Data Archive System (DAS)
User Guide

Version 2.2.2

For use with DTS

HEASARC
Laboratory for High Energy Astrophysics
NASA/GSFC, Code 662

Sep 24 2021

Prepared by:
Micah Johnson, Jesse Allen & Lorella Angelini (HEASARC/NASA)

Table of Contents

CHANGE RECORD PAGE (1 of 1)

DOCUMENT TITLE : DTS Users Guide			
ISSUE	DATE	PAGES AFFECTED	DESCRIPTION
Version 2.2.2	Sep 2021	All	Changes to aligned with 2.2.1. Imported from LaTeX (JSA)

1 Introduction

This document describes the installation and operation of the Data Archive System (DAS).

If questions arise concerning the usage of the DAS, contact dtshelp@heasarc.gsfc.nasa.gov.

The Data Transfer System (DTS) supports the execution of scripts based on the type given at send-time as defined in the dts.scripts file. DAS is one such script which takes the staging directory and the transferred files as arguments, and then copies them into a destination directory, preserving directory structure and ensuring that file integrity is preserved. All activities of DAS are logged in detail and when a fatal error occurs multiple responsible parties can be notified via email.

```
Syntax: das.pl [OPTIONS] staging_dir file1 file2 file3 ...
```

Generally, this script is not run directly, but rather spawned by a `dts -get` run. The intended usage is to include a line in the dts.scripts file. For example:

```
type="/home/dtsops/das/das.pl -t type"
```

2 Installation

Download the das.tar.gz package from https://heasarc.gsfc.nasa.gov/dts/.

The recommended place to install the DAS package is the home directory of the DTS operator account, however, any directory accessible by the DTS operator account will work.

If the DAS option revision checking will be used (`-r revpatt`), the Astro::FITS::CFITSIO perl module must be installed systemwide or in a directory defined in the PERLLIB environmental variable.

If the command `perl -e "use Astro::FITS::CFITSIO"` returns nothing, the module is already installed, however, if it returns a `"Can't locate"` error, the CFITSIO library and the Perl wrapper must be downloaded and built. The CFITSIO package is available from https://heasarc.gsfc.nasa.gov/fitsio/ and the Perl CFITSIO wrapper from http://hea-www.harvard.edu/~rpete/cfitsio/. Then, the Perl module must be downloaded and linked to the CFITSIO library.

Edit das.config to reflect the specifics of your site.

3 Options

version (`-v`)
Prints the DAS version and exit.

destdir (`-d destdir`)
Overrides destination directory for current run. Default is defined as DESTDIR in das.config.

bakdir (`-b bakdir`)
Overrides backup directory for current run. Default is defined as BAKDIR in das.config.

type (`-t type`)
Sets type used to spawn script. If any scripts must be spawned from DAS, this option must be set. The type name from dts.scripts in DTS that spawns DAS and the type in das.scripts should match.

partition (`-p`)
Use file set as PARTITION in das.config to partition ids into directories.

filepart (`-f partition.file`)
Overrides partition file for the current run. Default is defined with PARTITION in das.config.

expatt (`-e expatt`)
Exclude files matching this pattern from archive. The value given for **expatt** is interpreted as a Perl regular expression. If the current type is configured in the das.exscripts file, the assigned script will be executed with the staging directory followed by the excluded files as arguments.

keylist (`-k key1,key2`)
Overrides revision keywords for current run. Default is defined with REVKEYS in das.config.

revpatt (`-r revpatt`)
Get revision info for files matching this pattern. The value given for **revpatt** must be a regular expression compatible with Perl. On the command line, single quotes are recommended to avoid unwanted substitutions. For example, `-r 'sw[0-9]+g200170h\.evt'` will match files with "sw" followed by one or more digits, then "g200170h.evt". All files matching the pattern will have their revision keys checked as defined in REVKEYS in das.config, however, only the revision for the first matching file will be recorded. It is recommended that only one file match, as multiple keyword checks especially from large compressed files will slow down the archive process. When using this option any revision problem will cause the archive process to be aborted. The **revpatt** option is incompatible with **allchkrev**.

allchkrev (`-a`)
Compares the revision of all files based on REVKEYS in das.config, accepting or rejecting them on a case-by-case basis. If the incoming file's revision is newer, it is archived and the file it is replacing is stored in the BAKDIR directory. If the incoming file's revision is older or equivalent,

the existing version in the archive is left untouched and the incoming file is placed directly into the BAKDIR directory. The **allchk** option is incompatible with **revpatt**.

inventory (`-i`)
Records entry in inventory file. If this option is set, a file in the LOGDIR directory is written. The filename for the log is of the form `[INVROOT]_[type].log`, where "INVROOT" is the setting in the config file and "type" corresponds to the current type set with the -t flag. An entry is appended to the inventory file containing a timestamp, data unit, revision keys, and location for each data unit archived and if a directory is replaced, the details of the outgoing backed up directory is also recorded. If the data can be defined as a unit (for example, all the data is in a directory named for an observation number), the directory containing all the data will be recorded in the inventory file. If, however, the data consists of multiple files independent from each other, each file will have a separate entry in the inventory file.

config (`-c config`)
Sets location of das.config. The standard way to set the location of the das.config file is through the environmental variable, DAS_CONFIG, however, it may also be set with this option.

xdebug (`-x | -x chatlev`)
Sets debug level for terminal messages. If this option is set without a value, a chat level of 20 is assumed. The default terminal chat level is set with TERMCHAT in das.config, usually 10. With a larger chat level, more information is printed.

subdir (`-s subdir`)
This option allows an extra level of directory structure to be inserted between the file paths transmitted from DTS and the setting for the destination directory.

As of DAS 2.1, the subdir option may contain an expression of the form [FITSKEY] or [DATEKEY:dateformat], which is replaced based on the value for a given FITS keyword. The value of the keyword DATEKEY must be of the form "yyyy-mm-ddThh:mm:ss", while the date format is used to define how to use the date information. The string 'yyyy' is replaced with the four-digit year, 'mm' becomes the two-digit month, 'dd' becomes the two-digit day, and 'qq' is replaced by 'q' followed by the number of the quarter. For example, suppose DAS is given the option '`-s data/[DATE:yyyyqq]/obs`' and DATE is defined as '2004-11-08T11:02:15'. The data will be written to a subdirectory called 'data/2004q4/obs'. If the **revpatt** option is set, the FITS keyword will be read from the first file matching that pattern. If the **allchkrev** option is given, every file will be checked for the keyword and the subdir will be expanded accordingly. If neither is set, the first file will be used.

noclean (`-n`)
This option disables the cleanup of the incoming staging area, which can be useful if the DAS is run multiple times with the same data. For example if a mirror is to be made on another disk, one may run "`das.pl -n`" the first time and omit the "`-n`" the second time for proper cleanup.

mode remove (`-m REMOVE`)
With no mode option set, the default mode takes all the files and archives them to the destination directory. With mode=REMOVE, DAS instead expects a single file which contains a list of files to be removed from the archive.

mode nobackup (`-m NOBACKUP`)
As of DAS 2.2.2, an addition mode option, NOBACKUP, is supported. In typical operations, files brought into the system are backed up with a root directory specified either by BAKDIR in the configuration or with the -b option. NOBACKUP overrides this to suppress backup: this can be useful for reprocessing an existing dataset or handling information not destined for a public-facing archive.

4 Configuration File

The das.config file holds specifications on how the DAS is to behave. The location of the das.config is given either by assigning the -c flag or the DAS_CONFIG environmental variable. The configuration file contains the following variables:

```
LOGDIR - Location of logs
DESTDIR - Location of destination archive
BAKDIR - Location of archive backups
PARTITION - Path to partition file
PAUSEFILE - Temporarily halt archiving
INVROOT - Inventory root name: LOGDIR/INVROOT_type.log
INVBAKDIR - Inventory backup directory
MAILERR - List of emails to send errors
LOGCHAT - Log chat level
TERMCHAT- Terminal chat level
MAILCHAT- Chat level to send error email
USEIDDIR- Whether to use id from DTS as a directory
LOCK_RETRY_TIMES - Number of times to check for lock
LOCK_RETRY_DELAY - Time to delay between checks for lock (in
seconds)
REVKEYS - Revision keys to check
REPLACE_EQUAL - If TRUE, replace archive data with incoming of
equal rev
SCRIPTS - Location of das.scripts file
EXSCRIPTS - Location of das.exscripts file (see -expatt option)
UNZIPEXT- File types to uncompress in archive
CLEAN- Whether to clean incoming staging area
```

4.1 Directory Locations

There are three types of log files in the DAS, ".in" files which record the arguments given as input to the DAS, ".log" files which record the details of the DAS run, and an optional inventory file enabled by the -i option which records summary information. All log files are written to the directory defined as LOGDIR. The ".log" and ".in" files are named with the pattern: `das[timestamp]_[pid].[ext]` where "timestamp" has the form YYMMDDHHMMSS, "pid" is the current process id padded with zeroes to maintain a consistent filename length and "ext" is `log` or `in`. The inventory filename is determined by the INVROOT setting.

```
#
# Directory to write log files to
LOGDIR="/home/dtsops/das/logs"
```

The DESTDIR variable sets the location where the incoming files are archived. If files are to be replaced, they are backed up to BAKDIR inside a unique timestamped directory of the form `das[timestamp]_[pid]`.

```
#
# Top directory to archive files to (may be overridden with -d
flag)
DESTDIR="/home/dtsops/dest"
#
# Directory to backup replaced files (may be overridden with -b
flag)
BAKDIR="/home/dtsops/bak"
```

4.2 Partitioning

```
# File defines id partitioning
PARTITION="~/das/das.partition"
```

Due to disk space and directory navigation slow down issues, it is often desirable to divide the contents of a directory with many files into subdirectories. Archived data often has a sequence id associated with it that can be partitioned numerically. For example, the directory 10000000000 contains all ids greater than or equal to 10000000000 and ids less than 20000000000. Note: this sequence id corresponds to the top directory common to all files or the numeric portion on the sent -id value if USEIDDIR=TRUE. The file defined as PARTITION contains one line for each directory and comments are begun with a '#' :

```
#
# Partition sequence ids
#
unclassified
00000000000
10000000000
20000000000
30000000000
40000000000
50000000000
60000000000
70000000000
80000000000
90000000000
```

In this example if a unit of data being archived is not numeric, it will fall in the 'unclassified' directory. If the first line is numeric, the non-numeric data will not be partitioned. The data will be archived alongside the partition directories (e.g. 00000000000). The -p flag must be set when running DAS to enable partitioning.

4.3 Pausing

The ".in" file contains the DAS command as it was called with all its original arguments. The file may be run with the shell's "source" command to re-run the DAS command if whatever originally caused it to fail has been fixed.

When a long-term error occurs which may affect multiple DAS runs, such as an archive disk failure, the DAS has the ability to continue processing archive requests from the DTS without actually moving the files into the archive. Rather than going through multiple error emails and running the indicated files, a pause file can be created which collects each incoming DAS command in a single file which may be sourced when the archive is back to normal.

```
# If pause file exists in LOGDIR, only record das input and exit
PAUSEFILE="daspause.txt"
```

To pause the DAS, create the PAUSEFILE in the LOGDIR. For example:

```
cd /home/dtsops/das/logs
touch daspause.txt
```

To resume processing and run all the waiting DAS commands:

```
mv daspause.txt dasresume.txt
source dasresume.txt
```

4.4 Logging

The optional inventory file, enabled by the -i option, keeps a running record of the data being archived and the backup location if data is being replaced. Each type has a separate inventory file.

```
# Root used to name inventory log: LOGDIR/INVROOT_type.log
INVROOT="dasinv"
```

For the setting above, suppose that the type being inventoried is "`exdata`". Then, the inventory file is written to LOGDIR, named `dasinv_exdata.log`.

The inventory file keeps a running record and is a common file from run to run, so it is easy to see that the file is vulnerable to corruption. The backup mechanism for this file creates a separate file of the form `[INVROOT]_[type]_das[timestamp]_[pid].log` for each DAS run in INVBAKDIR. If anything happens to the main file, a simple cat command with a particular type can be used to reconstruct the file. In our example, the command would be `cat dasinv_exdata_das*.log > dasinv_exdata.log`.

```
#
# Directory to back up inventory log before writing to
```

```
INVBAKDIR="/heaswift/FTP/swift/bak/inv"
```

The amount of information that is recorded in the ".log" file when the DAS is running is configurable. In the script all print statements are assigned a chat level. With higher chat level, more information is printed. The recommended setting is 10 for the terminal and 20 for the log to record more detailed info. If an error is reported, usually at level 5, and the MAILCHAT setting is greater than or equal to its level an email will be sent with the error to the recipients defined in the MAILERR setting. Multiple email addresses must be separated by commas. Setting MAILCHAT to 0 turns off this error mailing feature.

```
#
# List of addresses to send errors to (comma-delimited)
MAILERR="user1@host1,user2@host2"
#
# Chat level at which to write to log file
LOGCHAT=20
#
# Chat level at which to write to terminal
TERMCHAT=10
#
# Chat level at which to email errors (Note: not warnings or
info)
MAILCHAT=5
```

4.5 Copy Details

The DAS reconstructs the directory structure as given to the original DTS send command when copying the files from the staging directory. For example, if the DTS command is of the following form:

```
dts -send DEST -type data -l dir1/type1/file.fits
dir2/type2/file.gif
```

Then, the DAS will copy the files to the designated DESTDIR, creating the necessary directories to preserve the directory structure and verifying the file integrity with a checksum. So, in the example the following new files would appear:

```
DESTDIR/dir1/type1/file.fits
DESTDIR/dir2/type2/file.gif
```

If they do not already exist, the following directories would be created:

```
DESTDIR/dir1
DESTDIR/dir1/type1
DESTDIR/dir2
DESTDIR/dir2/type2
```

If a file already exists in that location with the same name, it will be moved to the BAKDIR area with the same directory structure under a timestamped directory. If an error occurs at any point in the DAS run, the original destination area will be restored from these backup files, and any new files or directories will be removed.

There is another option which allows the passing of the topmost id directory as part of the staging directory name.

```
# If TRUE, uses staging directory to create a directory
# corresponding to the dtsid enclosing the files
# (format: dtsid_timestamp , only "dtsid" portion used as
# directory)
USEIDDIR="TRUE"
```

The DTS staging directory given as the first argument to the DAS contains the id. By default, the directory has the form dtsYYMMDD_HHMMSSTYPESITE, however, in DTS send mode, a alternative id may be specified.

The USEIDDIR option uses this id, archiving the data in a directory in the destination directory, which we will to refer to as a unit directory. For example,

```
dts -send DEST -type data -id seqid.YYMMDD_HHMMSS -l
type1/file.fitstype2/file.gif
```

The id option is used to pass along the sequence id and the following files are copied. Any text after the last '.' in the id will be stripped away and non-numeric characters will be ignored.

```
DESTDIR/seqid/type1/file.fits
DESTDIR/seqid/type2/file.gif
```

Note, if no id is specified in the DTS command, the staging directory has the form dtsYYMMDD_HHMMSSTYPESITE. DAS will assume that such a directory starting with "dts" is not intended as an id directory, effectively behaving as if USEIDDIR=FALSE.

If USEIDDIR is FALSE, it is recommended that all the files transferred by DTS have a common directory at the top of their paths. This allows the archiving of the files to be done as a unit, preventing a mixture of old and new files that is generally not desired. If the data can be classified as a unit and the directory exists in the archive, the entire directory will be moved to the backup area before copying files. Otherwise, individual files are backed up only if they conflict with an incoming file. Also, the inventory (-i) option will fail if the DTS transfer cannot be classified as a single unit.

```
# Revision keywords (comma-delimited)
REVKEYS="PROCVER,SEQPNUM"
```

If the **revpatt** (-r) option is set, the files matching that pattern will be checked for the given keywords in order. Supported revision keys include dates of the form "yyyy-mm-ddThh:mm:ss", versions of the form "X.Y.Z" and standard floating point numbers or integers. If the incoming data is a later revision, the archive data is moved into the backup area and is replaced by the incoming data. If the archive has a later or equivalent set of keywords, the incoming data is rejected and moved into the backup area.

For example, if the archive has PROCVER=8.0.0,SEQPNUM=001 and the incoming files are PROCVER=7.8.1,SEQPNUM=003 the incoming data is rejected. If the revisions are reversed the incoming data replaces the data in the archive.

Note, the default behavior is to reject incoming versions that are earlier than or equivalent to the archive data. In order to restrict this behavior to reject only earlier revision incoming data and allow equivalent revision data to replace the data in the archive, set the following configuration option.

```
REPLACE_EQUAL="TRUE"
```

4.6 Post-Processing

Like DTS, DAS will spawn a script based on the given type. NOTE: DTS does not automatically pass the type to the scripts it is spawning, so the type must be passed with the -t flag in the dts.scripts file. For example, in dts.scripts: `data="/home/dtsops/das/das.pl -t data"`

```
# File assigning types to scripts
SCRIPTS="/home/dtsops/das/das.scripts"
```

The format of the das.scripts file is like dts.scripts:

```
type="/home/dtsops/das/scripts/example.pl"
```

Each script receives the arguments:

```
-t type -b curbakdir dir file1 file2 ... fileN
```

The value for type comes from value given to the type flag upon execution of the DAS. The value for `curbakdir` is the full path to a timestamped directory in BAKDIR where directories and files that have been replaced in the current run are kept. If no files have been replaced no -b option will be passed.

See example.pl provided with the DAS distribution for the recommended implementation of DAS-spawned scripts. All standard output of the spawned script is recorded in the DAS logs. If a serious error occurs in the spawned script, the recommended implementation is to use the exit command

with a non-zero value. The exit status will trigger an error in das.pl which will restore the archive to its original condition and send an email notification.

The UNZIPEXT variable contains a comma-delimited list of the extensions which should be gunzipped in the archive. In general, graphics files have their own compression and need to be uncompressed in order for browsers to properly display them, so this feature allows them to be archived in the proper state. Use UNZIPEXT="" to turn this off.

```
#
# List of file extensions to automatically uncompress before
archiving
# (comma-delimited) For example "gif,ps" would match files
# =~ /\.(gif|ps)\.gz$/
UNZIPEXT="gif,ps"
```

If the staging directory and its contents are to be removed after the archiving operation has been successfully completed, set CLEAN="TRUE" otherwise set it to "FALSE".

```
#
# If TRUE, clean up files in the staging area after a successful
run
#
CLEAN="TRUE"
```

In some usages of DAS, there has been a call for the ability to strip out special files in the data set, and execute a script on the excluded files after the data has been archived. The -expatt option and the das.exscripts configuration file combine to provide this functionality.

For example if an individual must be informed that the data set has been archived, a special file can be included in the data, which contains the person's contact information. Suppose the file is called notification.txt and it contains an email address. The standard DAS argument list is given the option '-e notification', indicating that any file name containing the string 'notification' must be excluded. In this example, a script which parses the notification file and sends an email is developed and its path is given as follows in the das.exscripts file:

```
exampletype="/home/dtsops/das/scripts/notification.pl"
```

Each script for excluded files receives the arguments:

```
-t type stgdir file1 file2 ... fileN
```

The original staging directory of the data is given as the excluded files are not copied with the rest of the data. If a copy of the excluded file must be kept, the spawned script must perform the copy. DAS will exclude the file from the archive and clean it from the staging area after the archive process is complete.

4.7 Examples of post-processing at the HEASARC

DAS passes the files recently archived and any script may be spawned with those arguments, allowing further processing, verification, *etc.* For example, the archive directory and the backup directory from the script arguments may be used to call rsync and maintain an incrementally updated mirror of the main archive (See scripts/dplrsync.pl in the DAS distribution).

Also in the example script below, the DAS-spawned script copies the database table to a working directory and then sends an email to an address which is being monitored with procmail. Based on the message, a script is executed which ingests the table.

```
#!/usr/bin/perl
#
# Copy dbase file to holding area adding timestamp to name,
# then send email trigger
#
# Usage:
#
# dbnotify.pl -t type -b bakdir dir files
#
require 5.005;
use strict;
use File::Basename;
use Getopt::Long;
use File::Spec;
use File::Copy;
use Mail::Send;

#
# Local configuration
#
# Email address with procmail monitoring
#
my($email) = "procmon\@host.lab";
#
# Procmail triggered script sees file at
# $dbhold/$dbase/incoming/${dbase}-$datestring.data
# (File is renamed with timestamp to avoid overwrites)
#
my($dbhold) = "/local/swift/db";
#
# Options
#
my($type) = "";
my($dbase) = "";
my($bakdir) = "";
```

```
GetOptions( "dbase=s"=> \$dbase,
"bakdir=s" => \$bakdir,
"type=s" => \$type );
if ( $type eq "" ) { $type = "dbnotify"; }
if ( $dbase eq "" ) { $dbase = $type; }

my($dir,@files) = @ARGV;
my($ofile,$nfile);
my($FILE);
my($datestring)=`date +%Y%m%d%H%M%S`; # timestamp
chomp $datestring;

$| = 1; # Command buffering on, otherwise Mail spawn repeats
stdout

if ( !$dir ) {
print "ERROR in dbnotify: No directory given\n";
exit(2);
}
#if ( !@files ) {
#print "ERROR in dbnotify: No files given\n";
#exit(3);
#}
if ( @files != 1 ) {
print "ERROR in dbnotify: Exactly one file must be given\n";
exit(3);
}

# Copy files to holding directory with timestamp to avoid
overwrites
foreach $FILE (@files) {
$ofile = File::Spec->catfile($dir, $FILE);
$nfile = File::Spec->catfile($dbhold, $dbase, "incoming",
 "${dbase}-$datestring.data");
print "SCRIPT dbnotify: Copy $ofile to $nfile\n";
if ( -e $nfile ) {
print "ERROR in dbnotify: File already exists\n";
exit(4);
}
if ( !copy($ofile, $nfile) ) {
print "ERROR in dbnotify: Copy failed\n";
exit(5);
}
}

#
```

```
# Trigger dbase ingest
#
my($msg) = new Mail::Send;
$msg->to($email);
$msg->subject($dbase);
my($fh) = $msg->open;
print $fh "$nfile\n";
$fh->close;
```

5 Appendix A: DTS AUTORUN Recovery (runtype.pl)

The standard usage of the das.pl script is to associate it with a type in the dts.scripts file of DTS. The sending site must use that type (`-t TYPE`) when sending the data, then DAS is automatically run on the transferred data. Unfortunately, this ideal situation may not always exist. The sending site may fail to use the correct type flag or the receiving site may be improperly configured. When DTS properly transfers the data, the operation is considered a success and an acknowledgement message is sent to the sending site regardless of whether or not the script associated with its type runs successfully.

For example, if a bad path is given for a type in the dts.scripts file, the files will transfer but an error email will be sent to the DTS operator. One option is to correct the path, then perform a "dts -recover" on the appropriate mail file. The files, however, have transferred correctly and an ACK message has been sent to the sending site. Consequently, the files may have already been cleaned at the sending site. The runtype.pl script has been developed for such situations. Like the recover operation this script takes a mail file as input, but unlike recovery no file transfer is needed. The files are locally available in a staging area, so runtype.pl can execute any script on those files, picking up where a bad type or misconfiguration halted the process.

Usage:

```
runtype.pl [-t TYPE | -s script] [-i 1,3,4 | -a] [file.msg |
file.mail]
```

The runtype.pl script must be given a .mail or .msg file as an argument. A .mail file may contain multiple SEND messages, so inspection of the file may be necessary to determine the relevant message. The runtype.pl script can also handle .msg files which contain a single SEND message along with a log of the original send operation.

In the bad path example, after correcting the dts.scripts file the following would execute the script associated with the Proddata type, rerunning the failed operation:

```
runtype.pl -t Proddata dts021017_111004SProddataHEASARC.msg
```

script (`-s scriptpath`)
Sets the script to run with the arguments parsed from the mail file. Note, script flags may also be included if properly quoted on the command line (*e.g.* `-s "$\sim$/das/das.pl -i -t Proddata"`)

type (`-t type`)
Instead of setting the script directly with the -s option, a type may be specified. Using the DTS_CONFIG environmental variable, the dts.scripts file will be found and parsed. The script associated with the given type will be run with the arguments parsed from the mail file.

ilist (`-i 1,3`)
By default, only the first SEND message found in a given mail file will be run. If a .mail file is given, the -i option may be used to specify which messages with the file to run. The first message in the file is 1, second is 2, and so on. All message indexes in the comma-delimited list will be run unless an error occurs. If an error is encountered, the script stops immediately leaving the remaining message indexes in the list unused.

all (`-a`)
By default, only the first SEND message found in a given mail file will be run. If a .mail file is given, the -a option may be used to indicate that all messages in the file are to be run.

Note, the DTS_CONFIG environmental variable must be set so that the incoming staging area can be found by runtype.pl.

6 Appendix B: Acknowledgements

The DAS development team acknowledge the contribution of Micah Johnson to the original development of the DAS code, plus contributions since by Joe Eggen.