

Automating the Generation of the Cassini Tour Atlas Database

Kevin R. Grazier¹, Chris Roumeliotis², and Robert D. Lange²
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109

The Tour Atlas is a large database of geometrical tables, plots, and graphics used by Cassini science planning engineers and scientists primarily for science observation planning. Over time, as the contents of the Tour Atlas grew, the amount of time it took to recreate the Tour Atlas similarly grew—to the point that it took one person a week of effort. When Cassini tour designers estimated that they were going to create approximately 30 candidate Extended Mission trajectories—which needed to be analyzed for science return in a short amount of time—it became a necessity to automate. We report on the automation methodology that reduced the amount of time it took one person to (re)generate a Tour Atlas from a week to, literally, one UNIX command.

I. Introduction

During the Cassini spacecraft's cruise phase and nominal mission the Cassini Science Planning Team developed and maintained an online web-based database of geometric and event timing information called the Cassini Tour Atlas. The Tour Atlas consisted of several hundreds of megabytes of HTML display code, EVENTS program outputs, ASCII tables, plots, and images used by mission scientists and Cassini Science Planners for observation planning, constraint and flight rule checking, even data analysis. Over time the Tour Atlas for Cassini's Nominal Mission trajectory grew to encompass output totaling several hundreds of megabytes (currently gigabytes).

Over the span of a mission there were many reasons why navigators were required to make small changes to the Nominal Mission trajectory. Every time the mission was altered or "tweaked", a new Tour Atlas had to be regenerated manually because most mission events and science observation opportunities shifted, if albeit slightly. Although most observations were unaffected by these shifts, or the shifts fell within observation tolerances, for some classes of observations--radio science occultations being the best example--very slight shifts in event timing and position could kill the entire observation.

In the early phases of Cassini's Equinox, or Extended, Mission planning, an *a priori* estimate suggested that mission tour designers would develop approximately 30 candidate trajectories within a few weeks – a very short period of time in which to perform detailed analyses. So that Cassini scientists could properly analyze the science opportunities in each candidate tour quickly and thoroughly in order that the optimal series of orbits for science return could be selected, a separate Tour Atlas was required for each trajectory. This would have been impossible to do manually within the time allotted, so the entire task was automated using code written in five different programming languages: C shell, PERL, C, FORTRAN, and IDL. Here we discuss the contents of the Tour Atlas in general, and elaborate upon the details of the automation process in specific.

II. Tour Atlas Contents

A "tour" is the term for the Cassini spacecraft's series of orbits through the Saturn system. During its Nominal Mission, Cassini's tour consisted of 74 orbits, 45 Titan flybys, 37 icy satellite flybys, and dozens of distant flybys of these moons as well as flybys of the known smaller, or "rock" satellites. The Cassini Tour Atlas, as the name implies, was designed to be a sort of geometric road map that could reveal to scientists and science planners points in the mission where geometric requirements were met, and where scientifically interesting observations were possible. After the fact, the Tour Atlas has been used to place observations in context -- to determine the

¹ Investigation Scientist/Science Planning Engineer, Cassini/Huygens Mission to Saturn and Titan, 4800 Oak Grove Dr./MS 230-205, AIAA Non-Member.

² Science Planning Engineer, Cassini/Huygens Mission to Saturn and Titan, 4800 Oak Grove Dr./MS 230-205, AIAA Non-Member.

spacecraft's relative position to various objects within the Saturn system as an aid in data analysis. The Atlas contains information in numerous different formats, but which fall into four basic categories.

A. Petal Plots

Figure 1 is a petal plot for Cassini's Equinox Mission, also known as the Extended Mission or simply XM. Units are in Saturn radii. The innermost dotted white circle corresponds to the orbit of Titan; the outermost represents the orbit of Saturn's outermost large icy satellite Iapetus. The green square indicates the point at which the Cassini nominal mission ends, and the XM begins, whereas the red square represents the end of the XM, and beginning of the Cassini Solstice Mission (also known as the Extended Extended Mission or XXM).

Spacecraft coordinates are plotted in the rotating MPF (Mimi Planning Frame) coordinate system. The z axis corresponds to Saturn's spin axis. The cross product of the z axis and the Saturn-Sun vector yields the y axis; the cross product of z and y axes gives the x axis. The Sun, therefore, is always in the x - z plane, whereas the x - y plane is synonymous with Saturn's ring plane.

A benefit of this type of plot, over one plotted in an inertial coordinate system, is that with the Sun fixed, the Cassini spacecraft's hour angle, or phase angle, relative to Saturn is easy to approximate at any point in the trajectory.

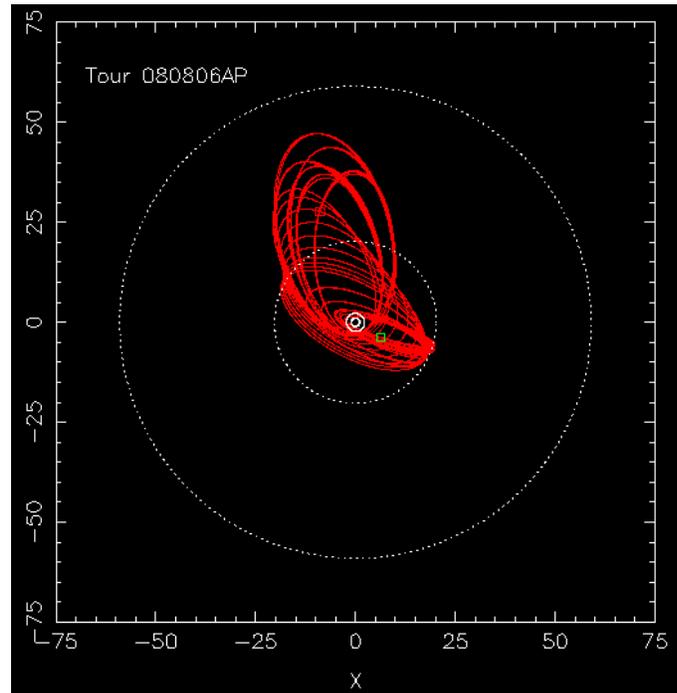


Figure 1. *XY Petal Plot for Cassini Equinox Mission. This is the series of orbits for the XM tour, looking down the spin axis of Saturn.*

B. EVENTS Outputs

The EVENTS program is a multi-mission planning and analysis software package that searches for geometrical events between a spacecraft and planetary objects like planets, satellites, rings, magnetosphere, etc. The program finds event times for both instantaneous events (e.g. apocenter and pericenter passages, satellite closest point of approach), and finite-duration events (occultations, time within a given range to a body, flux tube passages, etc.). An EVENTS output contains a header and list of event times. A typical header and a few lines of sample output appear like:

```
EVENTS FILE WRITTEN BY EVENTS  EVENTS FIELD WIDTH  40  TIME TYPE CHAR
EVENTS version 17.2: 30 10 30 30 30 30 5 7
SATURN      tour by CASSINI
KERNELS used for this output:
  /KERNELS/naif0008.tls
  /KERNELS/AllRocks.bsp
  /KERNELS/cpck21Mar2006.tpc
  /KERNELS/cpck_rock_27Feb2006.tpc
  /KERNELS/events.evk
All times given in UTC format.
  1 event(s) considered:
APOAPSIS
=====
Trajectory file: /KERNELS/PB-1_scpsc.bsp
Tour events in the interval of time from 2008 MAY 22 02:12:58.83
                                         to 2010 JUL 01 11:58:54.82

::
APOAPSIS      2008 MAY 29 09:15:50.02
APOAPSIS      2008 JUN 05 12:30:22.96
APOAPSIS      2008 JUN 12 15:44:13.99
APOAPSIS      2008 JUN 19 18:47:12.13
APOAPSIS      2008 JUN 26 20:44:48.72
```

The header contains information that can be passed to programs that post-process this output. Included in the header is a list of the kernel files that `EVENTS` used as input. Kernels are files in various JPL SPICE system formats. In the above example, the file with the `.tls` extension is a leapsecond kernel that provides information to software on how to apply leapseconds when using coordinate systems that require them. Files with `.tpc` extensions are planetary constant kernels, and contain information for planets and satellites such as radii, spin axis orientation, values of GM, etc. The file `events.evk` is in the same format, but contains values that the `EVENTS` program requires such as step sizes as well as search parameters and tolerances. Kernels files with `.spk` extensions are trajectory files for spacecraft, planets, or satellites. The double colon string `“:”` appears on its own line just before the first event, and just after the last, as a delimiter recognized by post-processing software as “bracketing” the list of geometric events and their associated times.

C. Tables of Geometric Values

The bulk of the The Tour Atlas output is ASCII-formatted tabular data. For example the spacecraft/Saturn distance, altitude, sub-spacecraft latitude/longitude, sub-solar latitude/longitude, radial and tangential speeds, and phase angle are tabulated at five minute and one hour intervals for the entire tour. Similar tables exist for Titan, as well as all the icy satellites, over the entire tour, and at five minute intervals within three days before and after closest approach for flybys.

D. Ground Tracks

The Tour Atlas software takes `EVENTS` outputs for Titan and Satellite flybys, passes that to an IDL program, and for every Titan and icy satellite flyby in the mission, this code generates several types of ground track plots. We see an example of one type in Figure 2. In this figure we see the sub-spacecraft latitude and longitude for a Titan flyby projected onto a map of that moon. The top plot is color-coded to indicate the spacecraft/Titan distance; the bottom plot coded for phase (Sun-Titan-Spacecraft) angle. The Tour Atlas also contains ground tracks plotted against polar views of the bodies, as well as 3-D plots that detail flyby geometry.

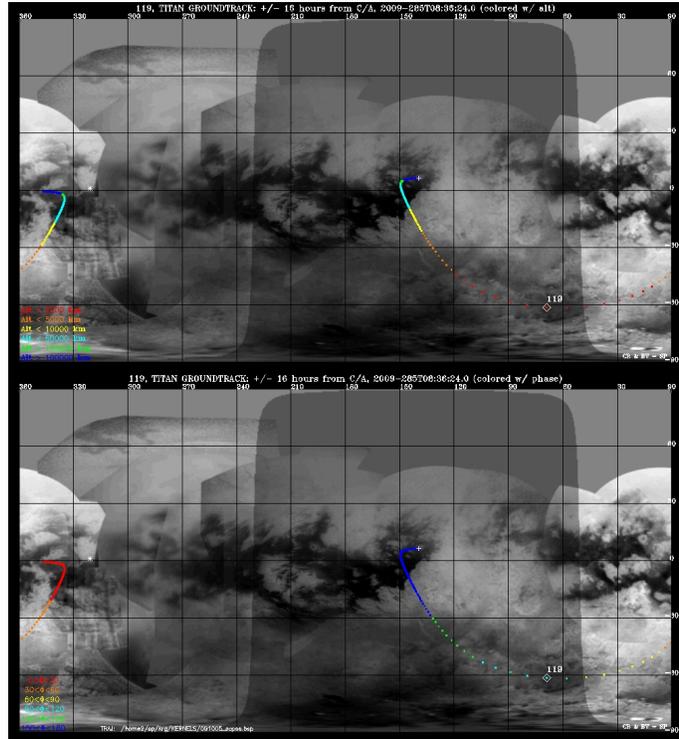


Figure 2. Titan Ground Tracks. These plots show the sub-spacecraft latitude/longitude for a Titan flyby projected against a map of the moon.

III. Tour Atlas Auto Generation

The code that automatically generates a Tour Atlas for a given trajectory is a collection of PERL scripts, C programs, FORTRAN programs, and IDL scripts bound together by a hierarchy of UNIX C-shell scripts. The code takes as its principle input a JPL trajectory, or SPK, file. The code assumes that the file names for all potential trajectory files will be of a fixed format. For Cassini’s Equinox Mission, spacecraft trajectory files had the format `XX-N_scpsc.bsp`, where the `Xs` represent letter and `N` an integer. For example a fictitious file name with the correct format, and the example we will use throughout, would be `PB-1_scpsc.bsp`. To run an atlas for a trajectory, the command line is simply `TAgen <trajectory file>`. To generate an atlas of our sample trajectory file, the user would type simply:

```
TAgen PB-1 .
```

`TAgen` is a UNIX C-shell script which calls two other scripts:

```
TAstructure $1
TAcontent $1
```

The entry “\$1” represents the first command line parameter passed to any C-shell script. In our example above, the value of \$1 would be “PB-1”. The `TAGen` script, in turn, passes the trajectory name to the two other scripts it invokes.

A. Create Directory Structure

The `TAstructure` script creates a tree of directories for data storage starting with the “root” directory, which we will call the Tour Directory. In our example, the Tour Directory would be created with the name `PB-1`. The script then makes copies of all input files, input file templates, HTML file templates, and executable files in sub directories under the Tour Directory. This is a straightforward way to run multiple versions of the code in parallel on discrete CPUs that share disk storage—processing several trajectories simultaneously.

B. Generate Tour Atlas Content

The `TAcontent` script that `TAGen` calls, in turn, invokes several other shell scripts—treating them essentially as subroutines organized by output type:

```
PETALS_content $1
TITAN_content $1
ICY_content $1
OCCULT_content $1
RINGS_content $1 .
```

Note that the trajectory file name that was passed to `TAcontent` is passed to each of the nested scripts that it calls subsequently. The key technical hurdle that we had to surmount in automatically generating a Tour Atlas largely dealt with how input files and system commands were built.

Most of the programs that generate output displayed by the Tour Atlas existed before the automatic Tour Atlas generation software task began. In order to operate within the this suite of software, we could have written slightly-modified versions of those previously-existing programs that accepted either a trajectory file name, or the unique portion of a file name, as input on the command line. In that case the scripts that generate content for the Tour Atlas would need only to pass to the value of \$1 to that program:

```
program_name $1 .
```

In our example, that would translate to:

```
program_name PB-1 .
```

While we did, in fact, use this technique in a few instances, it would be time-inefficient to modify every single program that generates Tour Atlas content in this manner. Instead we found it more efficient to apply a degree of standardization to the file names input to and output from, these programs. To this end a very short PERL script was the workhorse of the Tour Atlas automatic generation package. The following short script takes as input a file containing the string `XX-X` somewhere within, and replaces that string with a string input on the command line:

```
#!/tps/bin/perl
if ($#ARGV == 1) {$file = $ARGV[0];$tour_num = $ARGV[1];}
if ($#ARGV != 1) {print "usage: <file to have XX-X replaced> <##>\n";exit;}

@temp = split(/\./,$file);

rename $file, $temp[0].".backup";
open(IN, "< $temp[0].backup") || die "Cannot open read file: $file \n";
open(OUT, "> $file") || die "Cannot open file $file for writing \n";

while (<IN>) {
    s/XX-X/$tour_num/g;
    print OUT;
}
close(IN);
close(OUT);
```

As an example, let's take a file called `KERNELS` – a file used by numerous programs that generate Tour Atlas content – that has one modification. The input line that would normally state the trajectory file to be analyzed, the last line of the example file below, is replaced by a dummy string:

```
@    SPICE kernels.
@
LEAPSECONDS /KERNELS/naif0009.tls
SPK         /KERNELS/090507AP_RE_90165_18018.bsp
SPK         /KERNELS/090202BP_IRRE_00256_50017.bsp
PCK         /KERNELS/cpck19Sep2007.tpc
PCK         /KERNELS/cpck_rock_21Aug2008.tpc
SPK         /KERNELS/XX-X_scpse.bsp
```

Invoking the `Tour#_replacer.prl` PERL script from within one of the C-shell scripts guiding the Tour Atlas generation

```
Tour#_replacer.prl KERNELS PB-1
```

alters the last line of the file to read

```
SPK          /KERNELS/PB-1_scpse.bsp      .
```

Now every single program called by the Tour Atlas C shell scripts that look for this file in order to run, knows exactly which kernel files, in particular which Cassini trajectory kernel, to process.

One of the most important and useful elements of a Tour Atlas are outputs from the EVENTS program. Not only are these files displayed as highly-referenced parts of the Atlas, but many other programs use as input post process EVENTS outputs. The ground track IDL programs, for example, process as input twice-post-processed EVENTS outputs. For every event for which the EVENTS program searches there exists a previously-created core input file, a fragment of what will become the final EVENTS input file. A Titan flyby EVENTS file has the format:

```
TBEG   = '2008 JUL 01 00:00:00';
TEND   = '2010 JUL 01 00:00:00';
OUTNAM = Titan_Flybys.UTC.XX-X;
TSTRNG = UTC;
CBODY  = SATURN;
SC     = CASSINI;
;
NTRAJ  = 1;
TRAJECTORIES;
/KERNELS/XX-X_scpse.bsp;
;
NEV = 1;
EVENTS;
FLYBY   TITAN      100000;
```

The above input file is not a complete EVENTS input, however, in two aspects. First there are two instances of our “XX-X” string placeholder. Also, like the `KERNELS` file in the previous example, the EVENTS program utilizes an entire suite of SPICE kernels. Therefore there is also a second fragment listing the remaining required kernel files:

```
NKERNELS = 6;
KERNELS;
LEAPSECONDS /KERNELS/naif0009.tls;
SPK         /KERNELS/090507AP_RE_90165_18018.bsp;
SPK         /KERNELS/090202BP_IRRE_00256_50017.bsp;
PCK         /KERNELS/cpck19Sep2007.tpc;
PCK         /KERNELS/cpck_rock_21Aug2008.tpc;
PCK         /KERNELS/events.evk;
```

In order to run certain events additional kernels are required, some of which are quite large. It is inefficient to load huge kernels for every `EVENTS` run, so the kernels required for each run are attached on an event-by-event basis. There are therefore several kernel lists that we can attach to the core `EVENTS` input files. To perform an `EVENTS` run, then, the C-shell script first builds the input file by concatenating the two input file fragments, replaces every instance of an “XX-X” string with a valid trajectory file name, then passes that input to `EVENTS`. If the `EVENTS` core file is named `EVENTS.TITAN` and the above kernel file `KERNELS_EVENTS`, for example, the C-shell commands would be:

```
cat KERNELS_EVENTS >> EVENTS.TITAN
TOUR#_replacer.prl EVENTS.TITAN $1
events < EVENTS.TITAN
```

That would generate a list of events like:

```
::
FLYBY TITAN 100000          2008 JUL 31 02:12:56.13
FLYBY TITAN 100000          2008 NOV 03 17:34:50.97
FLYBY TITAN 100000          2008 NOV 19 15:56:01.87
FLYBY TITAN 100000          2008 DEC 05 14:25:19.42 .
```

We’ve omitted the header as an unnecessary duplication, and shown only a few events as they sufficiently demonstrate the point.

Post-processor files are also built by the C-shell scripts, but the process is normally much simpler. The `LABEL` program, for example, adds the orbit number and day of year to each line of an `EVENTS` output. It uses the `KERNELS` file we previously-created, as well as a two-line input file. That two-line file, in turn, specifies the name of an `EVENTS` file to be processed, as well as an output file name. For our Titan flyby example, the two-line file, `LABEL_TITAN` would initially look like:

```
Titan_Flybys.XX-X
Titan_Flybys.XX-XL .
```

where the “L” in the output file name means “labeled”. After being processed by the `Tour#_replacer.prl` script, our two-line file becomes:

```
Titan_Flybys.PB-1
Titan_Flybys.PB-1L .
```

Many Tour Atlas post-processor input files are similar. In this way the output file name will always be of a given format, and subsequent programs that may post-process these output files, like the ground track program, will look for a file of a known name.

Finally the C-shell commands

```
TOUR#_replacer.prl LABEL.TITAN $1
label < LABEL.TITAN
```

will create a post-processed `EVENTS` file (again header omitted) of the form:

```
Orbit  DOY
-----
 78    213  FLYBY TITAN 100000          2008 JUL 31 02:12:56.13
 91    308  FLYBY TITAN 100000          2008 NOV 03 17:34:50.97
 93    324  FLYBY TITAN 100000          2008 NOV 19 15:56:01.87
 94    340  FLYBY TITAN 100000          2008 DEC 05 14:25:19.42 .
```

PERL scripts similar to the `Tour#_replacer.prl` script exist to process other types of input files, as well as HTML display files. The above examples, however, clearly demonstrate the methodology.

IV. Conclusion

Not only has the code and methodology described here eased the individual labor necessary to generate a Tour Atlas, and allowed Cassini scientists the ability to analyze in depth over 30 candidate Extended Mission trajectories, Cassini has already found subsequent application. This methodology has been used to examine downlink geometries for Cassini's Solstice Mission rapidly. There is every reason to expect that the techniques on which we report won't find application again on Cassini, and could prove useful in applications for other missions.

Acknowledgments

Copyright 2010 California Institute of Technology. Government sponsorship acknowledged.