to take the override off and programmatically the flag would be set to Incomplete or Good.

Overridden Good:

The plugging is declared to be Good. This could be changed by the observers at any time. The observer could change the state to Overridden Incomplete or to take the override off and programmatically the flag would be set to Incomplete or Good.

Some example applications of the tile status flag:

To programmatically determine if a plugging is Good:

- Find all the exposures for the plugging
- Are there enough exposures to meet the minimum exposures needed for each camera
- Calculate the $\Sigma((s/n)^2)$ for each camera and compare to the requirements.
- The plugging is not done, unless all of the $(s/n)^2$ and minimum exposures requirement is met for all the cameras

To find the status of a tile:

- Find all of the plates for the tile
- Find all of the pluggings for all of the plates
- The tile is not Good, unless any of the pluggings are Good

To find the status of a plate:

- Find all of the pluggings for all of the plate
- The plate is not Good, unless any of the pluggings are Good

2C. Exporting the Plugging Information

Currently the plugging information is stored in a flat file. In order to maintain compatibility and allow people to easily see the information, a python program would be written to export the information.

The file would be in a common format such as yanny, and will include the following information: plugging name, first DR, status flag.

Currently, the plan is that the only way to make changes to plugging status is through a UI to the database. In other words, there are no plans to allow changes made to the flat file to be imported into the database. This could be fairly easily done, where the usage scenario would be something along the lines of: export file, make changes, import file with changes. I don't like this though because I think it is prone to introducing errors and can't really be made safe. Imagine, a file is exported and then plugging 1's status is changed in the file. In the UI, someone else modifies plugging 2's status in the database. Then when the file is imported, the change to plugging 1 will be made correctly, but plugging 2 will have its recent change incorrectly undone. If an older file is imported, then the introduced errors could be much worse.

3. Processes

As each new exposure is recorded in plateDb the $(s/n)^2$ will be updated and the status flags calculated.

There would also need to be a script that would be able to run in a mode that processed all the data to insure the flags are correct and can be regenerated.

4. UI

From the web interface, the observers should be able to set a tile status flag from any plugging, plate or tile that they are working with. For example, if the observers, set a plate as Overridden Done, the UI code should traverse plateDb to file the correct tile and then set the flag for the tile.