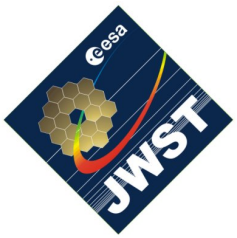
estec

European Space Research
and Technology Centre
Keplerlaan 1
2201 AZ Noordwijk
The Netherlands
Tel. (31) 71 5656565
Fax (31) 71 5656040
www.esa.int



NIRSpec Technical Note
NTN-2012-004

Author(s): G. Giardino
Date of Issue: December 20, 2012
Version: 3.0

MSA-detect tool

Abstract:

In this note we give a brief introduction to the software package *MSA-detect*. The main purpose of this tool is to derive the mapping betwen each MSA micro-shutter and the position of its image (when illuminated in imaging mode) onto the detector, in pixel coordinates. This mapping can then be used to identify failed or partially-failed open shutters in MSA all-closed exposures and closed shutters in all-open exposures.

1 INTRODUCTION

During NIRSpec calibration activities, a number of imaging exposures with the Micro-shutter array (MSA) in different configurations will be acquired to establish the precise orientation of the MSA with respect to the Focal Plane detector array (FPA) and monitor the MSA health. In particular: exposures with a MSA "checkerboard" configuration will be obtained to derive the mapping between the MSA-shutters and their projection onto the detector in pixel coordinate; exposures with the MSA "all-closed" configuration will be acquired to monitor the number and location of failed or partially-failed open shutters; exposures with MSA in "all-open" configuration will be acquired to monitor the number and location of failed-closed shutters.
Below we describe the software tool *MSA-detect* which allows the mapping between MSA-ij indexes and shutter detector-images (in pixel coordinates) to be derived. This mapping can then be used to identify failed open or failed closed shutters.

2 MSA-DETECT

The *main*-programs within this tool are located in directory `JWST_Python/MSA` of NIRSpec software repository. The programs use a library of specialized functions located in files (`mastransform.py`, `shutter.py` and `ds9regions.py`) in directory `JWST_Python/lib`. Programs and library are written in Python.
The *main*-programs are the following:

- **splitQuadrants.py** - Given an exposure count-rate file, splits the image in two, each one corresponding to the image of *one* MSA quadrant, saved into two FITS files named `quad_3.fits` and `quad_4.fits`, for SCA491 data, and `quad_1.fits` and `quad_2.fits`, for SCA492.

- **findShuttersCenters.py** - Given one of the quadrant images of the MSA (output of splitQuadrants.py), it identifies open shutters and save the xy-detector coordinates of the open shutter centers in a ds9-region file. It requires a threshold (in counts/s), giving the count value above which a typical shutter image is to be found (it does not need to be very accurate).

- **getMSAtransform.py** - Using the 'Cross-hair' exposure it derives a first order linear coordinate transform between MSA shutter centers and corresponding MSA IDs, that is the MSA ij-indexes of the shutters. Output (the forward and backward transforms) is saved in a FITS file.

- **refineMSAtransform.py** - Given a 3x3 'Checkerboard' exposures it refines the initial linear coordinate transform and derives also a distortion map that is used to have an improved overall transform MSA-to-FPA and back. The output consists of the refined forward and backward linear transforms (two 3×3 arrays) plus the five coefficients of the polynomial fit to the geometrical distortions in x and y direction and it is saved in a FITS file.

- **findFailedShutters.py** - Given a quadrant image from an all-closed exposure detects MSA failed open shutters. It requires a threshold (in counts/s), giving the count value above which a typical open shutter is to be found and the directory where the MSA-to-FPA transforms FITS file is located.

- **findFailedOpenShutters.py** - A newer variant of program `findFailedShutters.py` above, it uses the same algorithm of `findClosedShutters.py` (below) and (on simulated data) it appears to be better at handling contiguous failed open shutters. Given a quadrant image from an all-closed exposure detects MSA failed open shutters. It requires a threshold (in counts/s), giving the count value above which a typical open shutter is to be found and the directory where the MSA-mapping FITS file is located.

- **findClosedShutters.py** - Given a quadrant image from an all-open exposure detects the closed MSA shutters. It requires a threshold (in counts/s), giving the count value below which a typical closed shutter is to be found and the directory where the MSA-mapping FITS file is located.

3 USAGE

3.1 Generating the MSA-mapping

The following tutorial aims at finding the center of the FPA-image of all MSA shutters, in image coordinates (pixels). It is assumed that the user is working with two exposures: one of type 'cross-hair' and the other of type 'checkerboard3x3-1' as those generated by the IPS simulation software and located on the JWST-server in `/JWST/MSAdemo/PREP-MSA-CHKCFG-X_01/` and `/JWST/MSAdemo/PREP-MSA-CHKCFG-07_01/`.

Set-up Make sure you have added to your `$PATH` the directory

`/home/../Software/JWST_Python/MSA`

where the MSA-tool main-programs are located. In addition the PYTHONPATH environment needs to be specified:

```
setenv PYTHONPATH ${NIRSPEC_DIR}/../JWST_Python/lib
```

Step 1 Split the Cross-hair two count-rate images into the four images of the MSA-quadrants. Run:

```
splitQuadrants.py dir/$
$ NIRSPEC_491_IPSFAKEPREP-MSA-CHKCFG-X_01_S_2012-11-30T18h29m26.cts.fits dir/
splitQuadrants.py dir/$
$ NIRSPEC_492_IPSFAKEPREP-MSA-CHKCFG-X_01_S_2012-11-30T18h30m09.cts.fits dir/
```

This generates, in directory `dir`: `quad_1.fits`, `quad_2.fits` `quad_3.fits` and `quad_4.fits`.

Step 1 Generate first-order linear transform FPA-to-MSA and back, by running (e.g.):

```
getMSAtransform.py PREP-MSA-CHKCFG-X_01/quad_1_start.reg
```

The program will bring up a ds9 display window as shown in Fig.1. The user needs a this point to carefully place the provided 5 circular regions at the center of each cross and save the regions, according to the normal use of ds9. When saving the region file, ds9 will provide a default name for that file. It is important *not* to change that name. After saving this region file, the program will proceed in computing the linear transforms, that will be saved in fits file:

`PREP-MSA-CHKCFG-X_01/quad_1_initAB.fits`

Step 2 Find the shutter centers of the Checkerboard exposure. First split the Checkerboard exposure in the 4 quadrants, by running `splitQuadrants.py` with the two count-rate image as input, as described above. Note that, depending on the number of failed open shutter, you may want to first subtract from the Checkerboard exposure an 'All-Closed' exposure to remove the imprint of these failed shutters (this step was not necessary on the simulated data). Run e.g.:

`findShuttersCenters.py dir/quad_1.fits threshold`

This generates the list of centers of detected MSA shutter that will be saved in file `dir/quad_1.reg`.

Before exiting, the program will open a ds9-window displaying the result, that is the center position of the detected shutters as circular ds9-regions overlayed on the input image (see Fig. 2). This is intended to provide the user with feedback on the choice of the threshold value. If a large fraction of open shutters (say greater than 5-10%) are undetected try re-running the program with a lower threshold value. The evidence so far is that a threshold value of approximately $0.1-0.5$ the typical peak value of an open shutter should be a good choice.

Step 3 Generate the MSA-to-FPA mapping using the derived centers of the (detected) shutter in the checkerboard exposure. Run (e.g.):

`refineMSAtransform.py PREP-MSA-CHKCFG-X_01/quad_1_initAB.fits $`

`$ PREP-MSA-CHKCFG-07_01/quad_1.reg`

This program uses the output of Step 2 to improve the linear transform derived in Step 1 using 5 points and derive a map of distortion in the x and y direction, by using all the available shutter centers in the checkerboard image. The output of this program is two linear transforms (FPA-to-MSA and back) and the 6 coefficients of two (5th) order polynomial fits to the distortions between the shutters FPA position as computed using the (improved) linear transform and their observed position (as derived with program `findShuttersCenters.py`). The two 3×3 matrices and the two sets of polynomial coefficients are saved in FITS file `PREP-MSA-CHKCFG-07_01/quad_1_transforms.fits`. Before exiting, `refineMSAtransform.py` will display the result by opening a ds9-window where each MSA-shutter is represented by a circular ds9-region positioned at the shutter image center, with text providing the MSA-id (see Fig. 3).

3.2 Detecting failed-open shutters

The following tutorial aims at identifying the failed-open MSA shutters in an all-closed (IPS) simulated MSA imaging exposures. These are located in `/JWST/MSAdemo/PREP-MSA-CHKCFG-01_01` that we rename `expdir` in the example below.
We will assume that the MSA-transform FITS files obtained using the program `refineMSAtransform.py` are located in the Checkerboard exposure directory `/JWST/MSAdemo/PREP-MSA-CHKCFG-07_01`.
See Sect. 3.1 for the software set-up.

Step-1 Split the two image files of the all-closed exposure into the 4 MSA-quadrants by running `splitQuadrants.py` (as outlined in the previous section).

This command will save MSA-quadrant images `quad_1.fits`, `quad_2.fits`, etc. in directory `expdir`.

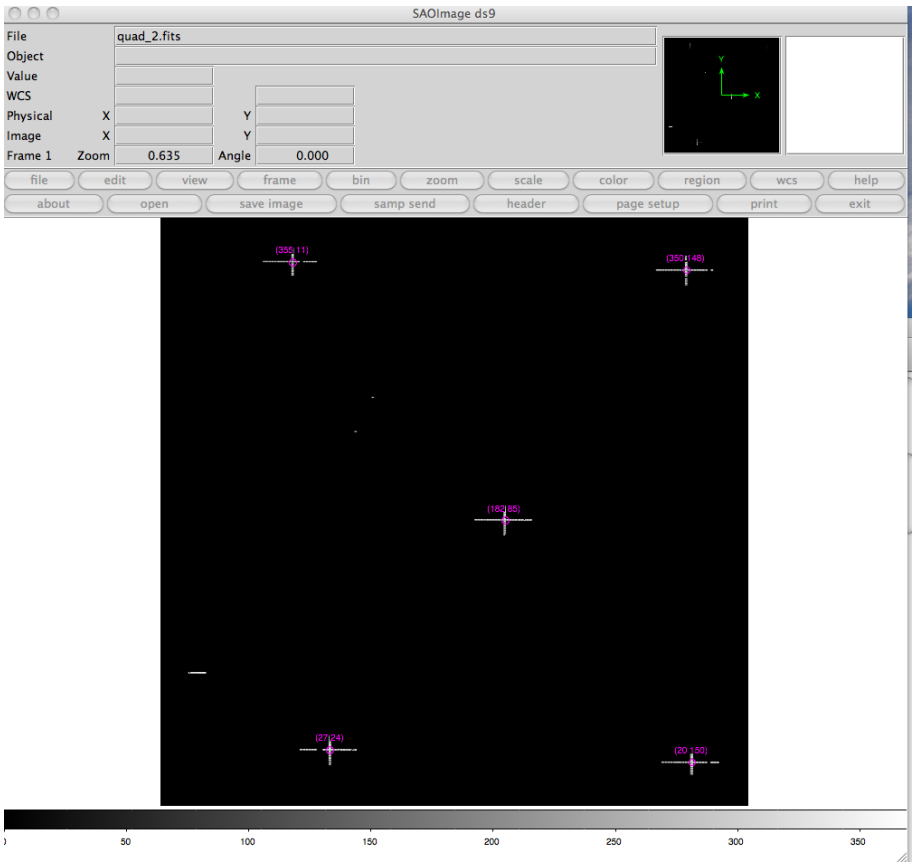


Figure 1: Cross-hair exposure of MSA quadrant 2. The centers of the 5 cross patterns are identified by a circular region. The users has manually to place these regions at the cross centers and save the result in the provided ds9-region file. Program 'getMSAtransform.py' will then compute the first order linear transform between MSA-index and FPA shutter-centers.

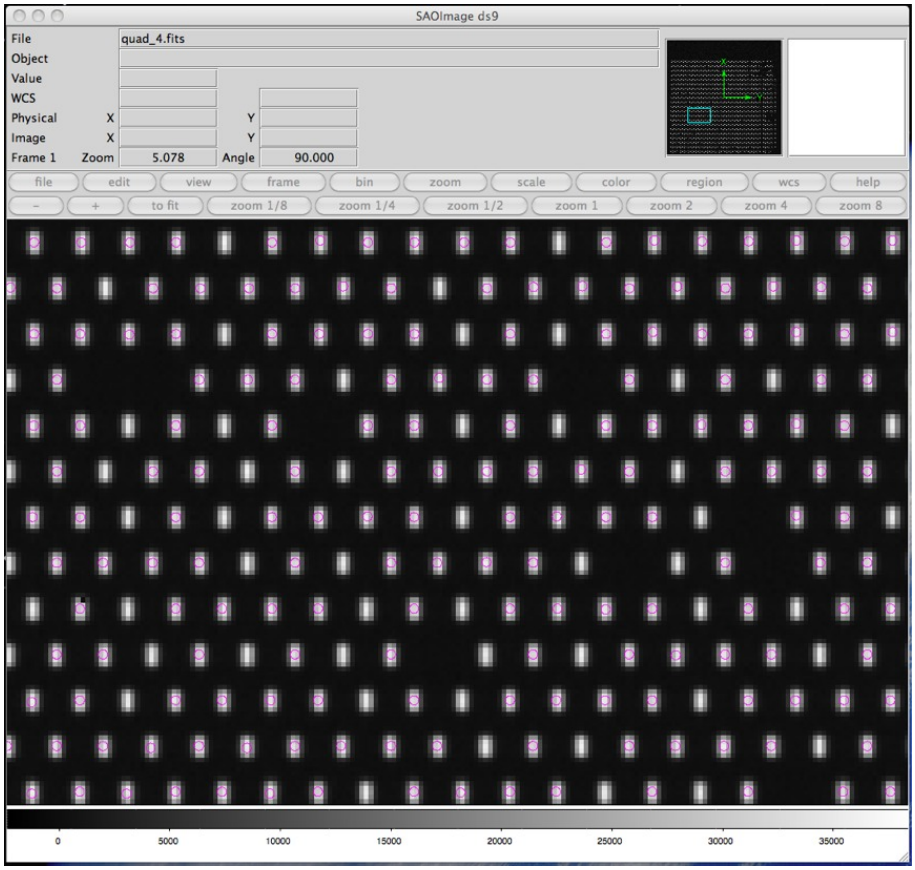


Figure 2: Zooming in a ds9-window opened by program findShuttersCenters.py that displays the center position of the detected shutters (magenta circles) overlayed over the input count-rate image of a checkerboard exposure.

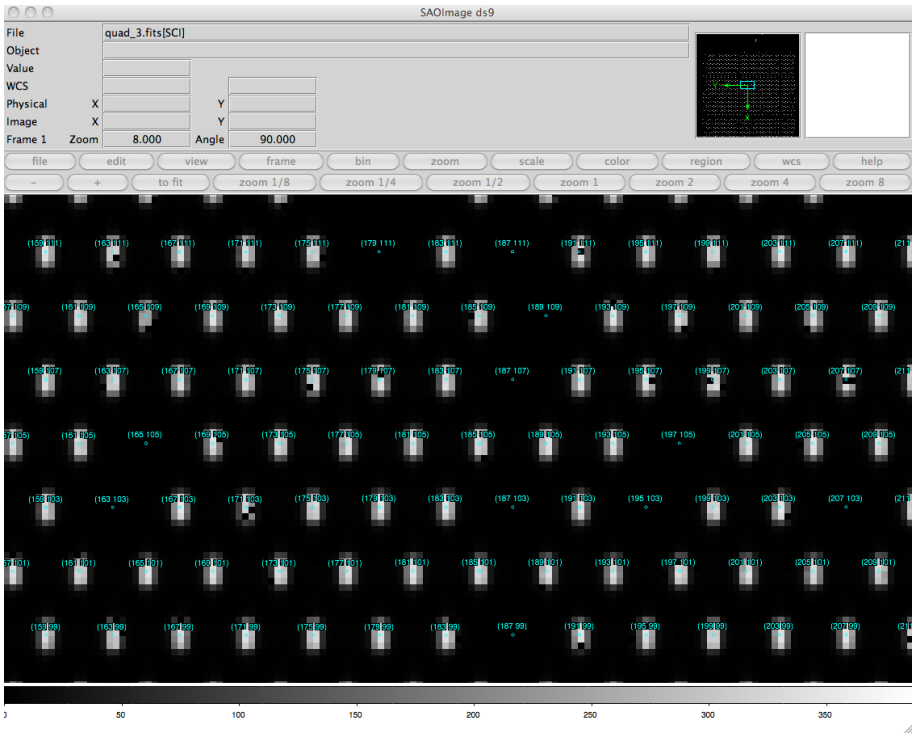


Figure 3: A zoom onto the ds9-window opened by program refineMSAtransform.py that shows the MSA-id to detector coordinate mapping as ds9-regions. For each shutter the center of each detector image is given together with its MSA-ij index. The ds9-regions are overlayed over the checkerboard image that was used to derive the improved MSA-to-FPA transform.

Step-2 Identify failed and partially-failed open shutters, by providing a threshold in image units (e.g. count/s) above which failed shutters are apparent. Note that by providing a low value for the threshold the program will also identify partially-open shutters. For the simulated exposure the value of 150.0 counts/s was used. Run:

```
findFailedShutters.py expdir/quad_1.fits PREP-MSA-CHKCFG-07_01/ 150.0
```

```
findFailedShutters.py expdir/quad_2.fits PREP-MSA-CHKCFG-07_01/ 150.0
```

etc.

The program will generate a list of the centers and MSA-id(s) of the failed shutters and save it in a ds9-region file named e.g. `quad_1_failed.reg` and located in the same directory of the input image. It will also generate a plot of failed-shutters total-flux vs average-flux that can help in identifying truly partially-failed shutters from the occasional cluster of hot-pixels (which typically for a given total-flux tends to have higher average-flux). Before exiting, the program opens a ds9-window showing the identified failed shutters (Fig. 4) overlayed over the input image, together with a view of the plot of total-flux vs average-flux for those failed shutters, in `png` format (Fig. 5).

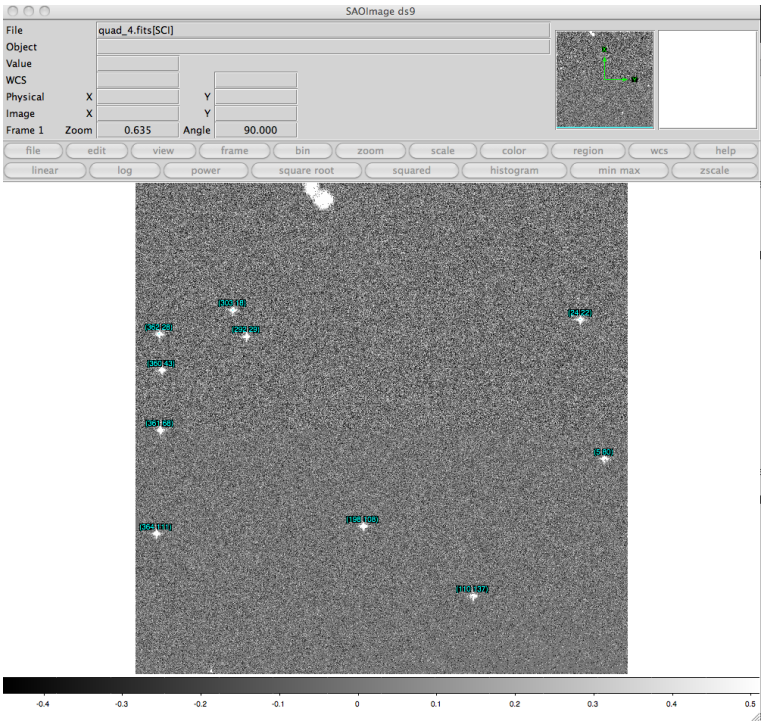


Figure 4: The ds9-window opened by program findFailedShutters.py showing failed and partially-failed open shutters in the all-closed simulated exposure (quadrant 4). For each shutter the center of each detector image is given together with its MSA-id (MSA-ij indexes).

The `png` file is also automatically saved in the same directory of the input image and the values of total-flux and average-flux are saved as comment-lines in the ds9 region file.

Alternatively, the newer program `findFailedOpenShutters` can be used. It has exactly the same calling syntax and very similar outputs.

3.3 Detecting failed closed shutters

This tutorial aims at identifying the failed closed MSA shutters in an all-open MSA imaging exposures. In the example we use a simulated all-open exposure located in directory `/JWST/MSAdemo/PREP-MSA-CHKCFG-04_01/` and the MSA-mapping that we derived from the corresponding simulation of a Checkerboard exposure, saved in files `quad_1_transform.fits`, `quad_2_transform.fits`, etc. , in directory `/JWST/MSAdemo/PREP-MSA-CHKCFG-07/` (see Sect., 3.1).

Step-1 Split the two image files from the all-open exposure into the four MSA-quadrants by running `splitQuadrants.py` This creates the MSA-quadrant images (`quad_1.fits`, `quad_2.fits`, etc.) in directory `PREP-MSA-CHKCFG-04_01/`

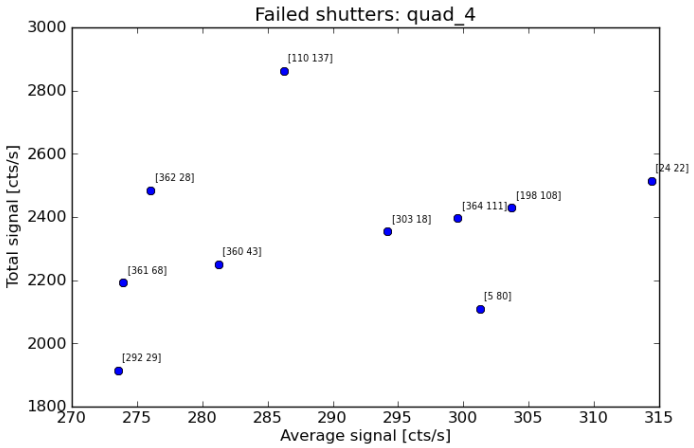


Figure 5: Plot of total-flux versus average-flux for the detector images of the failed shutters detected by findFailedShutters.py.

Step-2 Identify failed closed shutters, by providing a threshold in image units (e.g. count/s) below which shutters are clearly closed. Run (e.g.):

```
findClosedShutters.py$
PREP-MSA-CHKCFG-04_01/quad_1.fits PREP-MSA-CHKCFG-07/ 50
```

The program will generate a list of the centers and MSA-id(s) of the failed closed shutters and (if applicable) a list of MSA-id(s) of shutters falling on so many bad pixels (more than 4) that their (open or closed) status cannot be determined. These lists are saved in ds9-region files named e.g. `quad_1_closed.reg` and `quad_1_badpix.reg` (and located in the same directory as the input image). Before exiting, the program opens a ds9-window displaying the identified closed shutters as ds9-regions overlayed over the input image, as shown in Fig. 6.

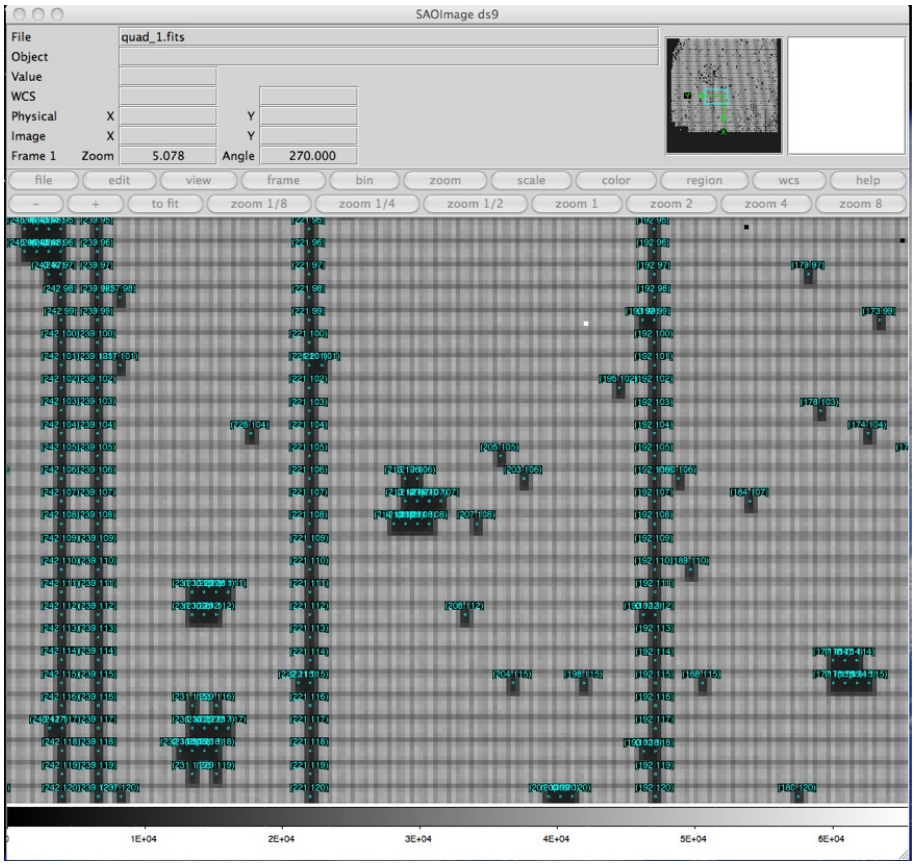


Figure 6: A zoom into the ds9-window opened by program findClosedShutters.py showing the failed closed shutters in a simulated all-open exposure. For each shutter the center of each detector image is given together with its MSA-id (MSA-ij indeces).