In [1]:
```
import matplotlib.pyplot as plt
import numpy as np

import lsst.daf.butler as dB
import lsst.cp.verify.notebooks.utils as utils
import lsst.afw.display as afwDisplay
```

In [2]:
```
# This cell should be edited to match the data to be inspected.

# Which calibration type to analyse.
calibType = 'flat'
physical_filter = 'empty~empty'
# Camera
cameraName = 'LATISS'

# Set which display to use.
afwDisplay.setDefaultBackend("matplotlib")

# Collection name containing the verification outputs.
verifyCollection = 'u/czw/DM-28920/verifyFlat.20210720Xb'
# Collection that the calibration was constructed in.
genCollection = 'u/czw/DM-28920/flatGen.20210720Xb'
```

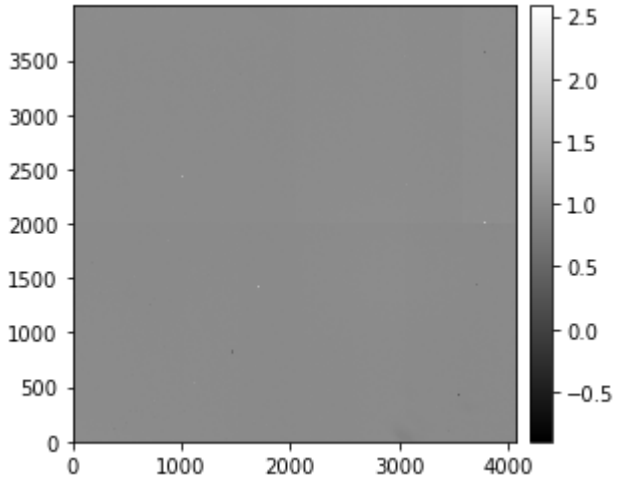
In [3]:
```
# Get butler and camera
butler = dB.Butler("/repo/main/", collections=[verifyCollection, genCollection
camera = butler.get('camera', instrument=cameraName)
```

In [4]:
```
# Get Run Statistics
runStats = butler.get('verifyFlatStats', instrument=cameraName)
runSuccess = runStats.pop('SUCCESS')
```

In [ ]:
```
display = afwDisplay.Display(dims=(1000, 1000))
display.embed()
```

In [6]:
```
# View calibration images:
continueDisplay = True
for detector in camera:
    detectorId = detector.getId()
    calib = butler.get(calibType, instrument=cameraName, physical_filter=physi
    calibArray = calib.getImage().getArray()

    # Get simple stats
    q25, q50, q75 = np.percentile(calibArray.flatten(), [25, 50, 75])
    sigma = 0.74 * (q75 - q25)
    print(f"Detector: {detector.getName()} Median: {q50}   Stdev: {sigma}")

    display.mtv(calib)
    display._scale('linear', (q50 - 3.0 * sigma), (q50 + 3.0* sigma), "")

    continueDisplay, skipNumber = utils.interactiveBlock(f"{calibType} {detect
    if continueDisplay is False:
        break
```

```
Detector: RXX_S00 Median: 0.9987115859985352   Stdev: 0.011343903541564941
flat RXX_S00 Continue? [h, c, q, p, #]c
```



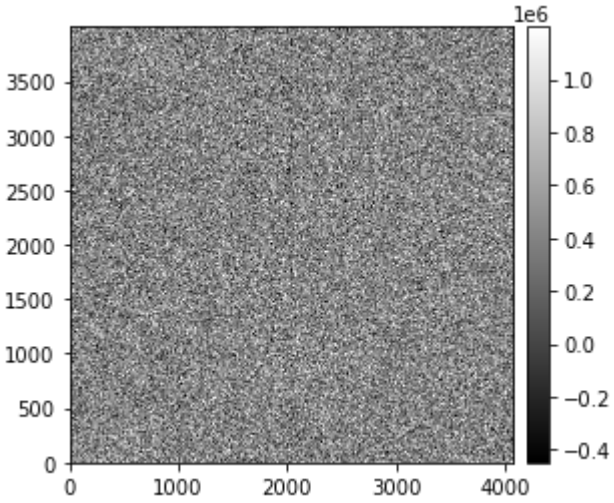
In [7]:
```
# Display summary table of tests and failure counts.
utils.failureTable(runStats)
```

Exposure	Detector
2021011900083	ALL PASS
2021011900088	ALL PASS
2021011900091	ALL PASS
2021011900092	ALL PASS
2021011900093	ALL PASS
2021011900094	ALL PASS
2021011900095	ALL PASS
2021011900096	ALL PASS
2021011900097	ALL PASS
2021011900098	ALL PASS
2021011900099	ALL PASS
2021011900100	ALL PASS
2021011900101	ALL PASS
2021011900102	ALL PASS
2021011900103	ALL PASS
2021011900104	ALL PASS
2021011900105	ALL PASS
2021011900106	ALL PASS
2021011900107	ALL PASS
2021011900108	ALL PASS
2021011900109	ALL PASS
2021011900110	ALL PASS
2021011900111	ALL PASS
2021011900112	ALL PASS
2021011900113	ALL PASS
2021011900114	ALL PASS
2021011900115	ALL PASS
2021011900116	ALL PASS
2021011900117	ALL PASS
2021011900118	ALL PASS
2021011900119	ALL PASS
2021011900120	ALL PASS
2021011900121	ALL PASS
2021011900122	ALL PASS
2021011900123	ALL PASS
2021011900124	ALL PASS

2021011900125	ALL PASS
2021011900126	ALL PASS
2021011900127	ALL PASS
2021011900128	ALL PASS
2021011900129	ALL PASS
2021011900130	ALL PASS

In [9]:
```
# This block allows the residual images to be scanned for concerns.
blinkResiduals = True
if blinkResiduals:
    continueDisplay = True
    skipNumber = 0
    for exposureId, stats in runStats.items():
        for detector in camera:
            if skipNumber > 0:
                skipNumber -= 1
                continue

            detId = detector.getId()
            residual = butler.get('verifyFlatProc', instrument=cameraName, exp
            detStats = butler.get('verifyFlatDetStats', instrument=cameraName,
            display.mtv(residual)
            display.scale('linear', 'zscale', None)

            continueDisplay, skipNumber = utils.interactiveBlock(f"{exposureId
            if continueDisplay is False:
                break
        if continueDisplay is False:
            break
```

```
2021011900083 RXX_S00 Continue? [h, c, q, p, #]c
2021011900088 RXX_S00 Continue? [h, c, q, p, #]c
2021011900091 RXX_S00 Continue? [h, c, q, p, #]q
```



In [10]:
```
utils.plotFailures(runStats, camera, scaleFactor=8)
```

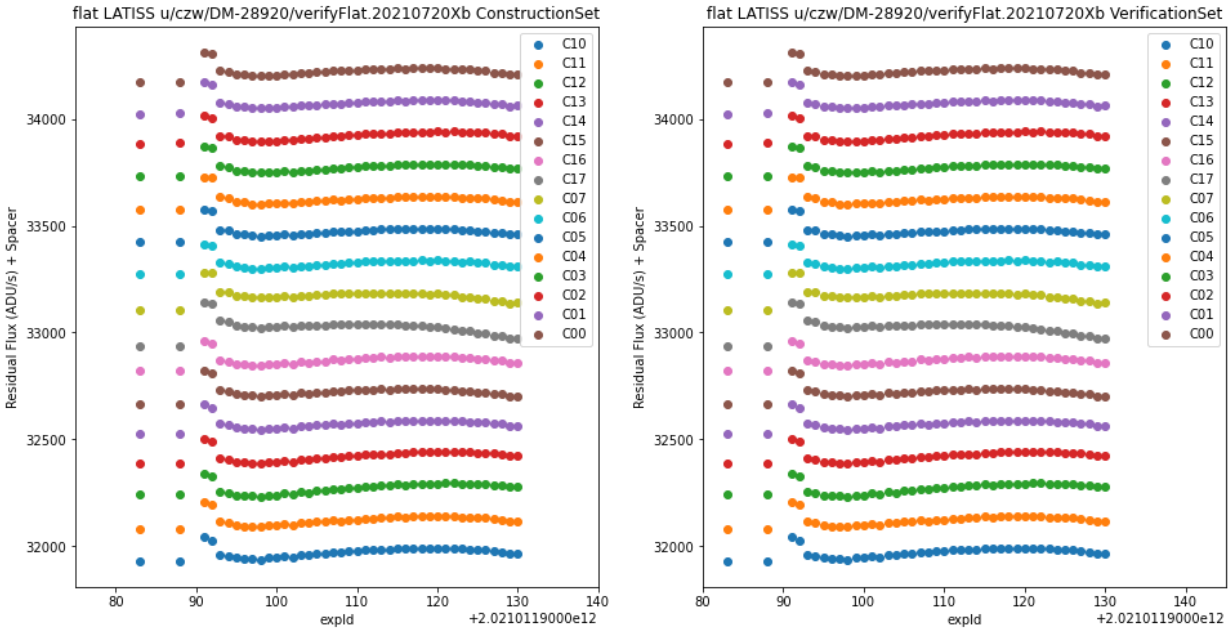
```
No failures found.
```

In [ ]:
```
# Additional cells follow here.
```

In [ ]:
```
# Get data for mean(expTime) plot.
ampMeans = {}
for detector in camera:
    ampMeans[detector.getName()] = {}
    for amp in detector.getAmplifiers():
        ampMeans[detector.getName()][amp.getName()] = {'ID': [], 'EXPTIME': []

for exposureId, stats in runStats.items():
    expTime = next(butler.registry.queryDimensionRecords('exposure',
                                                         instrument=cameraName
                                                         exposure=exposureId))
    for detector in camera:
        detId = detector.getId()
        detStats = butler.get('verifyFlatDetStats', instrument=cameraName, exp

        for amp in detector.getAmplifiers():
            mean = detStats['AMP'][amp.getName()]['MEAN']
            ampMeans[detector.getName()][amp.getName()]['ID'].append(exposureI
            ampMeans[detector.getName()][amp.getName()]['MEAN'].append(mean)
            ampMeans[detector.getName()][amp.getName()]['EXPTIME'].append(expT
```

In [16]:
```
# Plot flux as a function of exposure id, to look for time trends.
continueDisplay = True
for detector in camera:
    detName = detector.getName()

    horizontalSpace = 0.0
    verticalSpace = 150
    plt.figure(figsize=(8, 8))
    fig, axes = plt.subplots(1, 2, figsize=(2 * 8, 8))
    for axis, chunk in zip(axes, [0, 1]):
        for spacer, amp in enumerate(detector.getAmplifiers()):
            axis.scatter(np.array(ampMeans[detName][amp.getName()]['ID']) + ho
                         np.array(ampMeans[detName][amp.getName()]['MEAN']) /
                         np.array(ampMeans[detName][amp.getName()]['EXPTIME'])
                        label=amp.getName())
        axis.set_xlabel("expId")
        axis.set_ylabel("Residual Flux (ADU/s) + Spacer")


        if chunk == 0:
            axis.set_xlim(2021011900075, 2021011900140)
            axis.set_title(f"{calibType} {cameraName} {verifyCollection} Const
        else:
            axis.set_xlim(2021011900080, 2021011900145)
            axis.set_title(f"{calibType} {cameraName} {verifyCollection} Verif
        axis.legend()
    plt.show()
    continueDisplay, skipNumber = utils.interactiveBlock(detName, {})
    if continueDisplay is False:
        break
```

```
<Figure size 576x576 with 0 Axes>
```

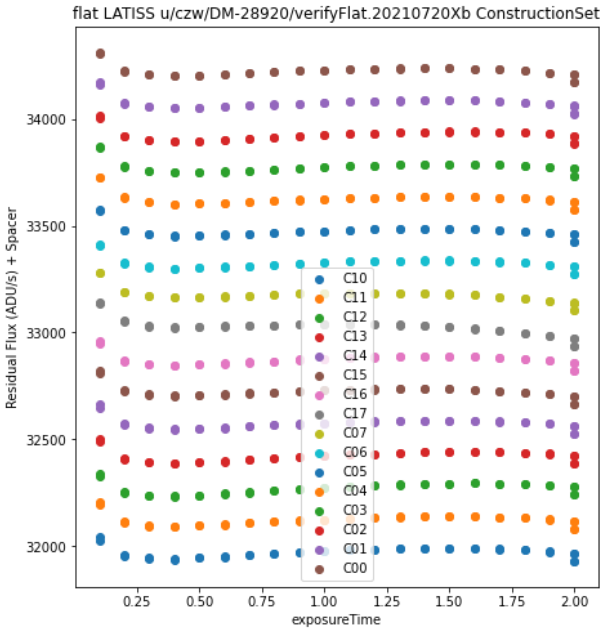


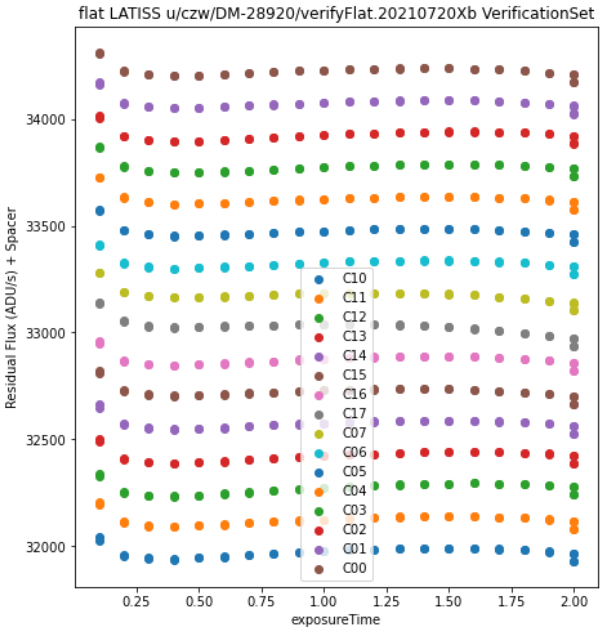
RXX_S00 Continue? [h, c, q, p, #]

In [18]:

```
# Plot flux as a function of exposure time, to confirm the residual is flat.
continueDisplay = True
for detector in camera:
    detName = detector.getName()

    horizontalSpace = 0.0
    verticalSpace = 150
    plt.figure(figsize=(8, 8))
    fig, axes = plt.subplots(1, 2, figsize=(2 * 8, 8))
    for axis, chunk in zip(axes, [0, 1]):
        for spacer, amp in enumerate(detector.getAmplifiers()):
            axis.scatter(np.array(ampMeans[detName][amp.getName()]['EXPTIME'])
                         np.array(ampMeans[detName][amp.getName()]['MEAN']) /
                         np.array(ampMeans[detName][amp.getName()]['EXPTIME'])
                        label=amp.getName())
        axis.set_xlabel("exposureTime")
        axis.set_ylabel("Residual Flux (ADU/s) + Spacer")


        if chunk == 0:
            axis.set_title(f"{calibType} {cameraName} {verifyCollection} Const
        else:
            axis.set_title(f"{calibType} {cameraName} {verifyCollection} Verif
        axis.legend()
    plt.show()
    continueDisplay, skipNumber = utils.interactiveBlock(detName, {})
    if continueDisplay is False:
        break
```

<Figure size 576x576 with 0 Axes>






```
RXX_S00 Continue? [h, c, q, p, #]
```

In [ ]: