Strata: A Stratigraphic Modeling Package

Release 2.14

P.B. Flemings, J.P. Grotzinger, J.E. Morris

You can download the code, user manual, and a tutorial guide to Strata here: http://www.jsg.utexas.edu/flemings/intranet/software/strata/strata-download-the-code-manual-and-tutorial/.

*** Strata was last revised December '97 ***

If you have an older version, you will want to download updated material.

Table of Contents

1. History 2

2. Introduction 2

3. Using Strata 2

3.1. Setbasin 3

3.2. Simbasin 3

3.3. Plotbasin 4

3.4. Filmbasin 4

4. Discussion of the controlling variables 7

4.1. Measures Group Parameters 7

4.2. Ages Group Parameters 8

4.3. Clastics Group Parameters 8

4.4. Carbonates Group Parameters 9

4.5. Sealevel Group Parameters 10

4.6. Subsidence Group Parameters 11

4.7. Compaction Group Parameters 11

4.8. Heat Flow Group Parameters 12

5. Seismic Response Prediction 14

6. Appendices 15

6.1. References 15

6.2. Downloading Strata 17

6.3. Licenses and Copyrights 18

6.4. GNU General Public License 19
6.5. Changes from Release 2.0 25
6.6. Unimplemented Features 27
6.7. File Formats 27

1. History

Strata is a basin modeling package now freely available on the Internet. The model was originally developed as a part of Flemings' dissertation at Cornell University under the direction of Teresa Jordan (Flemings and Jordan (1989), Flemings and Jordan (1990), and Jordan and Flemings (1991)). Subsequently, support from Chevron Oil Field Research Company (now Chevron Petroleum Technology Company) allowed Flemings and Grotzinger to begin collaboration with Morris, then an undergraduate at MIT. Development has continued at both Penn State and MIT. NSF Grant EAR-90-19633 (Flemings), PRF American Chemical Society Grant 18887-AC2 (Jordan), and funding from Chevron (Flemings and Grotzinger) and Shell Development (Flemings) have supported this project.

2. Introduction

Strata is composed of four programs: a pre-processor **setbasin** to set the model's parameters, a processor **simbasin** to run the model with the user-defined parameters, a post-processor **plotbasin** to display results, and a special utility **filmbasin** which combines processing and post-processing to make a movie. The Strata package should work on UNIX and UNIX-like systems (incl. Linux, NetBSD) in general. (See Downloading Strata.) It is not available for other systems (e.g. MS Windows, Macintosh).

Some annotation in this manual: typewriter font indicates commands to be entered by the user. Names in &lt angles &gt indicate controls on a graphic interface. Italicized text usually indicates a file or directory.

The Appendices have information about academic references, downloading Strata (and information on program development and environments), licenses and copyrights and the GNU General Public License, changes from the last release (including patches), known bugs and not-yet-implemented features as of this release, Strata-related mailing lists, and necessary file formats to interface your own models with Strata.

3. Using Strata

Refer to Downloading Strata for instructions on downloading Strata. You should have the executables for **setbasin**, **simbasin**, **plotbasin**, and **filmbasin** either installed on your system (so that they can be used from anywhere) or in their subdirectories in *src*. If the latter, we recommend you make a *simulations* directory somewhere and link the executables into it. (E.g. ln -s

/myhomedir/strata/src/*basin/*basin /myhomedir/strata/simulations/ will link all of them at once, *after* you've compiled them in src.)

With the executables available, you are ready to begin using the applications.

3.1. Setbasin

This is the pre-processor; its purpose is to edit the model's parameters. These are held in a *library file* which defines the basin and is always given a *.dat* suffix to distinguish it. (All Strata programs will assume a *.dat* suffix for libraries.) Library files are created with **setbasin** and passed to **simbasin**; users shouldn't edit them directly. (Users of Strata 2.0 should see Changes from Release 2.0 about using **setbasin** to convert their old libraries to v2.1 format.)

To create a library from scratch, you might do the following:

- Type setbasin [A **setbasin** window will pop up; it will have the default parameters]
- **Setbasin** presents the parameters of the basin in groups. See the help entries for details. Edit any or all of the parameters in any or all of the groups to describe your basin.
- Click < save > [A subwindow will pop up]
- Type mybasin in the subwindow and hit < confirm > . [**Setbasin** will always add the *.dat* extension if you leave it out]
- Click < exit > [You will *not* be asked for confirmation, because it knows all your changes have just been saved]

You now have the library *mybasin.dat* with the parameters you specified. Any you did not change have the default values; you can get a perfectly good library (which produces a short, small simulation) with just the default values and no changes.

To make a similar library later, you could start up with setbasin mybasin. This automatically loads up *mybasin.dat* and presents you with those parameters. You can then make changes and save back to *mybasin.dat* (the default choice it presents you with when you click < save >) or to a different library.

If you're already running **setbasin** and want to load your library, you can click < load > and type mybasin in the resulting subwindow (and then hit < confirm >). **Setbasin** will then have the *mybasin.dat* parameters. It is not necessary for a library file to be complete. If some parameter is not specified --- either because you used the options to save and load in groups, or (most likely) because new parameters have been added in a new release --- **simbasin** will use the default values for the unspecified parameters. If you load an incomplete library into **setbasin**, it will use all the values the library does give, and leave the other parameters as they were before.

3.2. Simbasin

This is the processor; its purpose is to run the simulation and generate the output files to be viewed with **plotbasin** (the post-processor).

- simbasin *mybasin*
  [N.B. This may take a while. The length of the run will depend greatly upon the number of nodes and timesteps you choose.]
- ls *mybasin**
  [You should see the files *mybasin.dat, mybasin1.out, mybasin2.out, mybasin3.out, mybasin4.out, mybasin5.out, mybasin6.out, mybasindat.out*, and *mybasins.out*. All of these except the *.dat* are input for **plotbasin**.]

3.3. Plotbasin

This is the post-processor; it displays the results of the simulation and can generate hardcopy.

- plotbasin mybasin
  [The plotbasin window will appear, largely blank, but with a row of ages and a control console at the bottom]
- Click on one of the ages (e.g. < 50e5 >) and after a few moments an image will appear.

To produce hardcopy in plotbasin, select an option under < dump > and specify a path. An *.xwd* ending produces an xwd file, and a *.gif* ending, a gif. Note that the colors may not come out as finely as they appear in **plotbasin**.

3.4. Filmbasin

Filmbasin is used to produce a sequence of snapshots of the simulation, which can then be viewed together as a movie. **Filmbasin** splits itself into a copy of the processor and a copy of the post-processor and arranges for them to talk to each other in a certain way. After the post-processor knows what data to display and how, the processor simulates the model up until the time for the first snapshot, then pauses while the post-processor displays this and saves the picture, after which the post-processor pauses while the processor continues the simulation until the next snapshot... etc. At the end of the simulation the processor and post-processor die automatically, leaving your files behind. **Filmbasin** has the same prerequisite as **simbasin**; namely, a library file as created by **setbasin**. The syntax is then

filmbasin *library tag* [-s simulator] [-p plotter] [-f number of frames] [-a]

Library specifies the library in the usual fashion. The *tag* is a string to affix to the file names to make sure things aren't overridden --- if you make a film from the library *mybasin* with tag *poros*, you'll get files named *mybasin.film.poros.001.gif* etc., which then won't get erased when you run a film from the same library with a different tag (hence different names).

You may optionally specify a plotter (post-processor) with the -p flag. If you do not specify one, it will look for *plotbasin* (i.e. something named "plotbasin" along your path). You may specify a simulator (processor) with the -s flag. If you don't, it will look for something called *simbasin*. These two flags are useful if (and only if) you have your own special processor and/or post-processor, or your copies of **plotbasin** and **simbasin** are located somewhere not on your path.

You will frequently want to specify the number of frames with the -f flag. If you do not, it defaults to the number of timelines as specified in the library file. If you specify a number greater than this, it will be cut down to this.

If the -a flag was given, **filmbasin** will prompt you for arguments to pass to the plotter and simulator. Either or both may be left blank. Any arguments you specify will have the same effect as using them directly in the plotter or simulator. For instance, if you specify -p myparams as a plotter argument at this point, and the plotter is **plotbasin**, it will load the parameter file *myparams* at startup, that being what plotbasin *library* -p myparams would do.

The size of the **plotbasin** window is fixed when started by **filmbasin**, because the methods we have found so far to turn gifs into real physical films require this size; if people find problems with this it'd be easy enough to change (I don't know why it isn't a settable flag already). You can set whether the timelines are visible, what color data to plot, etc., normally in **plotbasin**. Set these as desired; they remain fixed throughout the film. You will probably want to run **plotbasin** normally on the output of **simbasin** on this library to see what settings produce nice-looking pictures beforehand. Once you are done with the settings on the plotter started up by **filmbasin**, click on any of the age markings, which will all be zeros, to start it filming. Everything from there is automatic; you can no longer use the plotter controls.

If you are filming the progress of color data, remember that most types automatically scale between the highest and lowest values so far encountered, which means the color scale will change over the course of the film. To avoid this, figure out a range which includes the highest and lowest values you expect to encounter (doing a non-film run first is the best way to do this), set the data to scale between those values in the parameters window, and turn scaling on for that datum. See the **plotbasin** help entries on color data and scaling for specifics. Also note that data fields frequently assume atypical extreme values during the first few iterations of the simulation, so scaling may result in odd ranges (most notably for porosity, which seems to always have its low end pushed to zero at the start of films).

The files generated will be named in the format *(library).film.(tag).(number).gif*, where *(number)* is the number of the frame and gets zero-padded to make them all have the same number of digits (i.e. a 2000-frame picture will have numbers 0001... 0010... etc.). The plotter automatically makes gif rather than xwd files when started by **filmbasin**.

Note that all this will take a long time to run. If you use the default number of frames, and don't have an unusually coarse-grained or unusually small simulation, expect to leave it running overnight. Also, while a gif file is a highly compressed way to store graphics data, storing thousands of them still takes up a lot of space --- make sure you have room!

How do you get an actual film out of the files? There are companies that will take a collection of gif files and hand you film. Check the yellow pages.

There are also various utilities out there that let you view the film on the screen with little or no further processing of the *.gif*'s.

XAnim is one such utility. It's free and fairly widely distributed, so you may well have it already. If not, look at the xanim homepage: http://xanim.va.pubnix.com/home.html or ftp to xanim.va.pubnix.com. These have very good directions for getting, compiling, and using **xanim**.

We didn't write **xanim** and we haven't used it very much, so we probably won't be able to help if you have questions about it. We have found that **xanim** displays Strata films nicely when invoked by

```
xanim -Cd (library).film.(tag).*.gif
```

though you should note that it takes quite a while to start up to the point of providing a window, and a long time after that to start the animation --- it's loading a lot of data! Also, if there are a lot of gifs it may not be able to store all of them.

MPEG is a standard format for video data. Again, we didn't write it and we don't know much about it, so we'll only say a little about it here. There's a good chance you have the applications *mpeg_encode* and *mpeg_play* available on your system. The former can take a bunch of *.gif* files (among other things) and turn them into an **MPEG** file; the latter can display an **MPEG** file as a movie. An HTML link to an **MPEG** file (with a .mpeg, .mpg, or .mov extension) will provide a click-to-view movie on a webpage.

Mpeg_encode requires as input a parameter file describing what it's supposed to be doing along with the *.gif*'s it's supposed to do it to. We have found that a parameter file that looks like

```
PATTERN          IBBPBBPBBPBBPBB

OUTPUT           mybasin.mpeg

BASE_FILE_FORMAT    PPM

GOP_SIZE          30

SLICES_PER_FRAME    1

PIXEL            HALF

RANGE            10

PSEARCH_ALG        LOGARITHMIC

BSEARCH_ALG        CROSS2

IQSCALE           8

PQSCALE           10

BQSCALE           25

REFERENCE_FRAME     ORIGINAL

YUV_SIZE          640x480

INPUT_CONVERT       giftopnm *

INPUT_DIR          .

INPUT
```

```
mybasin.film.mytag.*.gif [001-250]

END_INPUT
```

produces the correct results. Depending on what graphics converters you have, you may need to change the BASE_FILE_FORMAT and/or INPUT_CONVERT lines. If the above is in *mpeg.params*, then mpeg_encode mpeg.params will read files *mybasin.film.mytag.001.gif* thru *mybasin.film.mytag.250.gif* and produce *mybasin.mpeg*. Mpeg_play mybasin.mpeg will then display the latter.

4. Discussion of the controlling variables

The following discussion will explore how each of the parameters, defined in **setbasin**, affects the operation of the simulator, group of parameters by group. It is recommended that you read **setbasin**'s help entry for each parameter before altering it. A tutorial of exercises in Strata is available with the Strata package; we recommend using it as a guide to the effects of each parameter.

At the heart of the simulation is the assumption that sediment transport behaves diffusively (i.e. volumetric flux is proportional to local gradient). For carbonate simulations the sediment source is proportional to water depth, while for clastic simulations the sediment source is a user-specified function. We strongly recommend that you flip through Jordan Flemings (1991), Flemings and Jordan (1989), Flemings and Jordan (1990), and Kaufman et al. (1991) to gain a physical insight into this process. The most useful of these for marine simulations will be Jordan and Flemings (1991).

What follows is a quick description of the important controls. A note on units: any system can be used as long as it is self-consistent. We have personally been working in an m/k/yrs world, and examples are in m/k/yrs units; when **plotbasin** gives a unit label, it assumes m/k/yrs, so you will need to mentally apply conversion factors there if you use another system.

4.1. Measures Group Parameters

total width covered

> specifies the horizontal span of the simulation.

spatial divisions

> determines how many intervals the total width is divided into; it is therefore the spatial accuracy of the simulation. Note that the simulator runtime is proportional to the square of the number of spatial divisions, and file output size is linearly proportional.

total time covered

> specifies the duration of the simulation.

temporal divisions

> determines how many intervals the duration will be divided into in calculating the time evolution of the basin; it is therefore the temporal resolution of the simulation. This must be a multiple of the number of timelines.

# timelines

determines how many recordings the simulator makes of the state of the simulation. It must be a factor of the number of temporal divisions. Each timeline records the surface condition (and the condition of every underlying surface) at that time. The default parameters specify 1000 temporal divisions in 100 timelines, so the state is saved after every 10th iteration (every 20,000 years). Note that the runtime of the simulation varies linearly with the number of temporal divisions; file output size is linear in the number of timelines. Also, while the validity of the model's approximations is obviously generally dependent on the number of temporal divisions, the porosity and thermal data fields are actually dependent on the number of timelines instead, as they are calculated at timelines only, rather than at each iteration.

4.2. Ages Group Parameters

use even increments

specifies whether the ages of the time slices should just be even divisions of the total duration. The default is that they should be, allowing you to ignore this group entirely.

age of *n*th time slice

specify the individual time slice ages if they are not being automatically set to even increments. They must be in increasing order. At each time slice, the simulator will save a data file describing the state of the simulation at that age. These ages are those listed beneath the image in **plotbasin** (well, actually each one is the first age at least equal to the specified age, since the iterations may not hit the specified age exactly).

4.3. Clastics Group Parameters

These define the diffusion specifications for the model. Both marine and non-marine transport rates are controlled by the topographic gradient,

$$q = -k\frac{\partial h}{\partial x}.$$

See Jordan and Flemings (1991) and Kaufman et al. (1991) for insight into the meaning of these diffusion constants. To generate a realistic shelf break, the nonmarine diffusion constant is set to a high value, while the marine diffusion coefficient is set to a much lower value. A simple rule of thumb is that to generate steeper clinoforms you should decrease the marine diffusion constant.

nonmarine diffusion constant

sets *k* in the sedimentation equation above, in at least the nonmarine zone (i.e. water depth less than zero.) Depending on the other parameters, it may set *k* everywhere.

simulate marine sedimentation

allows for *k* to vary between marine and nonmarine zones. If false, *k* will have the nonmarine value everywhere.

marine diffusion constant

sets *k* in the marine zone (very large water depth) if marine sedimentation is being simulated. The transition between nonmarine and marine diffusion constants can be smooth or abrupt; see the diffusion constant decay equation below.

decay coeff. for marine diffusion constant

governs the (exponential) rate of decay for marine diffusion from nonmarine to marine values. That is,

$$k = k_{marine} + (k_{nonmarine} - k_{marine}) \cdot e^{-\lambda w}$$

for *w > 0*, the decay constant being lambda.

left, right clastic fluxes

define the flux entering the basin from each side. Each may be specified as a number, which means a constant flux, or as a file, which means that file defines the flux as a function of time (see the **setbasin** help entry on flux files). We recommend that all clastic flux files be given a *.clas* extension.

pelagic sedimentation rate

defines a constant sedimentation rate in the marine zone as an additional source term.

4.4. Carbonates Group Parameters

This group of parameters deals with calcium carbonate sedimentation. The fundamental difference between CaCO3 sedimentation and clastic sedimentation is that the CaCO3 sedimentation is a source term whose magnitude is dependent on water depth and which can occur anywhere along the cross-section being modeled. In contrast, the clastic sedimentation is defined by an input flux on the left- or right-hand side of the model; this flux is then redistributed by slope-dependent diffusion. See Gildner and Cisne (1990) for more details.

carbonate file

Two carbonate deposition algorithms are possible and can be mixed together in a single profile, epeiric sedimentation and oceanic sedimentation. The carbonate file specifies (as per the **setbasin** help entry) exactly what carbonate deposition operates at each point. **Setbasin** has a facility (explained therein) to allow you to view the shape of the curves defined by each carbonate equation with your parameters. We recommend that all carbonate files be given a *.carb* extension.

$$\frac{d}{dt}sed = c_1 \cdot \frac{w}{w_0} \cdot e^{1-\frac{w}{w_0}} \qquad w > 0$$

Epeiric (exponential-depth) sedimentation, where *c_1* is the maximum sedimentation rate and *w_0* is the depth of the maximum sedimentation rate. See Gildner and Cisne (1990).

epeiric CaCO3 sed --- max rate

specifies *c_1* in the above equation.

epeiric CaCO3 sed --- depth of max rate

> specifies *w_0* in the above equation.

$$\frac{d}{dt}sed = c_1 \cdot e^{-c_2(w-w_0)} \qquad w > w_0$$

> Oceanic (lag-depth) sedimentation, where *c_1* is the maximum sedimentation rate, *c_2* is the exponential decay constant, and *w_0* is the depth of the maximum sedimentation rate. For *w < w_0, dsed/dt = c_1*.

oceanic CaCO3 sed --- max rate

> specifies *c_1* in the above equation.

oceanic CaCO3 sed --- exp. decay constant

> specifies *c_2* in the above equation.

oceanic CaCO3 sed --- depth of max rate

> specifies *w_0* in the above equation.

lag on zero water depth

> controls carbonate lags when the system shoals to sealevel. If it is set to True, the lag (depth and/or time) is initiated when the depositional surface exactly matches (or exceeds) the sealevel, allowing multiple shoaling cycles even with no external forcing. If this is False, the lag is only initiated when the depositional surface is above the sealevel (which must be caused by non-carbonate effects).

isotope signal file, isotope offset time

> aren't really carbonate parameters, but they're closer to that than to anything else. They allow for the tracking of spatially homogenous, temporally varying isotope signals, as recorded at the surface. Subsequent *vertical* movement of the stratigraphy inhomogenously stretches and compresses the signal (possibly eliminating some, at lacunae); no other dynamics are taken into effect. The final (horizontally varying) distorted signals can be examined to test isochronicity fitting. See the help entries for descriptions of isotope files and further information.

4.5. Sealevel Group Parameters

sealevel file

> may be omitted completely (in which case the other sealevel parameters are used to generate a sine wave eustatic curve) or used to specify the eustatic curve in great detail. Such files can give a point-by-point specification of the sealevel, or define a series of fully-parameterized sine waves turning on and off with time whose sum is the sealevel.
>
> See the **setbasin** help entries for details on sealevel files. We recommend that all sealevel files be given a *.sea* extension.

The file *exxon.sea* is a special case; it holds the Haq Sealevel Curve, with times given relative to the present. Note that using it may be a bit non-intuitive due to the format of Haq's file. Since it gives time from the present, and the model always uses time from the start of the simulation, it is necessary to use a highly negative time offset, at least as great as the total duration of the simulation. For instance, if the simulation is to run for 20 ma, and the time offset is specified as -100 ma, the portion of the Haq curve from -100 ma to -80 ma will be applied.

time offset

applies to all sealevel functions. At time *T*, eustatic calculations will use *(T - offset)* as the time.

datum for sealevel oscillation, sealevel oscillation amplitude, sealevel oscillation period

define the eustatic curve in the absence of a sealevel file. The curve will be a sine wave with these parameters as center, amplitude, and period, and < time offset > as the offset.

4.6. Subsidence Group Parameters

subsidence rate

determines the driving subsidence rate, either as a number (a constant subsidence) or as a file (see the **setbasin** help entry for details on subsidence files). Any number of subsidence histories, defined with "wells" at positions along the profile, can be specified. We recommend that all subsidence files be given a *.subs* extension.

profile

is either ``cratonic'' (flat; subsidence constant across basin), ``foreland'' (subsidence is maximum at the left), or ``passive'' (subsidence is maximum at the right). The subsidence decrease along the basin in non-cratonic profiles is linear.

flexural isostatic compensation

determines whether or not the surface of the earth behaves as an elastic plate.

flexural rigidity

governs the elastic flex response of the earth to loads. A zero flexural rigidity will simulate perfect isostacy. Please see the **setbasin** help entry about low flexural rigidity values.

densities of air, crust, mantle, water; gravitational constant

have obvious impact upon the loads flexing the surface. **Setbasin** allows them to be specified to allow for other systems of units, and for the ambitious to model basins on other planets.

4.7. Compaction Group Parameters

compact sediments

determines whether compaction occurs at all. If this is set to False, the other parameters in this group become irrelevant. If it is set to True, compaction will occur.

We follow the approach of Sclater and Christie (1980) and assume porosity is an exponential function of depth:

$$\phi = f \cdot \phi_{0,sand} \cdot e^{-\lambda_{sand} z} + (1 - f) \cdot \phi_{0,shale} \cdot e^{-\lambda_{shale} z}$$

where *f* is the fraction of sand (assuming only sand and shale are present) and the constant parameters are defined below.

let erosion affect compaction

determines whether the effects of erosion will be taken into consideration in the compaction calculations. If so, compaction is irreversible (i.e. the strata do not expand when the overburden is decreased). See Hart et al. (1995) for a discussion of this behavior.

decay constants for sand, shale compaction

specify *lambda_{sand}* and *lambda_{shale}* respectively in the above porosity equation and govern the falloff of porosity with depth as per Sclater and Christie.

initial porosities for sand, shale

specify *phi_{0, sand}* and *phi_{0, shale}* respectively in the above porosity equation and determine the surface porosities as per Sclater and Christie.

We note that the default values from Sclater and Christie (1980) are:

$\phi_{0,shale}$	0.63
$\phi_{0,sand}$	0.49
$\phi_{0,chalk}$	0.70
λ_{shale}	$0.51 \cdot 10^{-3}\ m^{-1}$
λ_{sand}	$0.27 \cdot 10^{-3}\ m^{-1}$
λ_{chalk}	$0.71 \cdot 10^{-3}\ m^{-1}$

cutoff for sand composition, decay constant for composition

There are two ways in which Strata may calculate the sand-shale percentages in the model: from diffusion constants, or as a function of water depth. If < cutoff for sand composition > is set to a negative number, then < decay constant for composition > becomes irrelevant and the composition is determined by linear interpolation of diffusion constants. If < cutoff for sand composition > is zero or positive, then the composition will be taken to be pure sand between the surface and this depth, and decay exponentially with the specified constant below the cutoff depth.

4.8. Heat Flow Group Parameters

Strata is capable of using a simplistic steady-state model to calculate temperature distributions and thermal evolution (i.e. integrated time-temperature history, Sum TTI) of the basin. Only vertical heat transport is accounted for, and the only heat source is flux supplied at the base of the basin. The temperature distribution is calculated from

$$q = -k\frac{dT}{dz}$$

where *k* is the thermal conductivity (watts per meter degree), *T* is temperature (degrees Celsius), and *q* is the heat flux (watts per square meter). Because this is a steady-state problem, given *q* and *k* we can calculate the temperature gradient at any point given any boundary condition. We choose to set the temperature at the sediment surface by explicitly specifying the temperature at the air interface and using

$$T = T_{air} - \alpha z$$

to determine the temperature at the sediment-water interface for underwater areas. We can then calculate the temperature at any location.

The thermal conductivity is assumed to vary as a function of lithology and porosity.

$$k = \phi \cdot k_{fluid} + (1 - \phi) \cdot (f \cdot k_{sand} + (1 - f) \cdot k_{shale})$$

where *f* is again the fraction of sand in the sand and shale, and *phi* is porosity. We work in MKS units (since the thermal quantities only enter the model in equations with each other, we can safely think of them all in MKS instead of m/k/yrs with no effect on the numbers), and so our usual values are (Turcotte and Schubert, 1982)

$$\begin{array}{lll} k_{sand} = & k_{quartz} = & 3.00\ \frac{W}{m^2} \\ k_{shale} = & k_{illite} = & 3.01 \\ k_{fluid} = & k_{water} = & 0.50 \end{array}$$

where the third significant digit has been specified so that areas of sand and shale will be distinguishable when the model runs without compaction (and therefore with zero porosity).

thermal flux

> specifies *q* in the steady-state thermal equation above. It may be specified as a nonnegative number, giving a constant thermal flux, or as a file, giving the thermal flux as a function of time. No thermal calculations will be performed if there is no thermal flux. See **setbasin**'s help material for details on thermal flux files. We recommend that all thermal flux files be given a *.therm* extension. Typical continental heat fluxes are 56.5 milliwatts per square meter; typical oceanic heat fluxes are 78.2 milliwatts per square meter (Turcotte and Schubert, 1982). Note: syn-rift heat fluxes are typically higher than continental heat fluxes (Turcotte and Schubert (1982), Waples (1985)).

thermal conductivities of sand, shale, fluid

> specify *k_{sand}*, *k_{shale}*, and *k_{fluid}* respectively in the thermal conductivity equation above.

surface temperature

specifies *T_{air}* in the temperature falloff equation above.

surface temp falloff

specifies *alpha* in the temperature falloff equation above.

The simulator calculates the Sum Time-Temperature Index (Sum TTI) as

$$\int 2^{\frac{T-100}{10}} dt \approx \sum 2^{\frac{T-100}{10}} \Delta t$$

where the TTI which is summed is a continuous version of the empirical

$$TTI = 2^{n(T)}$$

for which *n(T)* has been found to be crudely approximated by

Temperature interval	n
$30-40\ ^{\circ}C$	-7
$40-50$	-6
$50-60$	-5
$60-70$	-4
$70-80$	-3
$80-90$	-2
$90-100$	-1
$100-110$	0
$110-120$	1

The fundamental trait here is that the reaction rate doubles with every temperature increase of 10 degrees C.

The TTI value indicates how much the sediment has matured in that time interval. The Sum TTI is thus the total maturity of the sediment. The following table shows the usual interpretation of the Sum TTI values. (Note that these are typically given in megayears, not years.)

Sum TTI	**Interpretation**
$10ma$	*early oil generation*
40	*peak oil generation*
75	*late oil generation*
180	*wet gas (has liquid generation)*
900	*dry gas*

5. Seismic Response Prediction

Plotbasin has the ability to predict the seismic response of the modeled basin. Briefly, since the porosity is everywhere known, as is the sand-shale composition, the density of and velocity of sound in the material is uniquely determined. (If the library turned compaction off, the porosity is taken to be zero everywhere.) As described in Compaction Group Parameters, simbasin and plotbasin determine the sand-shale composition via either the prevailing diffusion constant or deposition water depth.

Velocities are determined from the Wylie equation

$$\begin{aligned}\frac{1}{v_b} &= f\cdot\left(\frac{\phi}{v_{fluid}}+\frac{1-\phi}{v_{sand}}\right)+(1-f)\cdot\left(\frac{\phi}{v_{fluid}}+\frac{1-\phi}{v_{shale}}\right)\\ &= \frac{\phi}{v_{fluid}}+(1-\phi)\cdot\left(\frac{f}{v_{sand}}+\frac{1-f}{v_{shale}}\right)\end{aligned}$$

where *f* is again the fraction of sand in the sediment (ignoring fluid), *v_{material}* is the velocity of sound through that material. *V_b* is the resulting bulk velocity and is a selectable data type under < fill > , < log > , and < contour > .

The velocities of sound in sand, shale, and fluid are set in the parameters window. Typical acoustic velocities are (from Schlumberger, 1987)

v_{sand}	$5487\ m/s$
v_{shale}	$4545\ m/s$
v_{water}	$1604\ m/s$

Bulk density is determined from

$$\begin{aligned}\rho_b &= f\cdot(\phi\rho_{fluid}+(1-\phi)\rho_{sand})+(1-f)\cdot(\phi\rho_{fluid}+(1-\phi)\rho_{shale})\\ &= \phi\rho_{fluid}+(1-\phi)\cdot(f\rho_{sand}+(1-f)\rho_{shale})\end{aligned}$$

where *rho_{material}* is the material density and *rho_b* is the resulting bulk density, which is a viewable data type. Material densities are also specified in the parameters window.

The impedance of a layer of sediment to sound (also selectable data) is derived as

$$Z=\rho\cdot v.$$

The reflection coefficients are defined by

$$RC=\frac{Z_2-Z_1}{Z_2+Z_1}=\frac{\rho_2 v_2-\rho_1 v_1}{\rho_2 v_2+\rho_1 v_1}.$$

and may be made visible with the < misc > menu.

The final signal, consisting of the explosive wavelet (selected in < misc > and visible when relevant in the top left) convolved with the reflection coefficients, may take a few moments to be calculated. It is viewable under < fill > or < log > . (Signal data cannot, of course, be contoured.)

6. Appendices

6.1. References

Flemings, P.B. and T.E. Jordan, ``Stratigraphic Modeling of Foreland Basins: Interpreting Thrust Deformation and Lithospheric Rheology,'' *Geology* 18: 430-434, 1990.

Flemings, P.B. and T.E. Jordan, ``A Synthetic Stratigraphic Model of Foreland Basin Development,'' *Journal of Geophysical Research* 94: 3851-3866, 1989.

Gildner, R.F. and J.L. Cisne (ed.), ``Quantitative Modeling of Carbonate Stratigraphy and Water-Depth History Using Depth-Dependent Sedimentation Function in Cross, T.A.,'' *Quantitative Dynamic Stratigraphy*. Prentice Hall, New Jersey, 1990.

Haq, B.U., J. Hardenbol, and P.R. Vail, ``Chronology of fluctuating sea levels since the Triassic,'' *Science* 235: 1156-1187, 1987.

Hart, B.S., P.B. Flemings, and A. Deshpande, ``Porosity and pressure: Role of compaction disequilibrium in the development of geopressures in a Gulf Coast Pleistocene basin,'' *Geology* 23 (1): 45-48, 1995.

Jordan, T.E. and P.B. Flemings, ``Large-Scale Stratigraphic Architecture, Eustatic Variation, and Unsteady Tectonism: A Theoretical Evaluation,'' *Journal of Geophysical Research,* 96 (B4): 6681-6699, 1991.

Kaufman, P., J.P. Grotzinger, and D.S. McCormick, ``Depth-dependent diffusion algorithm for simulation of sedimentation in shallow marine depositional systems in Franseen, E.K.,'' in W.L. Watney, C.G.S.C. Kendall, W. Ross (eds.) *Sedimentary modeling: Computer simulations and methods for improved parameter definition*, Kansas Geological Survey, 1991.

Log Interpretation Principle/Application, Schlumberger Educational Services. Houston, 1987.

Pegrum, R.M. and A.M. Spencer, ``Hydrocarbon Plays in the northern North Sea,'' in Brooks, J. (ed.) *Classic Petroleum Provinces*. Geological Society Special Publication, London, 1990.

Ross, W.C., D.E. Watts, and J.A. May, ``Insights from Stratigraphic Modeling: Mud-Limited Versus Sand-Limited Depositional Systems,'' *AAPG Bulletin* 79(2): 231-258, 1995.

Sclater, J.G. and P.A.F. Christie, ``Continental Stretching: An Explanation of the Post-Mid-Cretaceous Subsidence of the Central North Sea Basin,'' *Journal of Geophysical Research* 85 (B7): 3711-3739, 1980.

Turcotte, D.L, and G. Schubert, *Geodynamics: Applications of continuum physics to geological problems.* John Wiley & Sons, New York, 1982.

Waples, D.W., *Predicting Thermal Maturity, in Geochemistry in Petroleum Exploration.* International Human Resources Development Corporation, 1985.

6.2. Downloading Strata

Strata is written in C with X Windows graphics (using the X Toolkit and the Athena Widget set, both of which are part of the standard X distribution), development being with X11R5.

Strata has been successfully used on some DEC MIPS Ultrix, HP/UX v9, SGI, AIX, NetBSD, Linux, Solaris, and SunOS systems; the latter two platforms probably offer the most stability, being the main development environments. It ought to work in UNIX and UNIX-like systems in general. Strata is not available for other systems (e.g. MS Windows, Macintosh).

Copies of this manual in Latex and Postscript are available on the Strata Downloading page along with the Strata source, tutorial, and patches.
http://www.jsg.utexas.edu/flemings/intranet/software/strata/strata-download-the-code-manual-and-tutorial/

You may also retrieve Strata via anonymous ftp from the Penn State sytem:

- ftp hydro.geosc.psu.edu or ftp 128.118.41.244
- give anonymous or ftp as the login name
- give your complete email address as the password
- cd /pub/gbrn/strata
- binary
- dir (this gives you a list of the files there)
- for each file you want, get *filename*
- quit (this exits the ftp session)

The first file you should get is README; read it. It will tell you what the other files there are. For any other file that you download, if there's a README for it you should get that too and read it --- e.g. if you get *tutorial.tar.gz* first read *tutorial.README*.

Unpacking: Most of the material is packed in *.tar.gz* files. To unpack such a file, either gtar xvfz *name.tar.gz* if you have **gtar** (gnu tar) on your system, or else do gunzip *name.tar.gz* and then tar xvf *name.tar*. In either case, you will be left with a copy of the tarfile as well as its contents; you'll want to rm the leftover tarfiles.

Gzip: Most UNIX environments have gzip, but it's easy to get if you don't. Like other GNU applications (e.g. gtar), gzip source for various environments is available to anonymous ftp to prep.ai.mit.edu in /pub/gnu. If necessary, gzip -d acts the same as gunzip.

Source Code: The source tarball creates the directory *src*, and in it a directory for each of the programs, each with its own source. In *src* there is a *configure* script which will handle compilation for you. Cd to *src* and do

- ./configure
- gmake or make

This should leave you with each program's executable in its directory. If you have the correct access you can install these with make install --- if you have this access, you probably already know about this; if not, you should talk to your sysadmin if you want the programs installed. See *README.install*.

This manual does not appear in *src*, but is available (in LaTeX and Postscript) in *man.tar.gz*.

Tutorial: This material appears in a *tutorial* directory. It includes the libraries and data files used in the tutorial. **WARNING**: the tutorial postscript was made in FrameMaker and has lots of pictures so it is very, very big --- you may have trouble even printing it. You should be able to view it with **ghostview** and such.

Also **please note** that the tutorial has *not* been updated to reflect changes in the package since it was first written! For instance, the tutorial claims that the data files appear elsewhere, whereas they actually appear in the *tutorial* directory itself. Basically, the information on actually using Strata is accurate, but references to how the package appears is suspect. In case of conflict, this manual is correct.

Revisions: Revisions to Strata will be available in the same places (ftp and WWW); those important enough will be announced to the strata-users list.

Usually, but not always, a revision will be available both as a complete package to take the place of your old Strata and as a patchfile which can be used to patch your old Strata; there will not always be a patchfile. When there is, to patch your copy of Strata, download (and gunzip and untar, just as for the original source) the patches between your version and the latest. Place the resulting patchfile(s) in *src*. In that directory,

- cat *oldest_patch second_patch...* | patch -N -E -p1

where | is a vertical bar (``pipe''). This will patch your source. Then

- rm config.cache
- ./configure
- gmake or make

and you will have the new executables; of course, they'll need to be reinstalled if you installed the old ones.

6.3. Licenses and Copyrights

You should have received a copy of the GNU General Public License along with this program (*license.h* in the source code); if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

The source file *license.h* contains a copy of the Gnu General Public License, and the source file *strata.h* contains some excerpts from this manual.

You may wish to note that the xwd capabilities of the program are derived directly from the xwd source (files *xwd_dsimple.c*, *xwd_dsimple.h*, and *xwd_here.c* of **plotbasin**), which has the following copyright notice:

Similarly, the gif-making code (*gif_here.c* of **plotbasin**) is taken from code with the following licensing / copyright notice:

6.4. GNU General Public License

Version 2, June 1991

Preamble

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third

parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is

END OF TERMS AND CONDITIONS

6.5. Changes from Release 2.0

- Bug fix: the density of water is now a parameter in **setbasin**. The original model used cgs exclusively, and *rho_w* was accidentally left as 1 when the units changed.
- Bug fix: diffusion constants of areas above water are now set to the nonmarine diffusion constant; they had been undefined previously.
- Bug fixes: various things about films. Two-way time works; seismic signal displays work; and sealevel and shoreline curves work.
- Bug fix: if the extreme values for some data field are equal, they are moved apart infintesmally to avoid division by zero.
- Bug fix: the ``default'' units (assuming m/k/yrs) displayed for accumulation and max accumulation are now really mm/yr; previously they claimed to be cm/yr and were actually in meters (with no time factor).
- Bug fixes: the isostasy algorithm now simulates perfect isostasy when the flexural rigidity is zero; also, the load now includes sediments from depositions previously left out of it, such as carbonates and pelagics.
- Bug fix: **plotbasin** now ought to work on all types of screens.
- Various subsidiary data files no longer require special termination lines.
- The *#.out* files are now in binary format. This compresses them to 30% of their old size and allows **plotbasin** to read in the data much more quickly. However, it also means that **plotbasin** v2.1 cannot read data files written by **simbasin** v2.0.
- Wheeler diagrams now display lacunae in grey for all data types, not just water depth. (For maximum accumulation, only hiatuses are so displayed, not vacuities, of course.)
- If the model was run without compaction and/or without heat flow, plotbasin blanks out functions that depend on those, rather than cluttering the screen with irrelevant options.
- The scalebars showing horizontal and vertical lengths now choose their own lengths appropriately instead of having to be set. The sealevel curve in the Wheeler diagrams acts in the same way, as does the new Wheeler vertical scalebar.
- Seismic signal displays work properly with the vertical axis as depth, instead of being restricted to two-way time.
- The < time offset > in **setbasin** applies to *all* eustatic functions, not just Haq curve use.
- **Setbasin** has been changed almost completely, and we're not going to list every change here; but in particular: Boolean variables can be specified as ``True,'' ``False,'' ``On,'' ``Off,'' etc. (see **setbasin** help for a list) instead of only as ``1'' and ``0'' (though those still work). Secondly, **setbasin** may be almost totally operated by the keyboard, rather than requiring switching

between keyboard and mouse; see **setbasin** help information for details. Lastly, if you have v2.0 libraries you wish to use, setbasin -convert *library1 library2 library3...* will convert *library1.dat, library2.dat, library3.dat...* from v2.0 to the current version, leaving the old versions in *library1.orig* etc.

(Strata was patched to v2.11)

- Mostly fixes to bugs that caused compiler complaints or compile/runtime failures on some systems, notably those with X earlier than X11R5.
- **Simbasin** correctly reads in v2.0-style carbonate files, which required a special termination line; those lines are no longer required, but are still acceptable for compatibility. A bug in the compatibility code had been leading to very odd results for such files.

(Strata was patched to v2.12)

- A misformed line in **setbasin**'s *help.h* is fixed.
- The above fix to carbonate files created a new problem by leaving in several lines of testing code, such that carbonate runs aborted. This has been fixed.

(Strata was revised to v2.13)

- A multitude of things that made some compilers annoyed were fixed.
- **Setbasin**'s help text should no longer have, on some displays, black blotches at the side.
- Some **setbasin** keybindings were fixed, and bindings for PageUp and PageDown were added.
- Gif dumps should not break on high-depth displays.
- Signal interpolation should no longer occasionally produce spurious axis-like lines.
- The internal operations of **filmbasin** have been changed extensively; this should remove its tendency to randomly crash.

(Strata was patched to v2.14)

- **Plotbasin** will no longer sometimes loop forever on startup.
- **Plotbasin** ought now to display properly on all setups. As a side effect, users will more often see other applications displayed in false colors when **plotbasin** has the focus (and vice versa); **plotbasin** should always display correctly when it has the focus.
- **Setbasin** will no longer on some setups (mostly Linux) produce random non-text characters in its parameters and/or output (which was very confusing to **simbasin**).
- Isotope record tracking has been added.
- The Wheeler sealevel curve shares a baseline with the isotope curve under isotope data; they are colored differently, and are much thicker than previously.
- **Simbasin** now explicitly requires that the number of temporal divisions be a multiple of the number of timelines, intead of letting this go by and then crashing.

- The **plotbasin** parameters window allows for easier movement between parameters, via the arrow keys.
- The Makefile.in's allow for install properly.

6.6. Unimplemented Features

- There should be a facility to print out data fields generated by **simbasin** or calculated by **plotbasin** in plain text files in grids on space-time, for users who want to do arbitrary further manipulations.
- **Setbasin** should check sets of variables for necessary relations to be warned about, instead of leaving that for **simbasin**. (For instance, it is left to **simbasin** to check that timeslice ages are in order and no more than the total duration.)
- Diffusivities, pelagic sedimentation, and various other constants should have the option of being specified as files varying in time or in space.
- The final state of a simulation should be available as the initial state of another simulation.
- Noise should be available at various parts of the simulation to represent stochastic processes.
- **Plotbasin** should be able to print directly to a printer and to postscript files.
- **Filmbasin** should have flags to allow the fixed **plotbasin** size to be set, and to allow the size to be left free.
- **Plotbasin** and **setbasin** should go to greater lengths to select appropriate fonts.

(Release 2.0 listed ``simulate compaction-driven fluid flow in basin'' as a desired feature; that has become a distinct, non-Strata program, **xflows**.)

6.7. File Formats

If you have or are interested in creating your own basin simulator, and wish to use **plotbasin** to display its output, your simulator will need to create the same output files as **simbasin** and in the same formats.

The *library*s.out file must contain the following, in ascii, in order, all numbers being separated by whitespace:

- Six integers giving, in increasing order, the number of timelines in each time slice.
- One integer for every timeline in the oldest time slice, giving the shore boundary's spatial node at that time, counting nodes from zero on the left edge.
- One double precision floating point number for every timeline in the oldest time slice, giving the sealevel at that time.
- For each isotope being tracked (possibly none), in order: one double precision floating point number for every timeline in the oldest time slice, giving the isotope value at that time.

The *library*dat.out file must contain the following, in ascii, in order, all numbers being separated by whitespace:

- An integer giving the model number. **Simbasin** is defined to be model zero; a simulator written by Jen Carlson is defined to be model one. Your model number must therefore be greater than one. **Plotbasin** checks the model number at certain points for model-specific behaviors; Carlson-model data is treated slightly differently than that from **simbasin**, and yours might have to be too (you get to edit the **plotbasin** source to accomplish this).
- A floating point version number, to allow version-dependent code as well as model-dependent code.
- An integer giving the total number of iterations (temporal divisions, as opposed to timelines).
- A double precision floating point distance between spatial nodes.
- An integer number of spatial nodes.
- An integer giving the number of iterations per timeline.
- A double precision floating point time in years between iterations.
- Six double precision floating point ages of the time slices, in order.
- The number 0 if compaction was turned off, and any other integer if it was turned on. If compaction was turned off, **plotbasin** will assume no porosity data was written out.
- A double precision floating point nonmarine diffusion constant.
- A double precision floating point marine diffusion constant.
- A double precision floating point cutoff depth for sand composition. A negative value indicates composition is to be calculated from diffusion constants.
- A double precision floating point decay constant for composition.
- Three double precision floating point heat conductivities, of sand, shale, and fluid in that order. Zero values indicate that thermal calculations were not performed, and **plotbasin** will assume no thermal data was written out.
- For each isotope being tracked (possibly none), in order, two strings (not containing whitespace), the first the isotope's name, the second its units.

Each of the files *library*1.out... *library*6.out is to be created as per

```
int i, j, k;

Boolean have_data[7];

FILE *fp;


have_data[0] = TRUE;       /* elevation                    */

have_data[1] = compaction;  /* porosity, if compaction is on          */

have_data[2] = TRUE;       /* water depth                   */
```

```
have_data[3] = TRUE;        /* max space from neighboring timeline, meters */
have_data[4] = TRUE;        /* prevailing diffusion constant               */
have_data[5] = did_thermal; /* temperature, if thermal calculations done   */
have_data[6] = did_thermal; /* Sum TTI, if thermal calculations done       */

for (i = 0; i < 7; ++i)  /* loop through the data fields */
  if (have_data[i])      /* skip those not calculated */
    for (j = 0; j < number of timelines in this slice; ++j)
      for (k = 0; k < number of spatial nodes; ++k)
        write out datum of field i on timeline j at node k to fp;
```

where the data should be written as per:

```
void write_datum(FILE *fp, double x)
{
  Boolean negmantis, negexpon;   /* mantis / exponent negative? */
  unsigned int mantis, expon;
  unsigned char buf[3];
  double y;

  if (negmantis = (x < 0))
    x = -x;
  else if (!x) {
    fwrite("\0\0\0", 1, 3, fp);
    return;
  }

  y = log(x) / .69314718 - 17;   /* binary log(x) = ln(x) / ln(2)  */
  if ((int)y == y) ++y;
  if (negexpon = (y < 0))
    expon = -y;
  else
```

```
    expon = y;

  if (expon > 31)
    if (negexpon) {
      fwrite("\0\0\0", 1, 3, fp);
      return;
    }
    else {
      printf("Simbasin crashes and burns:  can't handle %g (too big)\n", x);
      exit(1);
    }

  if (negexpon)
    mantis = x * (1 << expon);
  else
    mantis = ((int)x) >> expon;

  buf[1] = (mantis & 65280) >> 8;
  buf[2] = mantis & 255;

  buf[0] = expon;
  if (negexpon) buf[0] += 32;
  if (negmantis) buf[0] += 64;
  if (mantis & 65536) buf[0] += 128;

  fwrite(buf, 1, 3, fp);
}
```

That is: each datum is expressed as $\pm|mantissa| * 2^{\pm|exponent|+17}$ and written out in three bytes. The first byte gives the exponent (first five bits), whether the exponent is negative (sixth bit), whether the mantissa is negative (seventh bit), and the highest (seventeenth) bit of the mantissa (eighth bit). The second byte gives the ninth through sixteenth bits of the mantissa, and the third byte gives the first eight bits of the mantissa. Anything smaller than this dynamic range is written as zero; anything larger breaks.

Note that the max-space-between-timelines data field is saved in *meters*, and converted by **plotbasin** to mm/yr, because the conversion factor of *.001 / (years per timeline)* is generally enough to make the numbers small enough that they all turn into zeros.

See also **simbasin**'s *settime.c*.

Interactions via **filmbasin** are more complex; you'll have to look at **filmbasin**'s *specs.h* for that information.