Rebuilding the Hubble Space Telescope Exposure Time Calculators

Perry Greenfield
Ivo Busko
Vicki Laidler (CSC/STScI)
Todd Miller
Mark Sienkiewicz
Megan Sosey

Space Telescope Science Institute
Outline

• What is an Exposure Time Calculator and why you would need one
• History of HST ETCs
• What went wrong
• Rebuilding
• Sociology of scientific programming (huh?)
Thou shall not waste HST time!

It’s expensive: $100,000 per orbit

- HST proposals are judged on their practicality as well as scientific merit.
  - Can you see what you say you hope to see?
  - Are you asking for too much or too little time?
  - How do you demonstrate that?

With an Exposure Time Calculator, of course
What does an ETC do?

Basically:

1) Determine exposure time needed to achieve a specified Signal-to-Noise Ratio, or
2) Determine the SNR for a specified exposure time.

Doing that requires:

1) Knowing the details of what you expect to see.
2) Simulating how the telescope and instrument work.
Details, details, details

There is a lot to specify:

• Spectrum of source
  – lots of choices possible
• Brightness and geometry of source
  – simple isn’t it?--no
• Dust extinction, redshift…
• Expected background signal
• How the signal will be extracted
• Instrument configuration
History

• Originally not even planned
• Grassroots versions developed by instrument scientists
  – Despite the fact that managers said it wasn’t needed
• Instrument scientists get tired of supporting software, give it to someone else
• Clue here…
• Big new initiative for a grand unified proposal tool
  – Written in Java, GUI-based
  – Decade-long project
  – Worked well (or seemed to) until 2008
    • And then the trouble started
What went wrong?

• Despite origin, tool ended up as a web application.
• That became very unreliable
• Crashing every 5 minutes under heavy load near proposal deadlines
• This led to many unhappy astronomers
Why?

- Effort to improve reliability failed
  - Actually got worse!
- Analysis of system revealed many weaknesses
  - Too complex
  - Building/installing painful (manual process, context dependencies)
  - Many testing inconsistencies
    - Hard to run
    - Many different test formats and schemes
    - Test environment did not duplicate production environment
      - Tests could pass on test machine and fail in production
    - Results hard to analyze
  - Misuse of database
    - Contained information it shouldn’t have
    - Embedded, locked to one process
Why? (cont.)

• Weakness (cont.)
  – ETC algorithm represented by confusing XML.
  – Inadequate handling of program exceptions.
  – Inadequate testing of javascript features
  – Instrument parameters located in a multitude of locations.
  – Too many key responsibilities placed on one individual
  – The ultimate killer: modification/install/test cycle was at a minimum 2 weeks, often a month or more.

How this happened? Later…
Rebuilding

Since the problems were pervasive and touched all areas of the existing project, a complete rewrite was in order.

• In Python, of course
• Project began in June 2009
• Goal was to be ready for production in Dec 2010.
• Minimize changes to user interface
• Use existing tests and results to validate new version
Design goals

• One step install
• Support multiple installations on same computer
• Nightly regression testing
• Separation of web and compute functionality
• Ability to script calculations without web server
• Use standard Apache/Database server schemes to handle failover and load balancing
• Simple database structure
• Concentrate instrument info in one place
• Automatic test generation for better parameter space coverage
• No XML
• No cached results
Tools Used

- pysynphot (photometric simulations)
- pyfits (FITS file I/O)
- Pandokia (test reporting framework)
- mechanize (running old ETC through web)
- numpy
- matplotlib
- Django
- nose
Status

- All basic elements coded (web and compute)
- Currently refactoring compute code to simplify
- Working through remaining test discrepancies (primarily for one instrument)
- Tests all converted (over 8000 tests)
- Was a major effort to convert into a consistent, correct form
- Now working on generating test cases to span parameter space more thoroughly
- 130K LOC (Java) --> 27K LOC (Python)
- Will be used for James Webb Space Telescope
- Compute engine will be open source (pyetc)
ETC Request ID: ACS J1m.000312
DEBUG MODE: DICT

Requested Signal/Noise Ratio = 10.000

Time to Saturation (for a single exposure) = 21.39 seconds
Optimum SNR = 19.0478

Exposure time calculation: **WARNING**

**WARNING MESSAGE:** Result of the calculation is less than the minimum exposure time for this detector: value = 0.00307088, limit = 0.500000

### Detailed Information

<table>
<thead>
<tr>
<th>Counts (circle with radius 0.165 arcsec)</th>
<th>Count rate (e^-s)</th>
<th>Total counts (e^-)</th>
<th>Associated noise (e^-)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>14,801.604</td>
<td>454.539</td>
<td>21.320</td>
</tr>
<tr>
<td>Background</td>
<td>2,099</td>
<td>0.064</td>
<td>0.254</td>
</tr>
<tr>
<td>Sky</td>
<td>1,882</td>
<td>0.03779</td>
<td>0.2404</td>
</tr>
<tr>
<td>Dark Current</td>
<td>0.218</td>
<td>0.00668</td>
<td>0.0817</td>
</tr>
<tr>
<td>Read out</td>
<td></td>
<td>-40.14</td>
<td></td>
</tr>
<tr>
<td>Total in selected region</td>
<td>14,803.704</td>
<td>454.604</td>
<td>45.454</td>
</tr>
<tr>
<td>Brightest Pixel</td>
<td>3,959.042</td>
<td>60.789</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Count rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source (arcsec^-2)</td>
</tr>
<tr>
<td>Sky Background (infinitely large region)</td>
</tr>
</tbody>
</table>

**Extraction region**

| Fraction of flux                          | 0.890             |
| Sky Extraction Area (pixels)              | 34,971            |
| Fraction of flux in brightest pixel       | 0.214             |
| Effective Wavelength                      | 5397.58 Å         |

**Target (pysynphot target expression)**

Renormalized to r(unit(1,flam).band(johnson,v),15.000000,vegamag)

**Instrument name:** ACS

<table>
<thead>
<tr>
<th>Mode</th>
<th>Imaging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coronography</td>
<td>False</td>
</tr>
<tr>
<td>Detector</td>
<td>WF1</td>
</tr>
<tr>
<td>Bandpass</td>
<td>F555W</td>
</tr>
<tr>
<td>Gain</td>
<td>2.0</td>
</tr>
<tr>
<td>CR Split</td>
<td>2</td>
</tr>
</tbody>
</table>

**Selected background:**

<table>
<thead>
<tr>
<th>Sky Background</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Earth Shine</td>
<td>Average</td>
</tr>
<tr>
<td>Zodiacal Light</td>
<td>Average</td>
</tr>
<tr>
<td>Air Glow</td>
<td>Average</td>
</tr>
</tbody>
</table>

**Plots**

[Source spectrum] [Throughput]

These results were computed using: ETC version 06-01
Throughput for instrument configuration: ACS,WFC1,F555W

Wavelength (Å) vs. Throughput

Log

Axis Min

Axis Max

Plot size

Display entire wavelength range

Plots

Source spectrum

Throughput
Sociology

- Original problem had its roots in astronomer/software developer expectation mismatch.
- Drove feature-oriented schedule over the needs for good software engineering
### Astronomer vs. Developer

<table>
<thead>
<tr>
<th>Astronomer</th>
<th>Developer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad-hoc changes to handle various needs</td>
<td>One code base to handle all needed alternatives</td>
</tr>
<tr>
<td>Corner cases often ignored</td>
<td>Special cases given more attention</td>
</tr>
<tr>
<td>Little attention to user interface</td>
<td>Much more attention to user interface</td>
</tr>
<tr>
<td>Minimal error checking</td>
<td>Extensive error checking</td>
</tr>
<tr>
<td>No version control</td>
<td>Version Control</td>
</tr>
<tr>
<td>No unit or regression tests</td>
<td>Extensive tests</td>
</tr>
<tr>
<td>Minimal documentation</td>
<td>More extensive documentation</td>
</tr>
<tr>
<td>Refactoring rare</td>
<td>Hopefully not…</td>
</tr>
</tbody>
</table>