

CALACS Operation and Implementation

Warren J. Hack
May 28, 1999

ABSTRACT

This report describes the usage and implementation of CALACS. There are several stand-alone tasks folded into CALACS, each of which can be run separately or as part of the overall pipeline. Instructions for using them and for the format of the input data are described in this paper. Furthermore, the processing steps for ACS data and the functional flow of the entire pipeline is outlined, along with descriptions of how the memory model was implemented.

1. Introduction

ACS uses detectors built by the same contractors who developed the ones for STIS, therefore, it was natural to design *CALACS* based on code used for STIS data. Lessons learned from the STIS calibration pipeline software *CALSTIS* were applied to the creation of the ACS calibration pipeline software, *CALACS*. However, the larger datasets and multiple output products imposed restrictions on what could feasibly be done using the *CALSTIS*-model. As a result, *CALACS* has evolved into a unique pipeline for processing ACS data, and may eventually serve as the basis for a calibration pipeline for WF3.

This paper not only describes the overall design of *CALACS*, but it provides details about the input data, including association tables, required for processing, as well as how to run *CALACS*. A companion ISR ACS-99-04 (Mutchler, et al., 1999) describes how this pipeline will be tested using a standard image and reference files. Details of how *CALACS* operates on the data from start to finish, with some discussion of the memory management, file I/O and management of the trailer files, are also given here. In addition, a description of each calibration function, the reference files needed, and the keywords affected are given in Section 4. A summary of the keywords used by *CALACS* during the processing are also provided. Finally, run-times for different sets of *CALACS* data on a couple of different classes of workstations are provided with an examination of the I/O and CPU usage.

2. Running CALACS

The overall design for the *CALACS* package contains fewer separate tasks and less complexity than CALSTIS, which make it easier to run. The task *CALACS* serves as the primary controller for the entire pipeline, although each of the processing tasks in the package could be run on the data individually as well. The primary function of the *CALACS* task itself will be to interpret the existing ACS association table relevant to the data, and send the data to the proper tasks in the required order for processing. Association tables are lists of exposures which are related to each other, given with the names of the output products which will be created from the exposures. These tables serve as a table of contents for all the exposures that are taken for a given observation based on a single logsheet line in a proposal. In addition to working with associated data, the *CALACS* task processes individual exposures that are not part of any association.

Many calibration steps are performed during normal pipeline processing, and Figure 1 outlines where the individual tasks within the package are called during nominal processing by *CALACS*. The standard calibration processing steps performed by *CALACS* are shown in Figure 2 for CCD data and Figure 3 for MAMA data. From these figures, calibration steps which only need to be performed on CCD data can be seen, and these resulted in the creation of the separate tasks *ACSCCD* and *ACS2D* for the calibration of CCD and MAMA data. The tasks which comprise the *CALACS* package, listed in Table 1, can also be run separately should some variation in the normal processing be desired by the user.

Table 1. Tasks in CALACS pipeline

CALACS	Controlling task for entire pipeline
ACSCCD	CCD specific calibrations
ACSREJ	Cosmic-ray rejection task
ACS2D	Basic MAMA and CCD calibrations
ACSSUM	Repeat-obs summing task

Input data for the *CALACS* task could be any of the following:

_asn file	name of association table
_raw file	name of individual, uncalibrated exposure
sub-product	name of any sub-product from an association table

The command used for running *CALACS* as a host level task follows the syntax:

```
calacs.e [tstvq] input
```

where the switches are for printing of timestamps [t] during processing, saving intermediate products [s], outputting extra comments in a verbose mode [v] during processing, and processing

in quiet mode [q]. The quiet mode still produces all the normal comments, including any extras produced by the verbose mode, however, they are sent only to the trailer file and not to STDOUT.

The input could be given as either an association table name with the ‘_asn’ suffix, a single uncalibrated image with the ‘_raw’ suffix, or a sub-product with any product suffix. For an input association file, *CALACS* will process all the exposures listed in the input ‘_asn’ file, including all intermediate products referred to as sub-products, as appropriate for the exposure. An individual exposure would be run through the entire process when the input file is a ‘_raw’ file. Finally, if the rootname of any sub-product is given, *CALACS* will search the current directory for an association table which contains the given rootname, determine which exposures are involved in producing that sub-product, then process all those exposures as necessary.

Figure 1: Flow Diagram for ACS data shown with *CALACS* task names and processing steps actually performed during each task.

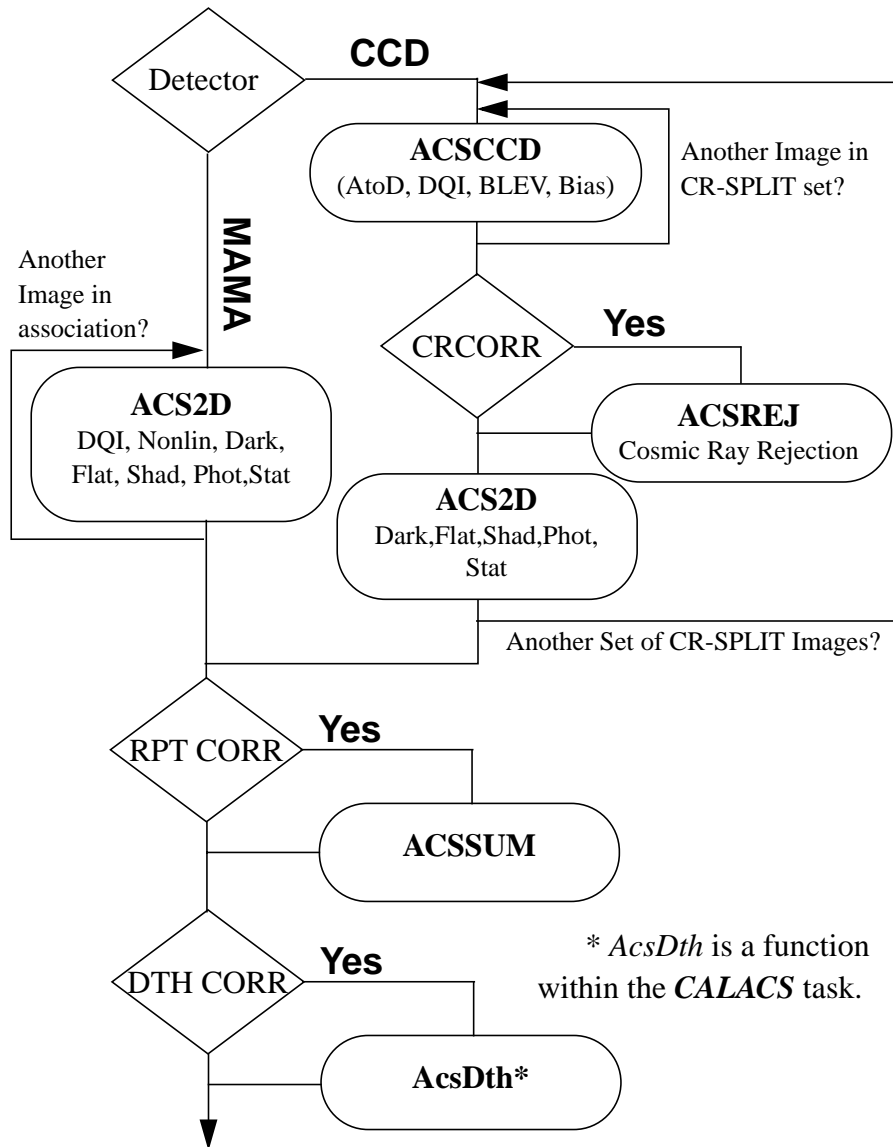


Figure 2: Flow diagram for CCD (WFC and HRC) data in *CALACS*.

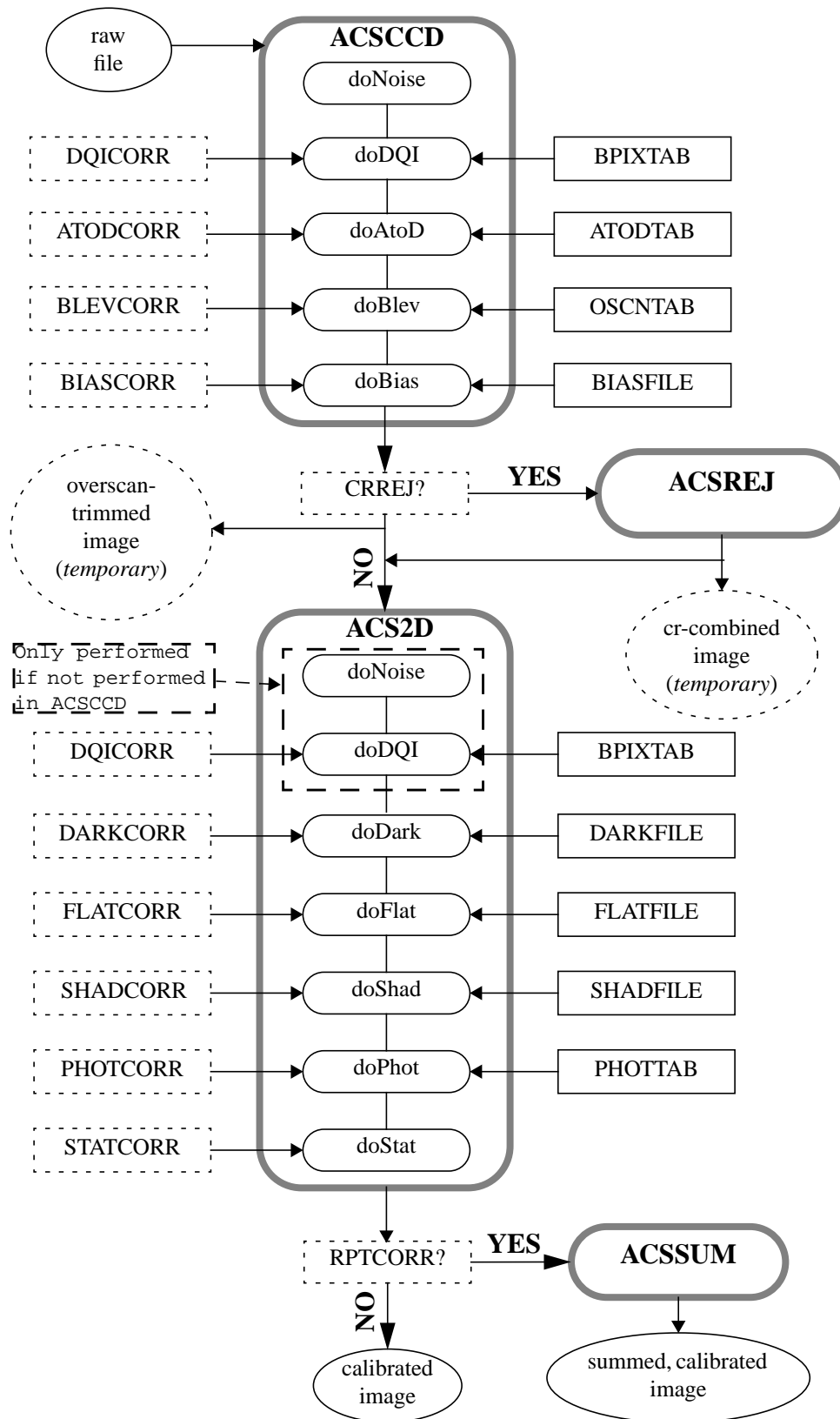
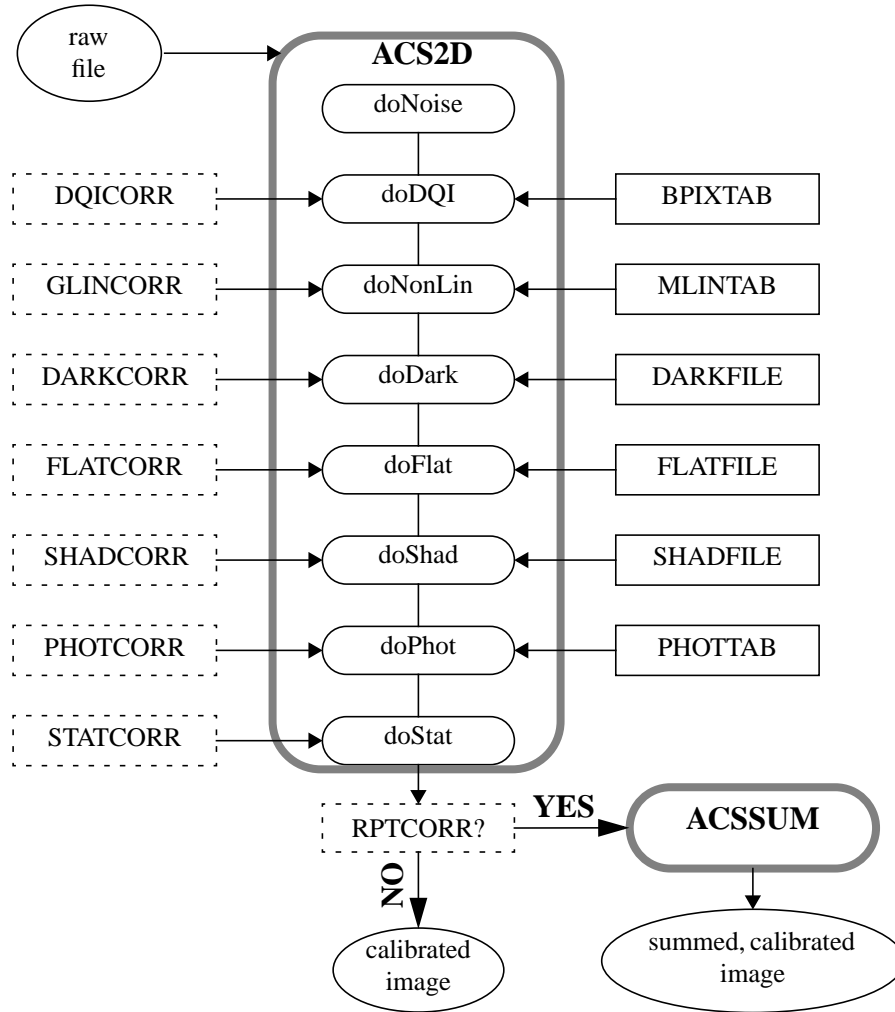


Figure 3: Flow diagram for MAMA (SBC) data in *CALACS*.



CALACS Association Table

A major component in the design of *CALACS* was the need to interpret association tables and to subsequently process the appropriate files. Association files are necessary due to the complex set of relationships that can exist between exposures taken with the Advanced Camera for Surveys (ACS).

These tables need to be in a format which closely resembles the NICMOS-style association tables, with three primary columns: MEMNAME, MEMTYPE, MEMPRSNT. Unlike the NICMOS tables, ACS tables will involve multiple output products. The MEMNAME column lists all the exposure filenames which comprise the entire association, complete with output product names. The MEMTYPE columns specifies the type of role the file has in the association. A different set of MEMTYPEs specific to ACS were adopted in order to provide the support for the multiple products. The baseline version of *CALACS* will recognize the MEMTYPE values listed in Table 2.

Table 2. Exposure types in ACS associations

MEMTYPE	Description
EXP-CRn	Input CR-SPLIT exposure for CR-combined image n
PROD-CRn	CR-combined exposure n
PROD-CRJ	CR-combined output product
EXP-RPn	Input REPEAT-OBS exposure # n
PROD-RPT	REPEAT-OBS combined output product
PROD-DTH	Dither-combined output product (<i>contains no image data at present</i>)

An example association table for a 2 position dither set with CR-SPLIT=2 is given in Table 3. The MEMTYPE for each component of the first CR-SPLIT exposure, j1xx52ecm and j1xx52egm, are given as EXP-CR1. The output product is not only given a name based on the association table's name, j1xx5201, but it is designated in the table with a MEMTYPE of PROD-CR1. This example not only shows how the MEMTYPE can be used to associate input and output products, but also demonstrates how the MEMNAME will be used as well. The last digit of each product's filename corresponds to the output product number in the MEMTYPE, with the designation of 0 being reserved for the final dither-combined output product. Since dither combining will NOT be a part of the initial version of *CALACS*, the final dither-combined product will only contain final output dither headers for each extension in the output but no image data.

Table 3. Sample Association Table J1XX52010_ASN

MEMNAME	MEMTYPE	MEMPRSNT
j1xx52ecm	EXP-CR1	yes
j1xx52egm	EXP-CR1	yes
j1xx5201 ₁	PROD-CR1	yes
j1xx52emm	EXP-CR2	yes
j1xx52eom	EXP-CR2	yes
j1xx5201 ₂	PROD-CR2	yes
<i>j1xx52010</i>	<i>PROD-DTH</i>	<i>yes</i>

CALACS will then use the number in the MEMTYPE to associate the input exposures, designated as EXP-CR or EXP-RPT, and process all of them before combining them into the output product. This output product then gets calibrated by the rest of the pipeline.

The current baseline version of *CALACS* will recognize and correctly process CR-SPLIT or REPEAT-OBS exposures resulting in either combined cosmic-ray rejected images or a combined

image of all the repeated exposures. As shown in Figure 1, the *CALACS* task would process related CR-SPLIT exposures through *ACSCCD* then combine them into a single CR-combined product using *ACSREJ*. This CR-combined product would then be processed through *ACS2D* to produce the calibrated product designated in the association table. This process would then be repeated for the remaining sets of CR-SPLIT exposures in the association table.

CALACS Data Products

Output from this calibration pipeline will consist of files containing the calibrated observations stored as IMSETs in each file. Each IMSET represents data from a single chip and consists of a SCI (science data) extension, a DQ (data quality) extension, and an ERR (error) extension. A single calibrated WFC IMSET would take up the most disk space, requiring nearly 85Mb plus the header information, for each chip’s IMSET. Raw ACS exposures which serve as input to the pipeline have the sizes given in Table 4.

Table 4. Approximate sizes of raw ACS data for each detector.

Detector	Size of FITS file (S_{raw})	X Pixels/array	Y pixels/array
WFC (2 Chips)	68.70 Mb	4144	2068
HRC (1 Chip)	4.4 Mb	1062	1044
SBC (1 detector)	4.2 Mb	1024	1024

The total size of the WFC image assumes both chips were used, while the HRC and SBC detectors only have one chip/array. These raw files also presume that only the SCI and DQ arrays are populated, and that the ERR array is a NULL array. Furthermore, the size of each WFC and HRC SCI array includes 24 columns of overscan on each side of each chip and 20 rows of virtual overscan for each chip. During processing, the SCI arrays get converted to floating point data from the input integer data. The ERR array also gets populated with floating point values, while the DQ arrays are populated with unsigned integers. As a result, the final size of the calibrated images, shown in Table 5, are much larger than the original raw images.

Table 5. Final sizes of calibrated ACS exposures for each detector.

Detector	Size of FITS file (S_{final})	X Pixels/array	Y pixels/array
WFC (2 Chips)	168 Mb	4096	2048
HRC (1 Chip)	10.5 Mb	1024	1024
SBC (1 detector)	10.5 Mb	1024	1024

The file suffixes which will be used for the ACS observations, listed in Table 6, closely mimics the suffixes used by STIS.

Table 6. Filename suffixes used by *CALACS*:

JXXYYZZT_RAW	Raw data
_ASN	Association file for observation set
_FLT	Calibrated individual exposure (not produced if CR-split)
_BLV_TMP	Overscan-trimmed individual exposure (not kept unless requested by the user)
_CRJ_TMP	Uncalibrated, CR-split-combined image (renamed to _CRJ if ACS2D is not run, otherwise deleted)
_CRJ	Calibrated, CR-split-combined image.
_SFL	Calibrated, Repeat-Obs-combined image.
_DTH	<i>Calibrated, Dither-combined image (empty image in initial version of pipeline)</i>

The initial inputs to the task *CALACS* will be the **_RAW** files from Generic Conversion and the **_ASN** (association) table for the complete observation set. Output files for a REPEAT_OBS association will be individually calibrated **_FLT** exposures and a calibrated summed **_SFL** image, while CR-SPLIT observations will produce **_CRJ** images. The **_BLV_TMP** file (*‘overscan-trimmed image’* from Figure 2) will be created for CCD images as output from *ACSCCD*, will serve as input to either *ACSREJ* or *ACS2D* and will be automatically deleted upon successful completion of *ACSREJ* or *ACS2D*. Similarly, for CR-SPLIT exposures, **_CRJ_TMP** (*‘cr-combined image’* from Figure 2) will be created as output from *ACSREJ* and will either be copied to an **_CRJ** file if *ACS2D* is not run, or simply deleted upon completion of *ACS2D*. Finally, if later versions *CALACS* perform dither-combining of **_CRJ** images, they will produce a **_DTH** image.

3. CALACS Design

ACS data provides several unique challenges to pipeline processing, not the least of which is the large size of the individual images. Interpreting the complex association tables is the most obvious difference from previous pipeline software, however, those same association tables create multiple output files, each of which should have an accompanying trailer file. Normally, trailer files were simply created by re-directing the STDOUT messages created as the pipeline is run to a file. Obviously this would not work when more than one output is created with a single run of the pipeline, so the pipeline software needed to handle the creation of the trailer files internally. Management and creation of the trailer files within the calibration pipeline represents a major departure from calibration pipelines for other instruments. In addition, *CALACS* employs line-by-line I/O to manage the large ACS data sets and reference images. The following sub-sections describe in more detail how *CALACS* actually manages the trailer files and memory requirements of ACS data.

ACS Trailer Files

All tasks within the *CALACS* package create messages during processing describing the progress of the calibration and these messages are sent to `STDOUT`. In contrast with previous calibration pipelines, however, multiple output files could be processed in one run of *CALACS*. This creates a problem as previous pipelines created trailer files by simply redirecting `STDOUT` to a file, yet there is no way to do this for the multiple output files that can be processed by *CALACS*. Therefore, each task within the *CALACS* package must decide for itself what trailer file should be used for its comments and automatically open, populate, and close each trailer file itself. In the end, *CALACS* sends messages to both `STDOUT` and to the trailer file appropriate for the observation currently being processed. Trailer files written out by *CALACS* will *always overwrite* information from previous runs of *CALACS* while preserving any comments generated by Generic Conversion. This insures that the trailer files accurately reflect the processing that was most recently applied to the data without any comments from previous pipeline processing done with potentially inaccurate reference files.

Trailer File Preface

CALACS starts its processing by interpreting the inputs as either an association table or a single exposure, and in reading in the information from any association table. During this time, *CALACS* has been creating messages describing the input situation, then proceeds on with the processing of the initial image. Until the first image to be processed has been identified, there is no file specified for any output comments. Therefore, all messages up to this point are stored in a buffer in memory and passed along until the first trailer file or set of trailer files are opened and the first comments are written out to it. This set of initial messages can be referred to as the ‘**preface**’. This preface will be always be appended to the trailer files created in Generic Conversion (if Generic Conversion comments exist) or will serve as the start of a new trailer file, and will represent the beginning of the trailer comments created by *CALACS*.

CALACS Processing Marker

A string unique to *CALACS*, *CALACSBEG*, will mark the first comment added to the trailer file by *CALACS* or any task within the *CALACS* package. If a trailer file already exists when *CALACS* is run, the trailer file will be searched for this string to determine where to start adding calibration processing comments. If it is not found, then the string will be appended to the end of the file and all comments will be appended after it. This results in any comments from previous processing being overwritten by the current processing comments to reflect the current state of calibration for the data, not its historical calibration.

Trailer File Determination

For a single image, a single trailer file will be created and, once the filename for the trailer file has been determined, the preface will be appended to any Generic Conversion comments or to the

beginning of the file if no Generic Conversion comments are present. On the other hand, multiple trailer files will be created during processing of an association table, and in these cases, the preface will be copied to the beginning of *each* output trailer file created by the first task in the *CALACS* package that processes the data. For a CCD association, all input files would be initially processed by *ACSCCD*, and output to ‘_blv_tmp.fits’ files, with corresponding ‘.trl’ trailer files. Similarly, associated MAMA data would first be processed by *ACS2D* creating ‘_flt.fits’ files and their accompanying trailer files. The trailer file for each input file in either case would contain a copy of the preface as well as its own unique processing messages. Once all the trailer files have been created successfully, the preface would then be erased from memory.

Processing would continue with *CALACS* appending the trailer files from all the input files associated with the output product into one trailer file. This can, in fact, result in multiple occurrences of the *CALACSBEG* string, but only the first one determines where to start overwriting old trailer file comments when an observation is reprocessed. Messages from current processing would then be appended to the end of the concatenated trailer file to create the final products trailer file. This duplicates some information across multiple trailer files, but for any product processed within the pipeline, the trailer file will contain all the information involved in processing all the inputs to the product.

Trailer File Safety

Throughout the processing, all trailer file comments are sent to a temporary trailer file until processing is completed, at which time the comments are transferred to the final output trailer file and the temp file is immediately deleted. This preserves all processing comments in the event of a catastrophic error during pipeline operation. Fortunately, these trailer files are usually very small, typically taking less than 20kb of space, so they do not represent any problem for disk space during or after processing.

Trailer File Functions

The functions designed to handle the messages produce a standard format for each message, with all ERROR and WARNING messages having the same prefix. In addition, the same functions send the same message to both STDOUT and the current trailer file. An option also exists for turning off the messages to STDOUT, so that the only logging of processing occurs within the trailer files.

CALACS Memory Management and ACS Data Size

Accommodating the larger ACS images required a significant change in the memory management of the IMSETS from the previous HSTIO methods. The working memory design model was implemented to keep one observational IMSET in memory during processing and built up during processing as the output image. The maximum size for an IMSET will be approximately 80Mb based on a WFC observation. As each task completes its own processing, the output file will be

written out using the currently available HSTIO functions without the need for temporary files. The only exception to this occurs in the cosmic-ray rejection task, where it needs to combine multiple images together. In this case, the one IMSET is still kept in memory as the output data, and sections of each image are read into memory as well to be compared to detection of cosmic-rays.

Line-by-line I/O has been used to read in the potentially large reference images, especially when more than one image is operated on at the same time, such as in flat-field subtraction or in summing images. In particular, the output image would be created in memory as a SingleGroup from one of the input images, then the other image(s) would be read in line-by-line and applied to the output image in-place.

Memory Usage

As a result of this memory model, processing a WFC data only requires about 95Mb of memory if no CR-combining is done, or about 135Mb when processing a CR-SPLIT association. These values should serve as upper limits to the memory usage, as WFC images are the largest observations that ACS can produce, with HRC and SBC images being 8 times smaller than a single WFC IMSET.

4. CALACS Tasks

The *CALACS* pipeline consists of 5 individual tasks, listed in Table 1, with the task *CALACS* controlling the processing rather than calibrating the data itself. The remaining tasks apply the desired calibration steps to the data and create all the output products, including the trailer files.

ACSCCD

This routine contains the initial processing tasks that need to be performed on all ACS CCD data and will perform the following processing steps (given in operational order):

doNoise	Error array initialization
doDQI	data quality array initialization
doAtoD	AtoD correction
doBlev	Bias level subtraction from overscan region
doBias	Bias image subtraction
final output	Output overscan-trimmed, or full, image

The upper level task, *ACSCCD*, uses header keyword values to determine what steps should be performed, and only performs them if needed, rather than blindly applying the correction more than once on the data. The bulk of this task was based on the code in *CALSTISI*, but then was simplified to control only those tasks applicable to ACS. Input to this task will be an image list or single image, which is automatically generated by *CALACS* when it is used or which could be

given by the user directly. *ACSCCD* processes each image in the input list one at a time and creates an output image with the *_BLV_TMP* file extension for each one. The output image contains its overscan regions throughout the processing performed in *ACSCCD*, only during final output does it get removed.

doNoise - Error Array Initialization

Header Switch: None
Header Keywords Updated: None
Reference File: CCTAB

The error array needs to be initialized in order to compute the errors during each of the calibration steps. This function examines the input data's ERR extension to determine the state of the ERR array. If the ERR array already contains an expanded array with various values, then this function does nothing, otherwise a simple noise model is used to initialize this array.

Input '*_raw.fits*' images delivered by Generic Conversion, however, contain an ERR extension without any image data. Instead, a NULL array is defined by use of the NPIX1, NPIX2 and PIXVALUE keywords, where PIXVALUE has a value of zero. This gets expanded, when read into memory by *CALACS*, into an array NPIX1 by NPIX2 in size where all the pixels have a value defined by PIXVALUE, which in most cases will be all zeroes. The function '*doNoise*' assigns a simple noise model to this array. The noise model reads the science array and for each pixel calculates the error value σ (in DN for CCD, in counts for MAMA):

$$\sigma_{\text{CCD}} = \sqrt{(DN - bias)/gain + (readnoise/gain)^2}$$
$$\sigma_{\text{MAMA}} = \max(1, \sqrt{\text{counts}})$$

This function works on both MAMA and CCD data, since it is also called when processing MAMA data as part of ACS2D.

doDQI - Bad Pixel Determination

Header Switch: DQICORR
Header Keywords Updated: None
Reference File: BPIXTAB, CCTAB

The function '*doDQI*' initializes the DQ array by combining it with a table of known bad pixels for the detector, stored in the BPIXTAB reference table. The values in the DQ array are combined using a bitwise OR operation with the values listed in the BPIXTAB. Each pixel in the BPIXTAB will have a DQ flag associated with it, and this function sets that flag in the observation's DQ array to the same value. For CCD data, the values in the SCI extension are also checked for saturation by comparison with the value of the SATURATE column in the CCD parameters

table CCDTAB. Each bit in a 16-bit word is used to flag a different error. The currently defined flags are:

Flag Value	Definition
0	Good Pixel
1	Reed-Solomon decoding error
2	data replaced by fill value
4	bad detector pixel or beyond aperture
8	masked by aperture feature
16	hot pixel
32	large blemish
64	<i>reserved</i>
128	bias level pixel
256	saturated pixel
512	bad pixel in reference file
1024	small blemish
2048	<i>reserved</i>
4096	<i>reserved</i>
8192	rejected during image combination

The DQ array may already have been populated with some values to flag pixels which were affected by telemetry problems or compression buffer overflow. This function combines the two sets of flags, from pre-processing and from the BPIXTAB, into a single result for the particular observation. The DQ array then becomes a mask of all pixels which had some problem coming into the calibrations, so that the calibration processing steps can ignore bad pixels during processing.

doAtoD - A-to-D Correction

Header Switch: ATODCORR
Header Keywords Updated: None
Reference File: ATODTAB

This step is currently not performed for ACS data, but may be needed if ground calibrations indicate. This step converts the pixel values in the raw input image from units of DN (16-bit integer) to counts (32-bit float) based on the value of CCDAMP, CHIP, and CCDGAIN. Pixels which were saturated during the exposure will be flagged in the DQ array as well. This step also converts the input image from integer data to floating point data.

For each pixel in the SCI extension, the input pixel value (still an integer at this stage) is used as an index into the correction (ATOD) array read in from the ATOD correction table (ATOD-TAB). The SCI pixel value is then replaced by the value of the ATOD array for that index. If the

input SCI value is less than zero, no change to the pixel value is made. If the input value is beyond the end of the ATOD array, the last element of the ATOD array is assigned as the SCI value, and the pixel is flagged as saturated in the data quality array.

doBlev - Bias-level Correction

Header Switch: BLEVCORR

Header Keywords Updated: MEANBLEV, CRPIX[1,2], LTV[1,2]

Reference File: OSCNTAB

The function ‘**doBlev**’ fits the bias level from the overscan region and subtracts it from the image data. The definition of the overscan regions that will be used for finding the overscan level are read in from the OSCNTAB. With the regions defined, the overscan level for each line of the input image is estimated, then a straight line will be fit to these overscan levels as a function of image line number. The fit will then be evaluated for each line of the input image, and that value will be subtracted from each pixel of the line. If the overscan level cannot be determined, or the overscan region is not present in the input image, a default value (CCDBIAS from the CCD parameters table) will be subtracted instead.

In *CALSTIS*, this bias-level subtraction also involved the removal of the overscan regions from the input image as it would no longer be necessary for subsequent processing. However, *CALACS* retains the output image in memory until the completion of all the processing steps in *ACSCCD*. Therefore, the physical and virtual overscan regions will not be trimmed off until it is written out to disk at the completion of *ACSCCD*. If the input data are binned, the pixels that are partly overscan and partly illuminated will also be removed.

The CRPIX_i and LTV_i keywords will be updated in the output to reflect the offset due to removing the overscan. In addition, the mean value of all overscan levels will be computed, and the mean will be written to the output SCI extension header as MEANBLEV.

doBias - Bias Image Subtraction

Header Switch: BIASCORR

Header Keywords Updated: None

Reference File: BIASFILE

The bias image subtraction will be performed prior to cosmic-ray rejection using ‘**doBias**’ to allow proper accounting of the bias contribution from each pixel in the CR-combined image. This function reads the bias reference image using line-by-line I/O, in order to operate with larger ACS images, and subtracts it from the input image in-place. Since the input image remains in memory throughout all of *ACSCCD*, it still contains the overscan regions, so the reference image must also contain the overscan regions, unlike the situation for *CALSTIS*. In addition, the reference bias image is assumed to already be scaled by the gain for each region of the chip appropriate to the AMP used to read it out.

Part of the dark subtraction is also included in this step, due to the way the bias reference image is created. Dark counts accumulate for an additional time beyond the exposure time, primarily the time required to read out the detector, and this part of the dark is subtracted along with the bias. This is described further in the section for DARKCORR.

Final Output - Overscan Trimming of Image

Upon completion of **ACSCCD**, the overscan regions will be trimmed from the image when it is written out, but only if BLEVCORR is performed successfully. Otherwise, if BLEVCORR was not completed, the full image with overscan regions will be written out.

ACS2D

Every observation will eventually be processed by this task, whether it is MAMA data or CCD data, as it contains the basic functions necessary for calibration. It does, however, contain the data quality initialization function used in **ACSCCD** and the error array initialization function ‘**doNoise**’ for the benefit of those observations which would never be run through **ACSCCD**. A check will be made to insure that the array initialization is not performed on CCD data twice. Calibration switches in the image header control the performance of the remaining calibration functions, with MAMA specific functions being initiated only when the MAMA specific calibration switch was set, for example.

The functions performed in **ACS2D** will be (in operational order):

doNoise	Applies a simple noise model to blank error array, <i>if not done in ACSCCD</i>
doDQI	Initialize data quality flags, <i>if not done in ACSCCD</i>
doNonLin	Apply and flag non-linear data (MAMA data only)
doDark	Perform dark image subtraction
doFlat	Combine and divide flat field images
doShad	Perform shutter shading correction for CCD, <i>if not CR-combined</i>
doPhot	Compute photometric values for header
doStat	Computes min, mean and max of good SCI data

In addition to these primary functions, other functions are used to ensure that the appropriate comments are saved to the history section of the header.

doNoise - Error Array Initialization

Header Switch: None
Header Keywords Updated: None
Reference File: None

Values in the error array are checked to be sure they are not all zero, indicating that no processing has been done on the image yet. In this case, it will perform the same expansion described in “doNoise - Error Array Initialization” on page 12 for *ACSCCD*. If, however, the input image has already been processed by *ACSCCD*, this array will already be expanded and nothing will be done to the ERR array. This same function is called in both *ACS2D* and *ACSCCD*.

doDQI - Bad Pixel Determination

Header Switch: DQICORR
Header Keywords Updated: None
Reference File: BPIXTAB

The switch value DQICORR will be checked, and if it is set to COMPLETE, this step will be skipped. Otherwise, the same initialization will be performed as described in “doDQI - Bad Pixel Determination” on page 12 for *ACSCCD*. This same function is called in both *ACS2D* and *ACSCCD*.

doNonLin - Linearity Correction for MAMA Data

Header Switch: LFLGCORR, GLINCORR
Header Keywords Updated: GLOBLIM
Reference File: MLINTAB

This routine corrects or flags global nonlinearity, and/or it flags excessive local nonlinearity. If *LFLGCORR* is PERFORM, we will check for and flag excessive global and local nonlinearity. If *GLINCORR* is PERFORM, we will check for excessive global nonlinearity (by comparing the value of keyword GLOBRATE and the GLOBAL_LIMIT from the table MLINTAB) and correct the nonlinearity if it’s not too large. The global linearity correction gets computed for every pixel below the global linearity limit specified by iteratively solving the equation $n = Ne^{(-\tau N)}$ to get the true count rate N.

If the global linearity limit is exceeded, the keyword *GLOBLIM* in the SCI extension header will be set to "EXCEEDED". If the global linearity limit is not exceeded, *GLOBLIM* will be set to "NOT-EXCEEDED" in the image header.

doDark - Dark Image Subtraction

Header Switch: DARKCORR
Header Keywords Updated: MEANDARK
Reference File: DARKFILE

This routine subtracts the dark image from the input image. For CCD data, the dark image is multiplied by the exposure time and divided by the gain before subtracting. The dark image is read in line-by-line, scaled by the exposure time (and gain, for CCD data), then subtracted from the input image in memory. The dark time is just the exposure time; it DOES NOT INCLUDE the idle time since the last flushing of the chip or the readout time. For MAMA data, the dark image is just multiplied by the exposure time before subtracting.

An overall mean is computed from the scaled dark image as it is subtracted from the science data. This mean dark value is used to update the keyword *MEANDARK* in the SCI image header.

doFlat - Flat-field image(s) Correction

Header Switch: FLATCORR

Header Keywords Updated: None

Reference File: PFLTFILE, LFLTFILE, DFLTFILE

This routine divides the science image by the combined flat-field determined from up to three flat-field reference files. The flat-field reference files are in three parts specified by the keywords PFLTFILE, LFLTFILE, and DFLTFILE. The PFLTFILE is a pixel-to-pixel flat-field correction file containing the small scale flat-field variations, and will always be specified for use. The LFLTFILE is a subsampled image which will correct for the large-scale flat-field variations. Finally, the DFLTFILE is a delta-flat containing recent changes to the small-scale flat. The LFLTFILE and DFLTFILE may or may not be specified in the SCI header, in which case, only the PFLTFILE will be used for the flat-field correction.

If all three reference files are specified, then they are read in line-by-line and multiplied together to form a complete flat-field correction which is then multiplied by the SCI data.

doShad - Shutter Shading File Correction

Header Switch: SHADCORR

Header Keywords Updated: None

Reference File: SHADFILE

This routine applied the shutter shading correction to the science data, if *SHADCORR* is set to PERFORM. The reference file, SHADFILE, is an image showing the large scale changes in exposure time across the detector as a result of the shutter opens. This correction has a significant effect only on images with short exposure times (generally less than 10 seconds long).

The pixels are corrected based on the exposure time using the relation:

$$\text{corrected} = \text{uncorrected} \times EXPTIME / (EXPTIME + SHADFILE).$$

The SHADFILE may be sub-sampled in order to save space, so this routine compares its dimensions to those of the science image. If sub-sampled, the SHADFILE will be read in by sections, expanded and then each line from the expanded section will be applied to the science image one at a time. This minimizes the memory usage by providing buffered line-by-line access to the SHADFILE. If the SHADFILE is not sub-sampled, then it will be read in line-by-line and applied normally.

This step may already have been done for CCD data if the data was combined in ACSREJ.

doPhot - Photometry Keyword Calculation

Header Switch: PHOTCORR

Header Keywords Updated: PHOTMODE, PHOTFLAM, PHOTZPT, PHOTPLAM, PHOTBW

Reference File: PHOTTAB, APERTAB

This routine gets the absolute flux conversion from PHOTTAB and the aperture throughput (filter throughput, really) from APERTAB. These are multiplied together, and then the SYNPHOT routine *PHOPAR* is called to determine the inverse sensitivity, reference magnitude (actually a constant), pivot wavelength, and RMS bandwidth. These values are then written to keywords in the primary header.

The keyword PHOTMODE is also built at this time to reflect the configuration of the instrument during the exposure.

doStat - Image Statistics Determination

Header Switch: STATFLAG

Header Keywords Updated: NGOODPIX, GOODMIN, GOODMAX, GOODMEAN, SNRMIN, SNRMAX, SNRMEAN

Reference File: None

This routine computes the minimum, mean, and maximum of the science data along with minimum, mean, and maximum S-to-N for data values that are flagged as good in the data quality array. These quantities are updated in the science image header. These values are also determined and saved for the error array.

ACSREJ Summary

Unlike CALSTIS, the cosmic-ray rejection task in *CALACS* does not keep multiple images in memory due to the potentially very large size of the images arrays (32Mb for a single WFC SCI array). Instead, line-by-line I/O was incorporated into the overall design and basic algorithm for finding and rejecting the cosmic rays applied in CALSTIS.

The processing starts by computing the initial comparison image from the list of images using line-by-line I/O. This comparison image is written out and the filename passed into the function ‘*acsrej_loop*’ for final rejection and creation of the summed output image. This routine was re-written to work with 1 output image and to read in sections of the input image and the comparison image. Once cosmic-rays are detected in the input image section, they are marked in the input file’s DQ array, then ignored when combined to create the output image. This can be seen in Figure 4, where those elements kept in memory are drawn in bold lines and file I/O is indicated by arrows.

doShad in ACSREJ

One additional change in the way *ACSREJ* will work involves the application of the shutter-shading correction, previously done in ‘*doShad*’. This correction, while small for long exposure images, can be significant for very short exposures. *CALSTIS2* does not apply this correction,

since it is applied after *CALSTIS2* completes processing the CR-SPLIT observations. The problem with that approach is that it improperly applies the shading correction to pixels that have had the values from one or more input images rejected for cosmic rays. It assumes that there have been N exposures and bases the correction on that number. To properly apply it after cosmic ray rejection would require keeping track of the number of input exposures contributing to each pixel, which is not available from the input data sets.

The way that the shading correction will be applied as part of cosmic ray rejection in *ACSREJ* is described as follows:

1. the initial guess image is computed in the same manner as *CALSTIS2*.
2. each input image pixel is compared to the initial guess image by scaling the initial guess image to match the effective exposure times of the single image and the initial guess image. The effective exposure times include the shutter shading term correction, i.e., for a single exposure, effective exposure time = nominal exposure time (as expressed by the header keyword) + shutter shading correction for that pixel. The effective exposure time for the initial guess image is the total of the individual nominal exposure times + N times the shutter shading correction for that pixel where N is the number of exposures in the CR-split set. Depending on whether or not the individual pixels value is within the acceptable range of the value derived from the initial guess image, it is marked as accepted or rejected.
3. If the pixel for that image is accepted, it is summed into the output array *unscaled*. The output array consists of a simple sum of input array values with no adjustments other than those applied before the CR rejection step. If the pixel is rejected, it is not summed into the output array. The mask for the input image is updated to indicate a value of 1 for accepted pixels and 0 for rejected pixels. (The format of the mask array will be described in more detail below)
4. when all input images have been processed, every pixel in the summed output array is scaled to reflect the proper number value corresponding to the total nominal exposure time (the same for all pixels). This is done by scaling the pixel by the factor

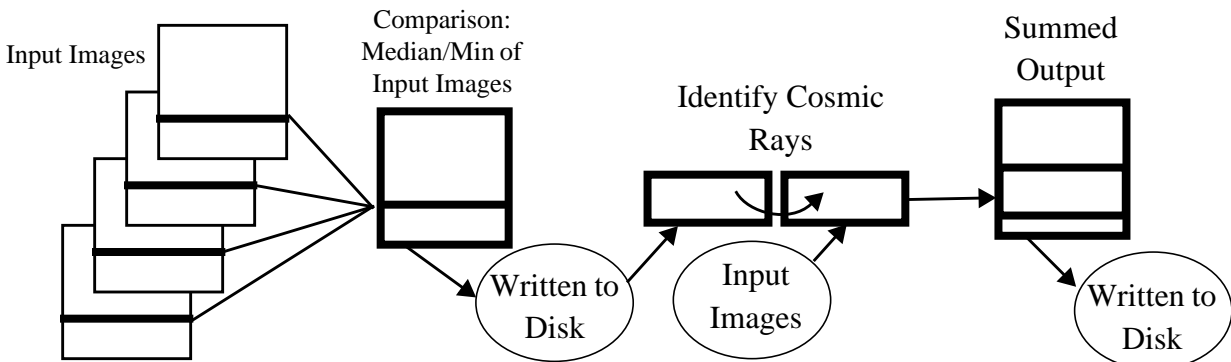
$$(\sum_i (T_i + \Delta t_i(x,y)) m_i(x,y)) / T$$

where T_i is the nominal exposure time for image i , $\Delta t_i(x,y)$ is the shading correction for the pixel x, y , and $m_i(x,y)$ is the mask value for the i 'th image at pixel x, y , and T is the total nominal exposure time for all input images (regardless of whether they were all used for that particular pixel). In short, the cosmic ray rejection step sums all the accepted values, computes the true exposure time for that pixel and scales the sum to correspond to the total nominal exposure time.

A mask array will be used instead of an accumulated exposure time array throughout this task. This array will use 1 bit for every pixel requiring only 1MB per input image. In contrast, an accumulated time array would require keeping another 32MB array in memory during processing, a requirement which was deemed excessive in addition to the 80Mb of memory already being used for the output image. The accumulated exposure time will be calculated from the mask arrays

using the above equation after all input arrays are processed and will require much less memory if the shading correction can be represented as a highly subsampled image or in a functional form (e.g., a 2nd order polynomial).

Figure 4: ACS Cosmic Ray Rejection



Summary of ACSREJ Operation

In short, the calculations will be done on each image one section at a time, and the summed cosmic-ray corrected image will be built up in memory one section at a time as well, along with a bit-mask reflecting what images contributed to each pixel. The bit-mask will then provide the information the exposure time for each pixel for use during the combining of the images and the application of the shutter-shading correction (if it was set to be done). This minimizes the amount of memory necessary for the computations, but the line-by-line I/O represents a limiting factor for the processing speed for many systems.

ACSSUM Summary

This routine stands as probably the simplest task in *CALACS*, simply summing a list of images together one line at a time and updating 4 keywords to reflect this operation. In general, *CALSTIS8* was used with very little revision. The primary modifications allowed the use of file list read in from the association file and the use of line-by-line I/O for the input images, while deleting all code related to spectral orders.

The function ‘**SumGrps**’ was revised to loop over image files rather than extensions of one image, since STIS associations were always contained in the same image file. The first image on the list will be read into memory, just as it is done in *CALSTIS8*. However, only enough memory for 1 line of data will be allocated for use by the remainder of the input images, then each image will be read in line-by-line using the new HSTIO functions. The function then loops over each line in the input image and adds it to the output image in memory. The calibration switch RPT-CORR is set to COMPLETE upon successful completion of the summation. In addition, the keywords NCOMBINE, EXPTIME, and EXPEND are adjusted to reflect the total of all the summed images. With these keywords updated, the summed output image is then written out.

5. Keyword Usage

Processing of the images in *CALACS* will be controlled by the population of keywords and calibration switches in the input image headers, as it is in CALSTIS. Those keywords essential for *CALACS* remain a subset of those used by CALSTIS, with keywords related to spectroscopy and to binning of the images being absent. Table 7 provides a summary of those keywords which used by *CALACS*, with those that are unique to ACS marked appropriately.

Table 7. ACS Keywords Used in *CALACS*

Keyword	Task(s)	I/O	Header	Sample of Possible Values
APERTURE	acsccd,acs2d,acssum	Input	Primary	WFC1,WFC2,WFC1-FIX, WFC2-FIX,WFC,HRC,SBC,...
ASN_TAB	calacs	Input	Primary	
ATODGNA, ATODGNB, ATODGNC, ATODGND	acs2d,acsccd,acsrej	Input/Output	SCI(Primary?)	
BADINPDQ	acsrej	Output	Primary	
BINAXIS1,BINAXIS2	acsccd, acs2d,calacs	Input	Primary	
CAL_VER	acs2d,acsccd,acsrej,acssum	Output	Primary	
CCDAMP	acsccd,acs2d,acsrej	Input	Primary	ABCD,ABC,ABD,AC,AD,ACD, BC, BD, BCD
CCDCHIP	acs2d,acsccd,acsrej,acssum	Input	SCI,DQ,ERR	1,2
CCDGAIN	acsccd,acs2d,calacs	Input	Primary	1,2,4,8
CCDOFSTA, CCDOFSTB, CCDOFSTC, CCDOFSTD	acs2d,acsccd	Input	Primary	
CD1_1, CD2_2	acs2d,acsccd	Input/Output	SCI,ERR,DQ	
CRPIX1,CRPIX2	acsccd,acs2d	Input/Output	SCI,ERR,DQ	
CRRADIUS	acsrej	Output	Primary	
CRMASK	acsrej	Output	Primary	
CRSIGMAS	acsrej	Output	Primary	
CRTHRESH	acsrej	Output	Primary	
DARKFILE	calacs,acs2d	Input	Primary	
DETECTOR	calacs,acsrej,acssum	Input	Primary	WFC,HRC,SBC
EXPEND	acssum	Input/Output	Primary	
EXPTIME	acsccd,acs2d,acsrej,acssum	Input/Output	Primary	
FILENAME	acsccd,acs2d,acsrej,acssum	Output	Primary	
FILTER1, FILTER2	acs2d,acsccd,acssum	Input	Primary	CLEAR1L,F555W,F775W,F625W,F550M, F850LP,CLEAR1S,POL0UV,POL60UV,...
GLOBLIM	acs2d	Output	SCI	EXCEEDED, NOT-EXCEEDED, NOTAPPLICABLE, UNDETERMINED
GLOBRATE	acs2d	Input	SCI	
GOODMAX	acs2d	Output	SCI,ERR	
GOODMEAN	acs2d	Output	SCI,ERR	
GOODMIN	acs2d	Output	SCI,ERR	
INITGUES	acsrej	Output	Primary	
INSTRUME	calacs	Input	Primary	ACS
LRFWAVE	acs2d	Input	Primary	
LTM1_1,LTM2_2	acs2d,acsccd	Input/Output	SCI,ERR,DQ	
LTV1,LTV2	acs2d,acsccd	Input/Output	SCI,ERR,DQ	
MEANBLEV	acsccd	Output	SCI	
MEANDARK	acs2d	Output	SCI	

Keyword	Task(s)	I/O	Header	Sample of Possible Values
MEANEXP	acsrej	Output	Primary	
NCOMBINE	acsrej,acssum	Input/Output	Primary	
NEXTEND	calacs,acssum,acsrej	Input/Output	Primary	
NGOODPIX	acs2d,acssum	Output	SCI, ERR	
NPIX1,NPIX2	calacs(acssth)	Output	SCI	
NRPTXP	acssum	Input	Primary	
OBSTYPE	acs2d	Input	Primary	IMAGING
OSCNTAB	calacs, acsccd	Input	Primary	
PHOTFLAM	acs2d	Output	Primary(SCI?)	
PHOTMODE	acs2d	Output	Primary(SCI?)	
PHOTPLAM	acs2d	Output	Primary(SCI?)	
PHOTBW	acs2d	Output	Primary(SCI?)	
PHOTZPT	acs2d	Output	Primary(SCI?)	
PIXVALUE	calacs(acssth)	Output	SCI	
READNSEA, READNSEB, READNSEC, READNSED	acs2d,acsccd,acsrej	Input/Output	SCI	
REJ_RATE	acsrej	Output	Primary	
ROOTNAME	calacs,acsrej,acssum	Input/Output	Primary	
SCALENSE	acsrej	Output	Primary	
SDQFLAGS	acs2d,acsccd,acssum	Input	Primary	
SKYSUB	acsrej	Output	Primary	
SKYSUM	acsrej	Output	Primary	
SNRMAX	acs2d,acssum	Output	SCI	
SNRMEAN	acs2d,acssum	Output	SCI	
SNRMIN	acs2d,acssum	Output	SCI	
STATFLAG	calacs	Input	Primary	T,F
TEXPTIME	acsrej	Output	Primary	
WRTERR		Input	Primary	
ACS Calibration Switches				
ATODCORR	acsccd, calacs	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
BIASCORR	acsccd, calacs	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
BLEVCORR	acsccd, calacs	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
CRCORR	calacs,acsrej	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
DARKCORR	calacs,acs2d	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
DQICORR	calacs,acsccd	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
DTHCORR	calacs(acssth)	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
EXPCORR	calacs	Input	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
FLATCORR	calacs,acs2d	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
GLINCORR	calacs,acs2d	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
LFLGCORR	calacs,acs2d	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
PHOTCORR	calacs,acs2d	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
RPTCORR	calacs,acssum	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED
SHADCORR	calacs,acs2d,acsrej	Input/Output	Primary	COMPLETE,OMIT,PERFORM,SKIPPED

6. Run-time Efficiency

Efficient operation of the code was a prime concern during development, as it will have to deal with images that take up to 32Mb of memory for each array, or up to 85Mb for each IMSET con-

taining the SCI, ERR and DQ arrays. Line-by-line I/O provides a savings in the amount of memory which is required during operation, but relies on the I/O sub-system speed more. Therefore, the operational efficiency and run-time for *CALACS* depends on a combination of factors, not just the CPU speed or simply the amount of memory.

CALACS Speed Tests

Some tests have been performed using ACS data on a couple of different systems which span the expected range of workstation performance used in astronomical research. The first test system was a Sun Sparcstation 4 with 96Mb RAM, a 110Mhz CPU, and a SCSI-2 hard disk system. Both a Sun Ultra-10 workstation (252Mb RAM, a 300Mhz CPU, and an Ultra-SCSI hard disks) and an Ultra-60 workstation (636Mb RAM, 360Mhz CPU and a mixed SCSI-2/Ultra-SCSI hard-disk system) were used for the timing tests. All the Sun workstations were running Sun OS 5.5.1 (Solaris 2.6). The final systems used were a Digital Alpha workstation running Digital Unix 4.0D with two 600-Mhz CPUs, 1500Mb RAM and an Ultra-SCSI RAID hard disk array, and a Compaq DS20 server running Digital Unix 4.0E.

The tests were run using both ACS WFC data and ACS HRC data, processing both single images and associations of CR-SPLIT=2 observations. All the data and reference files were located on the same disk local to the CPU, with *CALACS* being run as the sole active process. The results can be found in Table 8. The CPU percentages given in that table refer to the percentage of time the CPU was running during processing, with no CPU activity during disk I/O. The Sparc 4 ran into difficulty processing the large WFC observations, requiring paging to the disks, resulting in the CPU remaining idle more during processing than on the Ultra-10. This demonstrates the importance of having enough memory to run *CALACS*. In contrast, the run-times for *CALACS* on the Ultra-10 and Alpha seem to scale somewhat with CPU speed, indicating that although important, disk I/O doesn't limit the processing times when there is enough space in memory to avoid paging. In short, all the design decisions made for *CALACS* are supported by the overall efficiency demonstrated by these run-time tests.

Table 8. Run-times for *CALACS*

System	Single HRC		HRC Association		Single WFC		WFC Association	
	Time	CPU %	Time	CPU%	Time	CPU %	Time	CPU %
Sparc 4	50s	84	3m 33s	68	35m 16s	37	86m 39s	34
Ultra 10	19s	67	42s	89	4m 34s	79	12m 54s	71
Ultra 60	20	60	46	58	4m 07s	56	13m 32s	70
Alpha	16s	45	26s	76	3m 23s	41	5m 49s	74
DS20	17s	56	31s	70	3m 12s	53	7m 45s	74

Regardless of the amount of memory or speed of the I/O, it is fairly obvious that in order to efficiently process ACS data, a fast workstation will be necessary, with at least 256Mb of memory being a preferred to avoid excessive disk swapping which can dramatically slow down processing.

CALACS Hard-disk Usage

Another requirement for running *CALACS* that needs to be considered is the amount of storage needed for ACS data and its calibrated products. While the size of the final calibrated HRC or SBC exposures are comparable to those of STIS or WFPC2, the ACS WFC exposures are over 16 times as large. In addition, the following equation should be used to estimate the amount of free storage that should be available during processing of associated ACS data:

$$D_{max} = (1 + n) \cdot S_{final} + n \cdot S_{raw}$$

where:

- D_{max} is the maximum free disk space needed for processing,
- S_{final} is the size of the calibrated exposure (from Table 5),
- S_{raw} is the size of the raw exposure (from Table 4), and
- n is the number of exposures in each CR-SPLIT set or Repeat-Obs set.

7. Summary

The implementation of *CALACS* has resulted in a set of tasks which work with very large datasets in a memory-conserving manner without sacrificing too much speed. The design decisions made to conserve the amount of memory used during processing were justified by the efficient run-times for *CALACS*. Careful considerations, however, must still be made when deciding what systems *CALACS* will be used on due to the large ACS datasets. In general, though, current platforms seem able to provide sufficient fast processing times for most applications. Furthermore, the development of *CALACS* in C will facilitate quick revisions and debugging as needed in future support of the package, complete with regression tests described in the ISR by M. Mutchler.