Stylus: User and API Guide

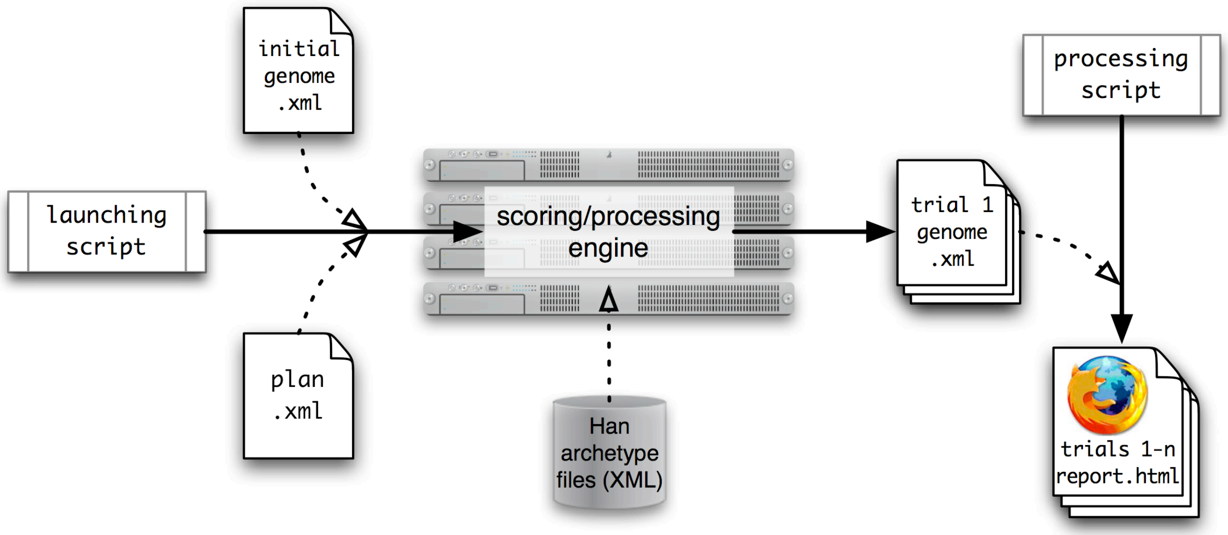
Last Saved: 6/3/08 7:42 PM

***Stylus* Version: 1.2.0**

1 Overview

This document describes the overall structure and operation of *Stylus* for the experimental user. For an overview of the concepts behind Stylus and examples of its application, see: (paper URL)

Stylus is not a single executable file, but a collection of interrelated scripts and a binary engine. It requires URL-access to `XML` files describing Han archetypes, genomes (currently restricted to a single gene), and experimental plans, as well as `XML` schema definitions. *Stylus* writes experimental results to a local or network-accessible directory. Additional scripts convert experimental output into `XHTML/SVG` reports for viewing in standard web browsers. Generally, users interact with *Stylus* through a command line script that launches instances of the core engine on one or more dedicated machines.



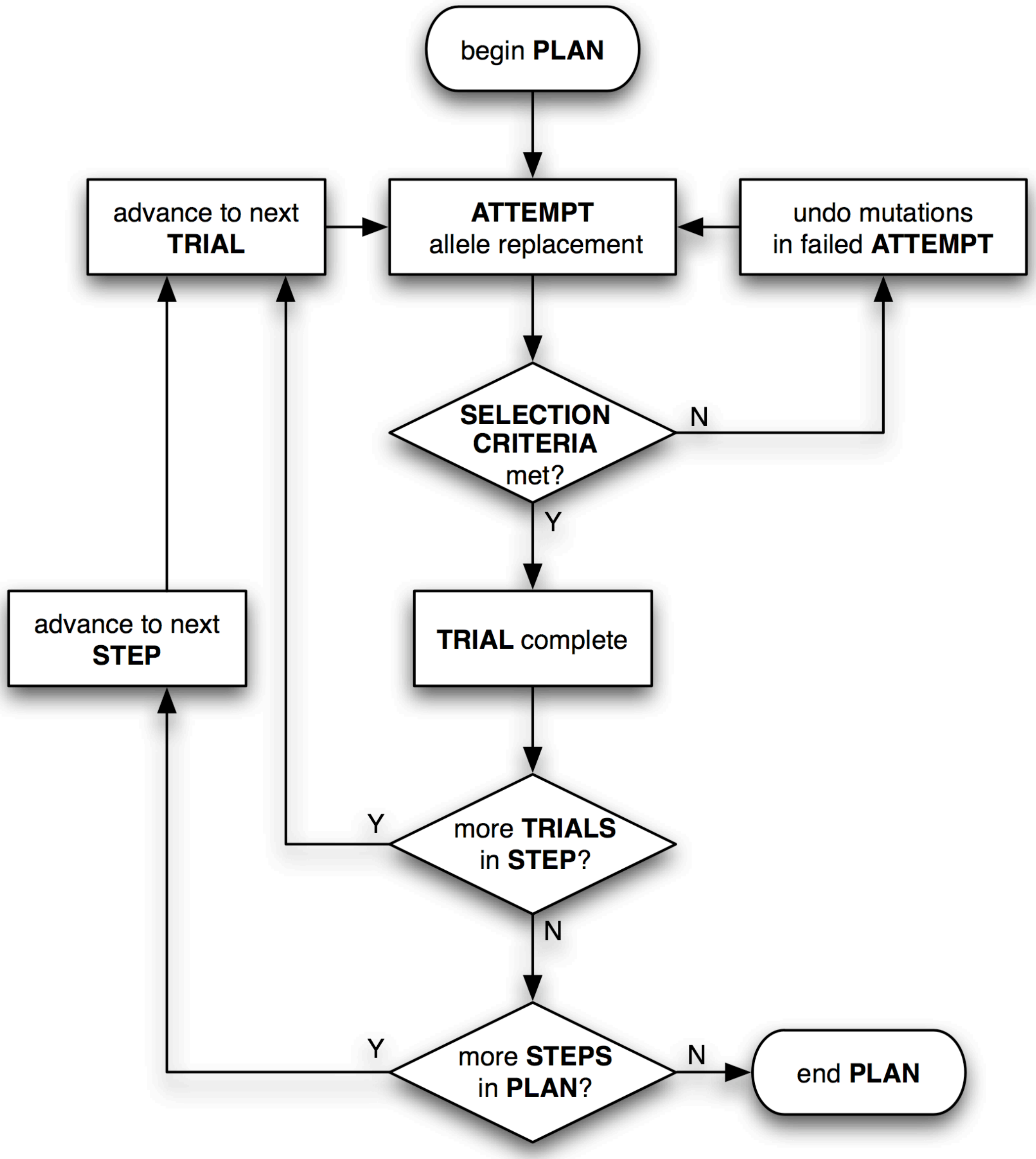
Schematic of *Stylus* system

Stylus runs line-of-descent experiments based on an initial gene sequence (read from a genome data file) along with mutation constraints and selection criteria (read from a plan file). The sequences of the genes produced during the experiment (along with other information described later) are written as a series of genome data files. Because these output files are of the same format as the initial genome file, they may be used as the starting point for new experiments.

The experimental plan provides complete specifications for a single line-of-descent experiment. The basic unit of evolutionary progression in these experiments is a *trial*, which culminates in an allele replacement event (i.e., an instance of a mutated allele entering the line of descent by passing the specified selection criteria). There is no explicit treatment of time, either in standard units or in biological generations. Time is instead counted in trials, with the events of trial n beginning after the allele-replacement event that completed trial $n-1$ and ending with the next allele-replacement event.

Depending on how stringent the selection criteria are, numerous mutant genomes may fail to meet them before one passes. The process of producing and testing any one of these mutant genomes is referred to as an allele-replacement *attempt* (or simply an *attempt*). A trial may therefore be viewed as a succession of attempts (at least one) culminating in a successful attempt.

Neither the selection criteria nor the mutation constraints need to remain fixed during the course of an experiment. Plans can specify changes of either kind by dividing the experiment into more than one experimental *step*, each having its own conditions. Plans may consist of any number of steps, which in turn may consist of any number of trials.



Overall flow of *Stylus* Plan execution.

Glossary of key terms

Term	Meaning
Allele replacement	An instance of a new mutant allele entering the line of descent by passing the *selection criteria*.
Attempt	An instance of generating a mutant allele and testing it against specified *selection criteria*. Passing this test results in *allele replacement* and *trial* completion.
Fitness	A genome-level quantity calculated from the *proficiency scores* of the encoded *vector proteins*, along with the base and vector resources used to make them (see Scoring doc).
Genome data file	An XML file specifying at least: a single gene sequence, the *Han archetype* against which it will be scored (identified by *Han Unicode number*), and locations of regions within the gene that encode *strokes*. Stylus both reads these files as inputs and writes them as outputs (with much more information included).
Han archetype	A geometric specification for a *Han character*, recorded in an XML file used by *Stylus* to calculate *proficiency scores*.
Han character	Any of the written characters of Chinese origin.
Han Unicode number	The standard 4-hexadecimal-digit identifier of any Han character (e.g., 8C5D specifies 豝).
Mutation constraints	Any of the restrictions on mutation type, frequency, number, or location, some of which may be specified for the whole experiment and all of which may be specified for an experimental *step*.
Plan	A user-edited XML file that provides complete specifications for a *Stylus* experiment.
Proficiency score	The degree of geometric likeness between a vector protein and a specified Han archetype, calculated according to the scoring algorithm (see Scoring doc).
Report	A web-browser viewable summary of a *Stylus* experiment or a portion thereof (down to the level of a single *trial*).
Rollback	An instance of undoing the mutations applied in an unsuccessful *allele-replacement attempt*.
Selection criteria	The *fitness* and/or *proficiency-score* criteria (specified in a *plan*) that must be met for an *allele-replacement attempt* to succeed.
Step	A section of an experimental *plan* specifying a number of trials to be completed and (optionally) any changes to *selection criteria* or *mutation constraints* pertaining to the whole *plan*.
Stroke	The basic structural unit of a Han character, corresponding to one drawn path.
Trial	A succession of one or more *allele-replacement attempts* culminating in successful *allele replacement*.
UUID	Universally Unique Identifier—a standard for file identification used by Stylus (see http://www.itu.int/ITU-T/studygroups/com17/oid/X.667-E.pdf).

Term	Meaning
`Vector protein`	`The product of a vector-world gene, obtained by translating codons into vectors and arranging the vectors to form a path.`
`XML schema`	`A document providing definitive specifications for XML files that are to be put to a particular use.`

2 *Stylus* scripts

Most often, users run *Stylus* by means of the provided scripts, though they may choose to interact with the binary library directly (through another binary application or their own scripts). *Stylus* scripts share most command line options and environment variables. The *Stylus* scripts are:

- **`stylus`** – A `BASH` shell script used to establish the environment *Stylus* requires. It passes most arguments to `stgrid.py`.
- **`stgrid.py`** – A Python script that launches instances of `stexec.py` and `strpt.py` either locally or on an accessible Xgrid.
- **`stexec.py`** – A Python script used to run experiments and interact with the *Stylus* binary engine.
- **`strpt.py`** – A Python script that generates web-browser loadable reports from experimental genome data files.

Each of these is described fully below. Values in **`bold`** must appear exactly as specified; `non-bold` items represent user-supplied values.

2.1 Environment Variables

The *Stylus* Python scripts share a common set of environment variables, listed in the table below, to reduce the number of required command line options. The scripts use environment variables for default values. Each has a command line equivalent which, if specified, overrides these defaults.

Variable	Definition	stgrid	stexec	strpt	inscribe[1]
`STYLUS_GRIDARGS`	Default launch script options	X			
`STYLUS_EXECARGS`	Default experiment options		X		
`STYLUS_RPTARGS`	Default report options			X	
`STYLUS_GMAKERARGS`	Default gene creation options				X
`STYLUS_DATAPATH`	Path for experiment data		X	X	
`STYLUS_RPTPATH`	Path for report data			X	
`STYLUS_GMAKERPATH`	Path for Han archetypes and Genes				X

[1] See Inscribe User Guide.

Variable	Definition	stgrid	stexec	strpt	inscribe[1]
`STYLUS_GENOMEURL`	URL for genomes		X	X	
`STYLUS_HANURL`	URL for Han archetypes		X	X	X
`STYLUS_HTMLURL`	URL for shared HTML files			X	
`STYLUS_PLANURL`	URL for plans		X	X	
`STYLUS_SCHEMAURL`	URL for the *Stylus* XML schema		X		
`STYLUS_AUTHOR`	Author name string		X		X
`STYLUS_XGRID_VOLUME`	Absolute path of *Stylus* on an Xgrid agent	X			
`STYLUS_XGRID_EMAIL`	E-mail address to notify when an Xgrid job completes	X			
`STYLUS_XGRID_SHELL`	Absolute path of the shell within which to launch *Stylus*	X			

2.2 Specifying Command Line Options

All command line options have two forms: A long form using two dashes with the full option name (such as, `--quiet`) and a short form consisting of a single dash followed by the first letter of the long name (such as, `-q`). Some take one or more additional values following the option and separated from it by one or more spaces (such as, `--datapath ./test/data`). If the option accepts multiple values, they are joined and separated by commas (such as, `--name evolve,uuid`). A few options have values accepting multiple items; these are joined and separated by a plus sign (such as, `--trace scoring+validation,3,4`). As indicated above, the scripts use default values (inherited from environment variables) when option values are not supplied. For example, if an option takes three values, the scripts will accept input with only the last value if the appropriate commas are also included (such as `--trace ,,4`). Leading commas are necessary; trailing commas are not (such as `--trace scoring+validation`).

2.3 stylus – Establishing the Environment

The `stylus` script is a helper to simplify using `stgrid.py`, `stexec.py` and `strpt.py`. It allows users to select various versions of *Stylus* (such as a debug vs. release build) and establish common environment variables (see above). The script takes few options of its own, passing most on to the target script(s). The table below lists its options. All others it passes without modification. It quits processing options at the first unrecognized option.

Option	Description	Comments
`d`	Run *Stylus* debug version	
`m`	Run *Stylus* profiled (measured) version	
`r`	Run *Stylus* release version	This is the default
`i`	Do not set argument environment variables	Affects `STYLUS_EXECARGS`, `STYLUS_RPTARGS`, and `STYLUS_GRIDARGS`
`l`	Use localhost web server for Han and Genes	Not generally used

Option	Description	Comments
`t`	Use the test configuration	Not generally used
`x`	Use the Xgrid configuration	Generally used by `stgrid.py` for Xgrid clients

For example, the first command below tells *Stylus* to run the plan `analysis.xml` on the genome data file named `4E28.gene`, and the second tells it to generate reports from the results (assuming `-x` is specified in the `STYLUS_RPTARGS` environment variable—see `--xhtml`, section 2.6). Both use the release *Stylus* version:

```
[Biologic:Stylus] $ ./stylus r –e -- -g 4E28.gene –p analysis.xml
[Biologic:Stylus] $ ./stylus r –e –r -- –g 4E28.gene
```

2.4 stgrid.py – Launching Stylus

`stgrid.py` eases launching one or more instances of either `stexec.py` or `strpt.py`. It recognizes the following options, passing those after a `--` to the invoked script(s).

2.4.1 Options

Option	Description
`--exec`	Run `stexec.py`
`--report`	Run `strpt.py`
`--job identifier,restart|results|spec`	Restart or query a previously started Xgrid job
`--genomes genome-directory`	Run *Stylus* once for each file in the passed directory
`--plans plan-directory`	Run *Stylus* once for each file in the passed directory
`--xgrid`	Launch *Stylus* using Xgrid
`--volume path-to-stylus`	Set the path to *Stylus* on an Xgrid client
`--notify e-mail address`	E-mail address for notification of completion
`--shell path-to-shell`	Shell within which to launch *Stylus* on an Xgrid client
`--controller Xgrid-controller,password`	Xgrid controller hostname and password
`--help`	Display help content
`--`	Marks the end of `stgrid.py` arguments

2.4.2 Launching Multiple Instances

Users can easily launch multiple instances of *Stylus* by specifying a directory of genome data files, a directory of plans, or both. *Stylus* will launch one instance of `stexec.py` and/or `strpt.py` for each possible combination.

Stylus expects these directories to be locally accessible not only to `stgrid.py`, but also to `stexec.py` and/or `strpt.py`. When running locally, this condition is met naturally. To ensure it's met when running on an Xgrid, `stgrid.py` launches itself, with the supplied `--genomes` and `--`

`plans` arguments, as the first Xgrid job , thereby forcing resolution of the values within the Xgrid client context. [2]

2.5 stexec.py – Conducting Experiments

`stexec.py` provides access to nearly all features of the *Stylus* binary engine including the ability to load/save genome data files, execute plans, control logging, and specify tracing. It provides additional features that make using the *Stylus* engine more convenient, such as URL-based loading of genome and plan files (*Stylus* requires these as strings) and control over the output directory name.

The `frequency` and `expand` options described below allow experiments with many trials to be run with periodic output, which may be filled in later if needed. The option `-f 100`, for example, tells *Stylus* (`stexec.py`) to write one genome data file for every 100 trials. The first of these (named `trial100.xml`) contains the genome sequence for trial 100 along with statistics on the block of trials ranging from trial 1 to trial 100. Another run may then be launched using the `expand` option to tell *Stylus* to fill in this block with more frequent genome data files (see section 4 for details).

2.5.1 Options

Option	Description
`--datapath data-path`	Specify the output path for genome data files
`--name experiment-name,asis|dir|file|unicode|uuid,plan`	Specify the experiment name
`--expand trial-number,repeat`	Specify final trial for re-running a block of trials
`--constants constants-url`	Constants to use (see `stylus.xsd`)
`--urls genome,Han,HTML,plan,schema`	URLs for accessing required elements[3]
`--genome genome-url`	Genome data file to load
`--plan plan-url,#-of-trials,#-of-first-trial`	Plan to execute
`--frequency trials-per-block,detail`	Write output in blocks, with specified level of detail
`--log rate,level,log-file-path,echo|silent,long|short`	Rate and level at which to log status
`--statistics summary|all,time`	Display statistics
`--trace regions,flow-level,data-level,trial,attempt`	Regions and levels to trace
`--author`	Specify the author name
`--quiet`	Run silently

[2] *Stylus* will use the Xgrid volume (specified by `--volume` or inherited through the `STYLUS_XGRID_VOLUME` environment variable) as the local directory.

[3] The HTML URL is not used, but include for consistency with `strpt.py`.

Option	Description
`--version`	Display *Stylus* version string
`--help`	Display help content

The `log`, `frequency`, and `trace` options take values dependent on the version of *Stylus*. Access the online help (run `stexec.py --help`) to see the exact values allowed.

2.6 strpt.py – Creating Reports

`strpt.py` creates reports from genome data files written by *Stylus* during an experiment. The option `mpdb` may also be used to generate coordinate files (pdb format) for the vector proteins corresponding to specified trials.

2.6.1 Options

Option	Description
`--reportpath report-path`	Specify the output path for report files
`--datapath data-path`	Specify the input path for genome data files to be used for reports
`--name experiment-name,asis|dir|file|unicode|uuid,plan`	Specify the experiment name
`--expand trial-number,repeat`	Specify final trial for generating reports on a re-run block of trials
`--urls genome,Han,HTML,plan,schema`	URLs for accessing required elements[4]
`--genome UUID|genome-url`	Specify the genome file on which the plan was run
`--plan plan-url,#-of-trials,#-of-first-trial`	Specify plan used[5]
`--xhtml SVG-options,XHTML-options`	Request XHTML/SVG reports
`--mpdb initial+final+trial-#+…,pitch,scale`	Request one or more vector-protein structure files (PDB format) by trial number
`--quiet`	Run silently
`--help`	Display help content

2.6.2 Creating Reports

The reports created by `strpt.py` contain almost all the information recorded by *Stylus*, including detailed score, mutation, and sequence data. Most current browsers, such as Internet Explorer 6

[4] The Schema URL is not used, but include for consistency with `stexec.py`.

[5] The first trial and number of trials values are not used, but included for consistency with `stexec.py`.

or above, Firefox, or Safari, will properly render them, though some may require installing an `SVG` rendering agent.[6]

Each report has a summary page showing the final vector protein from each block of trials in the experiment. Clicking any of these blocks on the summary page loads a page giving details for that trial block. Here, the final vector proteins is shown in detail with interactive display options. Below this (in the same window) are collapsible frames giving details of scoring, mutations, along with a display of the translated gene. Users may specify what kinds of information are initially displayed through sequences of single-character `SVG` and `XHTML` format options described below.

SVG Option	Description
`d`	Show the default feature set
`c`	Show coherent regions (strokes)
`i`	Show incoherent regions (moves)
`n`	Show noise (marks and dropouts)
`m`	Show mutations
`o`	Highlight overlap locations
`l`	Label strokes and moves
`v`	Show vector vertices
`g`	Show grid
`b`	Show bounding boxes

XHTML Option	Description
`d`	Expand the default feature set
`g`	Expand the SVG rendering
`s`	Expand scores
`m`	Expand mutations
`c`	Expand bases and codons

[6] Internet Explorer and Safari both use Adobe's `SVG` rendering agent for viewing `SVG` (see `http://www.adobe.com/svg/`). Firefox has native `SVG` support through its Mozilla rendering agent; though versions may have minor display errors.

3 Using *Stylus* Plan and Genome Files

Stylus files are `XML` files whose content is constrained by the *Stylus* `XML` schema (`stylus.xsd`). Although all files use intelligible tag names and attributes to facilitate editing, users should typically allow *Stylus* and its associated tools, like `inscribe.py`, to create genome and Han archetype files. Users are expected, on the other hand, to author plan files, which are relatively simple. Several examples are provided as starting points.

The following sections give overviews of the key files, discussing their purpose and important tags. However, since the *Stylus* `XML` schema (`stylus.xsd`) is definitive, it should be referred to for details.

3.1 Basic Structure of XML files

`XML` files are textual, structured mark-up files, similar to `HTML` but with tighter restrictions. All content in an `XML` file resides within either a case-sensitive *tag* or an *attribute*.

Tags are named items enclosed by angle brackets – such as `<genome>…</genome>`. They have a defined start, denoted by enclosing the tag name in angle brackets (e.g., `<genome>`) and end, denoted by placing `/` before the tag name within angle brackets (e.g., `</genome`). No characters or spaces may appear between the `<` or `</` and the tag name. All tags must include both a start and an end. For tags with no nested content, the abbreviated form `(<genome … />)` may be used instead of a separate start and end. Indentation is optional and, while a tag's attributes may appear on multiple lines, it's generally best to break tag content across multiple lines only on tag boundaries (that is, where tags begin and end).

Content appears within tags (that is, between their start and end) or as values within the start known as attributes. While tags may be nested, they may not overlap – the start and end must appear within the same containing tag. Attributes are named items (no angle brackets) immediately followed by the equal sign (`=`) and a quoted value (using either single – `'` – or double – `"` – quotes).

All *Stylus* `XML` documents must begin with `<?xml version='1.0' encoding='UTF-8' ?>` and contain an `xmlns` attribute in the opening tag, referencing the *Stylus* XML schema. A plan file, for example, might start with the following lines:

```
<?xml version='1.0' encoding='UTF-8' ?>
<plan xmlns='http://biologicinstitute.org/schemas/stylus/0.1'>
```

3.2 Genome Data Files

Genome data files are the key data files in a *Stylus* experiment. As input files, they provide gene sequences, stroke locations, and the Han Unicode number of the archetype. As output files they contain a great deal more information, including mutation statistics and geometric details of the encoded vector proteins.

The following summary shows hierarchical file structure by indicating tag start and end locations as in an XML file. Tags and attributes named in boldface are required.

`<genome>`

`uuid` – Required attribute identifying the genome data file.

`author` – Optional attribute specifying the user or genome author.

`<seed>` – Initializes the random number generator. Its content depends on the random number generator in use. *Stylus* interprets the seed as the state of the random number generator at completion of the most recent trial. Using the default random number processor,[7] the seed may be either two long integers separated by a space or a quoted[8] string using characters specified by ISO-10646 (except newline/return characters).

`</seed>`

`<bases>` – Tag containing the genome sequence, which in the current version of Stylus is a single open reading frame consisting upper-case Ts, Cs, As, or Gs without spaces, tabs, or newline/return characters.

`</bases>`

`<statistics>` – In this tag *Stylus* records various values for the most recent trial, along with maximum, minimum, and totals for the block of trials described by the genome data file.

`score` – Proficiency score of the gene.

`fitness` – Fitness of the gene.

`trialFirst` – Number of the first trial of the block of trials (block size ≥ 1) described by the genome data file.

`trialLast` – Number of the final trial of the block of trials (block size ≥ 1) described by the genome data file (this is the most recently completed trial at the time of recording).

`</statistics>`

`<lineage>` – In this tag *Stylus* records the history of attempts associated with the most recently completed trial.

`<acceptedMutations>` – Tag listing the mutation(s) in the final (successful) allele-replacement attempt within the most recently completed trial.

`</acceptedMutations>`

`<rejectedMutations>` – Tag listing all unsuccessful allele-replacement attempts within the most recently completed trial (including the mutations in each attempt).

`<rejectedMutations>`

`</lineage>`

`<genes>` – Tag containing one `<gene>` tag for each gene in the genome (this anticipates implementation of multi-gene genomes).

`<gene>` – Tag specifying gene locations within the genome (this anticipates implementation of multi-gene genomes).

`baseFirst` – Position of first base of gene (first base of start codon).

`baseLast` – Position of last base of gene (final base of stop codon).

---

[7] *Stylus* distinguishes random number generators using a `UUID`. The default random number generator has a `UUID` of `B7C9BB5D-C495-411D-82B2-1929FD30A7A3`.

[8] Using either single (') or double (") quotes.

`<hanReferences>` – Tag containing information on mapping vector protein to Han archetype.

`<hanReference>` – Tag containing the Unicode number of the Han archetype, along with mapping of vector-protein strokes to the strokes in the Han archetype.

`unicode` – Unicode number for Han archetype.

`<strokes>` – Tag containing one `<stroke>` tag for each stroke.

`<stroke>` – Tag specifying location of a vector-protein stroke within the gene and which archetype stroke it corresponds to.

`baseFirst` – Position of first base of stroke.

`baseLast` – Position of last base of stroke.

`corresponds to` – Number of the corresponding stroke in the Han archetype.

```
          </stroke>
        </strokes>
      </hanReference>
    </hanReferences>
  </gene>
 </genes>
</genome>
```

3.3 Plan Files

Plan files contain complete specifications for a single line-of-descent experiment.

The following summary shows hierarchical file structure by indicating tag start and end locations as in an XML file. Tags and attributes named in boldface are required.

`<plan>`

`<options>` – Tag containing mutation constraints, selection criteria, and termination conditions that apply to the whole experiment (though the first two are overridden by any specifications appearing within a step).

`accumulateMutations` – Optional attribute (taking a value of `true` or `false`) that controls whether *Stylus* passes the genome sequence from one trial to the next (for line-of-descent experiments) or reverts to the initial sequence after each trial (for mutagenesis experiments).

`preserveGenes` – Required attribute (taking a value of `true` in current version) that causes *Stylus* to disallow nonsense mutations or mutations that affect start or stop codons.

`ensureWholeCodons` – Required attribute (taking a value of `true` in current version) that forces insertions, deletions, and multi-base substitutions (changes) to occur in multiples of 3 bases.

`ensureInFrame` – Required attribute (taking a value of `true` in current version) that forces all mutations except point changes to start at the first position of a codon.

`rejectSilent` – Optional attribute that specifies whether silent mutations may be included in an attempt.[9]

`<trialConditions>` – Tag containing details of mutation and selection. Trial completion occurs when all selection criteria are met.

`<costCondition>` – Tag specifying a selection criterion based on resource-cost. For tag contents, see `<scoreCondition>` below.

`</costCondition>`

`<fitnessCondition>` – Tag specifying a selection criterion based on genome fitness. For tag contents, see `<scoreCondition>` below.

`</fitnessCondition>`

`<scoreCondition>` – Tag specifying a selection criterion based on the proficiency score of a gene/vector protein.

`gene` – Position of the gene (within genome) to which the condition applies (not used by `<costCondition>` or `<fitnessCondition>`). Since the current *Stylus* version supports only a single gene per genome, this value must be 1.

`mode` – Specifies what kind of threshold criterion to use. The options are `maintain, increase, or decrease.` Fixed thresholds are specified by `maintain` mode, whereas `increase` and `decrease` modes specify thresholds relative to values attained at the most recently completed trial.

`<value>` – In `maintain` mode, multiple instances of this optional tag can be used to specify a probabilistic distribution of fixed thresholds. For `increase` and `decrease` modes, only a single instance of this optional tag is allowed.

`value` – Value to maintain or from which to increase/decrease. When a selection condition is first applied in `increase` mode (at the beginning of a plan or step), *Stylus* uses the greater of this value and the value at completion of the most recent trial. Likewise, in `decrease` mode, *Stylus* uses the lesser of this value and the currently attained value.

`factor` – Positive multiplier applied to the value at completion of most recent trial (used only in `increase` and `decrease` modes)[10]

`likelihood` – Probability (greater than 0 and less than or equal to 1) of using the specified value. The sum of all likelihoods must be 1.

`</value>`

`</scoreCondition>`

`<mutationCondition>` – Tag containing one or more `<mutationsPerAttempt>` tags.

`<mutationsPerAttempt>` – Tag specifying the number of mutations made per attempt, either as a fixed count or as a distribution of possible counts.[11]

[9] When generating bases for `change` mutations, *Stylus* always ensures that the resulting sequence differs from the original.

[10] Because factors widen the range of acceptable values, it is possible for values to decrease in `increase` mode and to increase in `decrease` mode.

[11] If *Stylus* rejects a mutation – for example, a `change` that is silent when `rejectSilent` is enabled – it removes *all* of the mutations within the current attempt and randomly selects new number of mutations to make in the next attempt.

`count` – Number of mutations per attempt.

`likelihood` – Likelihood of using this count. The sum of all likelihoods must equal 1.

`</mutationsPerAttempt>`

`</mutationCondition>`

`</trialConditions>`

`<terminationConditions>` – Tag containing conditions for ending plan execution prior to completing all steps. Execution terminates when any of these conditions is met, with an error code (`ST_RCPLAN`) and explanation recorded in the final genome data file.

`<durationCondition>` – Tag that limits the total trials or attempts in an experiment.

`trials` – Maximum number of trials.

`attempts` – Maximum number of attempts.

`</durationCondition>`

`<fitnessCondition>` – Tag specifying bounds on fitness.

`maximum` – Upper fitness threshold, termination occurring if a greater fitness is attained.

`minimum` – Lower fitness threshold, termination occurring if a lower fitness is attained.

`</fitnessCondition>`

`<rollbackCondition>` – Tag that limits the number of attempts in a single trial.

`rollbackLimit` – Maximum number of attempts per trial. The value may be any non-negative integer or the word `infinite` (default value is the length of the genome).

`</rollbackCondition>`

`</terminationConditions>`

`</options>`

`<steps>` – Tag containing steps, which are plan sections that may carry their own mutation and selection conditions.

`<step>` – Tag containing step-specific conditions.

`trials` – Number of trials for this step (positive integer or the word `infinite`).

`deltaIndex` – User-supplied `sourceIndex` and `targetIndex` values (see below) are incremented by this amount after each trial. When provided with several commands, *Stylus* advances all indices, not just those of the executed command. Indices wrap if they extend outside the target range.

`indexRange` – A blank-separated pair of integers or integer percentages (e.g., `35% 55%`) that limits the range of base positions for random mutations.[12]

`geneRange` – Range of gene positions to be targeted by random mutations.[13]

`hanStrokeRange` – A Han Unicode number followed by an integer identifying a stroke to be targeted by random mutations (e.g., `52DC,6`).

[12] *indexRange* and *geneRange* are mutually exclusive; only one may appear.

[13] This optional attribute must be 1 in the current version.

`<trialConditions>` – Tag with content that follows same rules as the `trialConditions` tag under `options` (see above).
`</trialConditions>`

Any of the following mutation tags may appear within a step. If more than one appears, each must have a `likelihood` attribute specifying the probability that *Stylus* will select that command. The sum of all `likelihood` values must be 1.0. All attributes are optional; *Stylus* will generate random values for unspecified attributes (bases, for example) as needed.[14] In addition to the attributes listed below, mutation tags may carry any of the three range attributes allowed at the step level.[15]

`<insert>` – Insert one or more new bases at the specified location.
- `likelihood` – Proportion of mutations to be insertions with these attributes.
- `targetIndex` – Base position at which to make the insertion.
- `countBases` – Number of bases to be inserted (must be a multiple of 3)
- `bases` – Insert specified bases.

`</insert>`

`<delete>` – Delete one or more bases from the genome at the specified location.
- `likelihood` – Proportion of mutations to be deletions with these attributes.
- `targetIndex` – Position of first base to be deleted.
- `countBases` – Number of bases to be deleted (must be a multiple of 3).

`</delete>`

`<copy>` – Insert a copy one or more existing bases at a new location.
- `likelihood` – Proportion of mutations to be copies with these attributes.
- `sourceIndex` – Position of first base to be copied.
- `targetIndex` – Base position at which to make the insertion, or `tandem` to produce a tandem repeat.
- `countBases` – Number of bases to be copied/inserted (must be a multiple of 3).

`</copy>`

`<change>` – Change a specified number of consecutive bases.
- `likelihood` – Proportion of mutations to be changes with these attributes.
- `transversionLikelihood` – Likelihood (0.0 to 1.0) that a randomly generated point mutation will be a transversion (a change from a purine base to a pyrimidine base, or vice versa). The default value is 2/3, making the three possible changes at any position equally likely.
- `targetIndex` – Position of first base to be changed.
- `countBases` – Number of bases to be changed (must be 1 or a multiple of 3).

[14] The current version of *Stylus* requires insertions, deletions, copies, or transpositions of bases to be in whole codons (that is, multiples of three), to occur on codon boundaries, and be within the same gene (cross gene movement is, at present, disallowed).

[15] Mutation-level range values override step-level values.

```
            bases – Change to specified base(s).
        </change>
        <transpose> – Move one or more bases within the genome.
            likelihood – Proportion of mutations to be transpositions with these attributes.
            sourceIndex – Position of first base to be moved.
            targetIndex – Base position at which to make the insertion.
            countBases – Number of bases to be moved (must be a multiple of 3).
        </transpose>
    </step>
  </steps>
</plan>
```

3.3.1 Sample Plans

The following plan specifies an experiment where random point mutations (single base changes) are made at until either the fitness reaches 0.8955 or 100,000 attempts have been made.

```
<?xml version='1.0' encoding='UTF-8' ?>
<plan xmlns='http://biologicinstitute.org/schemas/stylus/0.1'>
  <options accumulateMutations='true'
           preserveGenes='true'
           ensureInFrame='true'
           ensureWholeCodons='true'>
    <trialConditions>
      <scoreCondition gene='1' mode='increase' />
    </trialConditions>
    <termnationConditions>
      <durationCondition attempts='100000' />
      <fitnessCondition maximum='0.8955' />
    </terminationConditions>
  </options>
  <steps>
    <step trials='infinite'>
      <change />
    </step>
  </steps>
</plan>
```

The next plan specifies an experiment where the first 100 bases of a gene (after the start codon) are mutated by random 3-base insertions (25% of the time), random 3-base deletions (25% of the time), or changes to a single base (50% of the time). The plan ends after 50,000 trials.

```
<?xml version='1.0' encoding='UTF-8' ?>
<plan xmlns='http://biologicinstitute.org/schemas/stylus/0.1'>
  <options accumulateMutations='true'
           preserveGenes='true'
           ensureInFrame='true'
           ensureWholeCodons='true' />
  <steps>
    <step trials='50000' indexRange='4 104'>
      <insert likelihood='0.25' countBases='3' />
      <delete likelihood='0.25' countBases='3' />
      <change likelihood='0.5' />
    </step>
  </steps>
</plan>
```

The final plan specifies an experiment where the third position of each codon is changed randomly in sequence as long as the proficiency score remains at or above 0.5892. If that condition continues to be met, the plan ends after 2,500 trials.

```
<?xml version='1.0' encoding='UTF-8' ?>
<plan xmlns='http://biologicinstitute.org/schemas/stylus/0.1'>
  <options accumulateMutations='true'
           preserveGenes='true'
           ensureInFrame='true'
           ensureWholeCodons='true'>
    <trialConditions>
      <scoreCondition gene='1' mode='maintain'>
        <value likelihood='1.0' value='0.5892' />
      </scoreCondition>
    </trialConditions>
   </options>
  <steps>
    <step trials='2500' deltaIndex='3'>
      <change targetIndex='6' />
    </step>
  </steps>
</plan>
```

4 Handling *Stylus* Data

4.1 Files, Paths, and URLs

Most of the files *Stylus* works with are `XML` files with content defined by the *Stylus* `XML` schema. *Stylus* makes no assumptions regarding filenames and extensions with the exception of Han archetype files, which must have the Unicode number (4 hex digits) of the associated Han character for the filename and an extension of `.han`. *Stylus* uses an extension of `.xml` for all files it creates.

Several options take either paths or URLs. Options requiring a path must be given a path, while those accepting URLs take either a URL or a path. Paths may be full, absolute paths (for example, those that begin with a '`/`' on Unix) or relative paths (all others). They may reference the current directory (by using a '`.`' on Unix) or a user's home directory (by using a '~' on Unix).[16]

Most options taking URLs allow a number of schemes including `http`, `https`, `ftp`, `file`, and `gopher`. URLs resolved by the *Stylus* binary engine, those referencing Han archetypes and `XML` schemas, accept only `http` and `file` schemes.

Since the number of Han characters is very large, *Stylus* assumes they are divided into subdirectories based on the first digit of their Unicode number. This affects the URLs *Stylus* creates for Han archetype files and genomes. For example, given an archetype directory URL of `http://biologicinstitute.org/stylus/han/` and a request for the archetype of the character whose Unicode number is `52DC`, *Stylus* will form and use the following URL: `http://biologicinstitute.org/stylus/han/5000/52DC.han`. *Stylus* makes similar assumptions for genomes not specified with an absolute path or URL and whose name begins with a Unicode number, such as `4E29--o1r_2_3_sx_50_pct.gene`; in this case forming a URL with the relative path of `4000/4E29--o1r_2_3_sx_50_pct.gene` (using same extension as provided).

When running *Stylus* on an Xgrid, it is important to note that path resolution occurs within the context of the *Xgrid client*. This first implies that current directory references (using a '`.`' on Unix) resolve relative to the directory within which Xgrid launches *Stylus*. Second, the Xgrid user (under which the Xgrid client executes) supplies the home directory context (for '~' on Unix). Lastly, *Stylus* resolves all other paths relative to the supplied Xgrid volume. It's generally simplest to not use current or home directory references when running *Stylus* on an Xgrid. Instead supply partial paths relative to the Xgrid volume.

4.2 Capturing and Reporting Experimental Data

Stylus generates several files from an experiment and when creating reports. The combination of the `datapath`, `reportpath`, `name`, `genome`, `plan`, and `expand` options determines where it saves these files. Generally, *Stylus* creates path names of the following form:

```
[Data or Report Path][Name][Genome Identifier][Expansion Path]
```

[16] See *Path Resolution Flowchart* on page 27 for complete details of *Stylus* path resolution.

All names begin with the data (for `stexec.py`) or report (for `strpt.py`) path.[17] What other elements the path name contains depend on the remaining options. *Stylus* includes `[Name]` only if the `name` option is specified. Various name option values control the `[Genome Identifier]`:

- `asis` – Do not include a genome identifier; use only the supplied name
- `dir` – Do not include either the name, genome identifier or expansion path; treat the data path as specifying the source directory (for `strpt.py` the report directory name will be the same as the *last* component of the data path)[18]
- `file` – Take as the genome identifier the genome's file name less the extension (the default)
- `unicode` – Use the genome's Unicode number[19]
- `uuid` – Use the genome's `UUID`

The final `name` option, `plan`, requests appending the plan file name (without the extension) to the genome identifier. Lastly, if the `expand` option is present, *Stylus* appends the `[Expansion Path]` as described in the next section. To achieve uniform naming of both generated data and the corresponding reports, `stexec.py` and `strpt.py` accept certain common parameters. See below for more detail.

Consider a simple experiment that examines a single genome line. Momentarily ignoring the other options, the following saves the output to the sub-path `data/4E29/` beneath the current directory:

```
[Biologic:Stylus] $./stexec.py -g 4E29.gene -d ./data/ -p analysis.xml
```

And this command will generate `XHTML/SVG` reports from that data into the sub-path `reports/4E29` beneath the current directory.

```
[Biologic:Stylus] $./strpt.py -g 4E29.gene -d ./data/ -r ./reports/ -x d,d
```

Specifying the `name` option along with the `uuid` value tells *Stylus* to save these results into the directory `/Stylus/data/devolve/ED4E63B6-8141-43FB-AAC0-DA63DC6618D3/` and report files to `/Stylus/reports/devolve/ED4E63B6-8141-43FB-AAC0-DA63DC6618D3/` (assuming that is the genome `UUID`):

[17] *Stylus* places no restrictions on `datapath` and `reportpath`. Though not recommended, both may reference the same directory hierarchy, effectively mixing experiment data and reports. Mixing data and reports, however, disables automatic expansion through the `--expand` option.

[18] This makes it possible to create reports by pointing `strprt.py` at a specific data directory.

[19] The genome file name must begin with the four-to-five hexadecimal Unicode number.

```
[Biologic:Stylus] $./stexec.py –g 4E29.gene –d /Stylus/data/ -n devolve,uuid –p
analysis.xml
[Biologic:Stylus] $./strpt.py –g 4E29.gene –d /Stylus/data/ -r /Stylus/reports/ -
n devolve,uuid –x d,d
```

4.3 Understanding the Organization of Experimental Data

All experiments generate at least three data files: The initial genome data file (`initial.xml`), the final genome data file (`final.xml`), and a copy of the plan used (`plan.xml`). Additionally, *Stylus* periodically saves genome data files at a frequency specified with the `frequency` option. These periodic data summaries are written to files named `trialxxxx.xml`, where `xxxx` is the number of the most recently completed trial.

In addition to recording the current genome sequence, *Stylus* records a large set of statistical values[20] in the periodic genome data files.[21] The statistics contained in the final genome data file are global, reflecting all the trials executed. The periodic genome data files, on the other hand, provide statistical summaries only for the block of trials following the last summary.[22] So, for example, if genome data files are written every 1000 trials (by specifying `--frequency 1000,all`), the statistics in the file `trial5000.xml` apply to trials 4001 through 5000.

Stylus resets its internal statistical counters each time a genome file is loaded rather than loading statistics from the file. It does this because the loaded genome may have come from any point in a prior experiment[23] and cannot be relied upon to have all values. Usually, this produces the expected behavior, especially when applying a plan to a new genome (or one purged of statistical data). It may cause unexpected behavior when re-running a plan containing a `durationCondition` (see Plan Files on page 13) with a genome from a prior experiment.

4.4 Managing Experimental Data

Long experiments could produce large quantities of output. The `frequency` and `expand` options allow genome data files to be saved at intervals, with the possibility of re-running single intervals to obtain more frequent data summaries. *Stylus* provides options that make data summary expansion of this kind nearly automatic.

Suppose, for example, that data summaries were recorded every 10,000 trials, and more frequent summaries are needed over the interval from trial 160,001 to trial 170,000 (which would be summarized in a file named `trial170000.xml`). The `expand` option tells *Stylus* to re-run part of the experiment and save the data hierarchically. *Stylus* re-runs the plan by loading the genome prior

[20] *Stylus* tracks such values as the total number of bases altered, number of attempts made, number of silent mutations, and so forth. See `stylus.h` or `stylus.xsd` for complete details.

[21] Users may control the level of record detail using the `frequency` option. See below for details.

[22] Meaning that statistics in `initial.xml` and `final.xml` reflect the entire plan run; those generated during execution cover just those trials. When expanding a trial (see the next section), the statistics of the expanded `final.xml` will agree with those of the selected trial *except* for the trial range, which will begin one trial sooner (since that is the trial loaded to initiate the expansion).

[23] Such as a `trialxxxx.xml` file, which would contain local rather than global statistics.

to the block to be expanded (in this case, the genome in file `trial160000.xml`)[24], saving the newly generated data within a subdirectory of those results. *Stylus* will look for and use the most detailed genome with the corresponding trial number.[25] In addition to the `expand` option, re-execution usually should also limit the plan trials created and alter the rate for recording results.

The first command below, executes an entire plan saving results every 10,000 trials into a directory named `/Stylus/data/devolve/ED4E63B6-8141-43FB-AAC0-DA63DC6618D3/` (by combining the datapath, experiment name, and genome `UUID`). The second command then expands the data from trial 70,000 by loading trial 60,000 and re-executing the plan for trials 60,001 through 70,000 saving the results every 1,000 trials into subdirectory named `trial70000`:

```
[Biologic:Stylus] $./stexec.py –g 4E29.gene –d /Stylus/data/ -n evolve,uuid –f
10000 –p analysis.xml
[Biologic:Stylus] $./stexec.py –g 4E29.gene –d /Stylus/data/ -n evolve,uuid –e
70000 –p analysis.xml,10000 –f 1000
```

Similarly, *Stylus* intelligently builds reports for expanded data by automatically creating links between parent and child directories when expanded data exists. Since *Stylus* creates these links when generating the reports for the expanded data, there is no need to re-build the reports in any parent directory.

As another example, suppose a plan is run with the following command:

```
[Biologic:Stylus] $./stexec.py –g 52DC.gene –d ~/data/ -f 1000 –p analysis.xml
```

Stylus will create the following files in the subdirectory `~/data/52DC` (with '~' representing the user's home directory):

```
~/data/52DC
    initial.xml
    trial1000.xml
    trial2000.xml     Standard
    trial3000.xml     files plus
    trial4000.xml     every
    trial5000.xml     1,000th trial
    final.xml
    plan.xml
```

If the user then found something interesting in the 3,000th trial (by examining the file `trial3000.xml`), running the command:

[24] `expand` provides the trial number of a trial to expand; that is, the trial whose preceding data is to be re-generated. *Stylus* loads the last previously completed trial before the number provided and executes the plan.

[25] This implies that re-executing the same command with the same trial number will expand it progressively deeper. The optional `repeat` value may be supplied to rebuild the expanded data from the first located instance (effectively deleting everything from that point down).

```
[Biologic:Stylus] $./stexec.py -g 52DC.gene -d ~/data/ -f 100 -p
analysis.xml,1000 -e 3000
```

expands the data between trials 2,001 and 3,000 capturing data every 100th trial resulting in the following files:

```
~/data/52DC
    initial.xml
    trial1000.xml
    trial2000.xml
    /trial3000
        initial.xml      ┐
        trial2100.xml    │
        trial2200.xml    │
        trial2300.xml    │
        trial2400.xml    │
        trial2500.xml    │  Standard
        trial2600.xml    │  files plus
        trial2700.xml    │  every 100th
        trial2800.xml    │  trial
        trial2900.xml    │
        trial3000.xml    │
        final.xml        │
        plan.xml         ┘
    trial3000.xml
    trial4000.xml
    trial5000.xml
    final.xml
    plan.xml
```

Note that if *Stylus* receives a request to expand trial 3,000 a second time it will use the bottom-most (within the directory hierarchy) genome data file. In this case, it would expand the genome data file `~/data/52DC/trial3000/trial3000.xml`. This is most often the desired behavior. Using the `repeat` value on the `expand` option will cause *Stylus* to expand the top-most genome data file it finds; that is, repeating the expansion in a top-down fashion.

Stylus generates and saves reports in a similar fashion when using `strpt.py` with the `expand` option.[26]

[26] `strpt.py` looks for the bottom-most report file and assumes there to be a data directory with the corresponding name.

5 Appendix

5.1 Binary API

The following sections provide a brief summary of the binary interface to *Stylus*. See the `stylus.h` header file for details.

Several *Stylus* methods take user-supplied buffers to hold values. If *Stylus* finds the buffer is too small, it will return an error code (`ST_RCBUFFERTOOSMALL`) along with the number of required bytes. Callers should allocate a buffer of the appropriate size and re-invoke the method.

5.1.1 General Methods

Stylus requires initialization before any other method invocation (except version retrieval). Well-behaved clients should also terminate *Stylus* prior to exiting.[27]

API	Description
`stInitialize`	Initialize the *Stylus* engine
`stTerminate`	Terminate the *Stylus* engine
`stGetLastError`	Return details on the last reported error
`stSetScope`	Provide the URLs *Stylus* requires to locate Han definitions and the XML schema
`stSetGlobals`	Set the *Stylus* global constants used (see `stylus.xsd`)
`stGetVersion`	Return the *Stylus* version details as a `NULL`-terminated string

5.1.2 Logging and Tracing Methods

Stylus provides rich control over logging (progress and other messages *Stylus* writes to `STDERR`) and tracing (detailed messages – also written to the log – covering internal processing). *Stylus* divides log messages into five levels: Error, Warning, Informational, Debugging, and Tracing. Enabling log messages for a level will generate messages for that level and all beneath it (for example, requesting Informational messages will also cause *Stylus* to write Warning and Error messages).

Stylus breaks tracing down into eight different trace regions, each of which it divides into two categories (flow and data), for which there are five levels of increasing detail. Flow trace statements tend to show movement through *Stylus* while data trace messages tend to document calculated values and results. While users may select trace regions independently, from all to none to various combinations, the category levels apply to all selected regions. *Stylus* recognizes the following trace regions:

- Global – Code not fitting within one of the other regions
- Genome – Code related to genome-level processing
- Han – Code related to Han definition processing
- Mutation – Code related to handling mutations
- Plan – Code ran during plan execution

[27] *Stylus*, however, does not retain open files or other system resources that are not generally cleaned up upon termination.

- Validation – Code that measures and validates a genome
- Scoring – Code that scores a genome
- XML – Code related to all `XML` processing

Lastly, since tracing can generate so much output, *Stylus* allows specifying when to begin tracing by trial number.

API	Description
`stSetLogLevel`	Specify the level of messages *Stylus* writes to the log (via `STDERR`)
`stGetLogLevel`	Return the active logging level
`stSetLogOptions`	Specify log message options
`stGetLogOptions`	Return the active logging options
`stSetLogRate`	Specify the rate at which *Stylus* should write rate-aware log messages
`stGetLogRate`	Return the current logging rate
`stClearTraceRegions`	Clear all *Stylus* trace regions
`stEnableTraceRegions`	Enable tracing in one or more *Stylus* trace regions (add to existing)
`stSetTraceRegions`	Enable tracing in one or more *Stylus* trace regions (override existing)
`stGetTraceRegions`	Return active trace regions
`stClearTraceLevels`	Clear all *Stylus* trace levels
`stSetTraceLevel`	Establish a trace level for a *Stylus* trace category
`stGetTraceLevel`	Retrieve the trace level for a *Stylus* trace category
`stSetTraceTrial`	Set the trial at which *Stylus* should begin tracing
`stGetTraceTrial`	Retrieve the trial at which *Stylus* will begin (or began) tracing
`stSetTraceAttempt`	Set the attempt at which *Stylus* should begin tracing
`stGetTraceAttempt`	Retrieve the trial at which *Stylus* will begin (or began) tracing

5.1.3 Execution Methods

Stylus provides rich control over loading, monitoring, and executing commands against a genome. Among other things, users can control the rate and detail at which *Stylus* records genome snapshots. *Stylus* also tracks numerous statistics, such as the number of commands executed and effect of the commands, useful for analysis. Lastly, *Stylus* accepts a callback method pointer, which it invokes at the rate requested, enabling callers to analyze, and possibly terminate, plan progress.

API	Description
`stSetRecordRate`	Set the rate, detail, and location where *Stylus* writes genome snapshots
`stSetGenome`	Load the genome from a `NULL`-terminated UTF-8 XML string
`stGetGenome`	Retrieve the genome as a `NULL`-terminated UTF-8 XML string
`stGetGenomeBases`	Retrieve the current genome bases as a `NULL`-terminated string
`stExecutePlan`	Execute the plan passed as a `NULL`-terminated UTF-8 XML string

API	Description
`stGetStatistics`	Obtain current statistics
`stGetGenomeState`	Retrieve the current genome state (e.g., dead, alive)
`stGetGenomeFailure`	Retrieve the last failure associated with the genome (only useful if *Stylus* returned an error code or a plan failed to complete all steps)

5.2 Python API

Since most users would likely prefer to use *Stylus* through a scripting language, *Stylus* includes a Python interface[28], briefly described below. For examples on using the API, examine the `stexec.py` script.

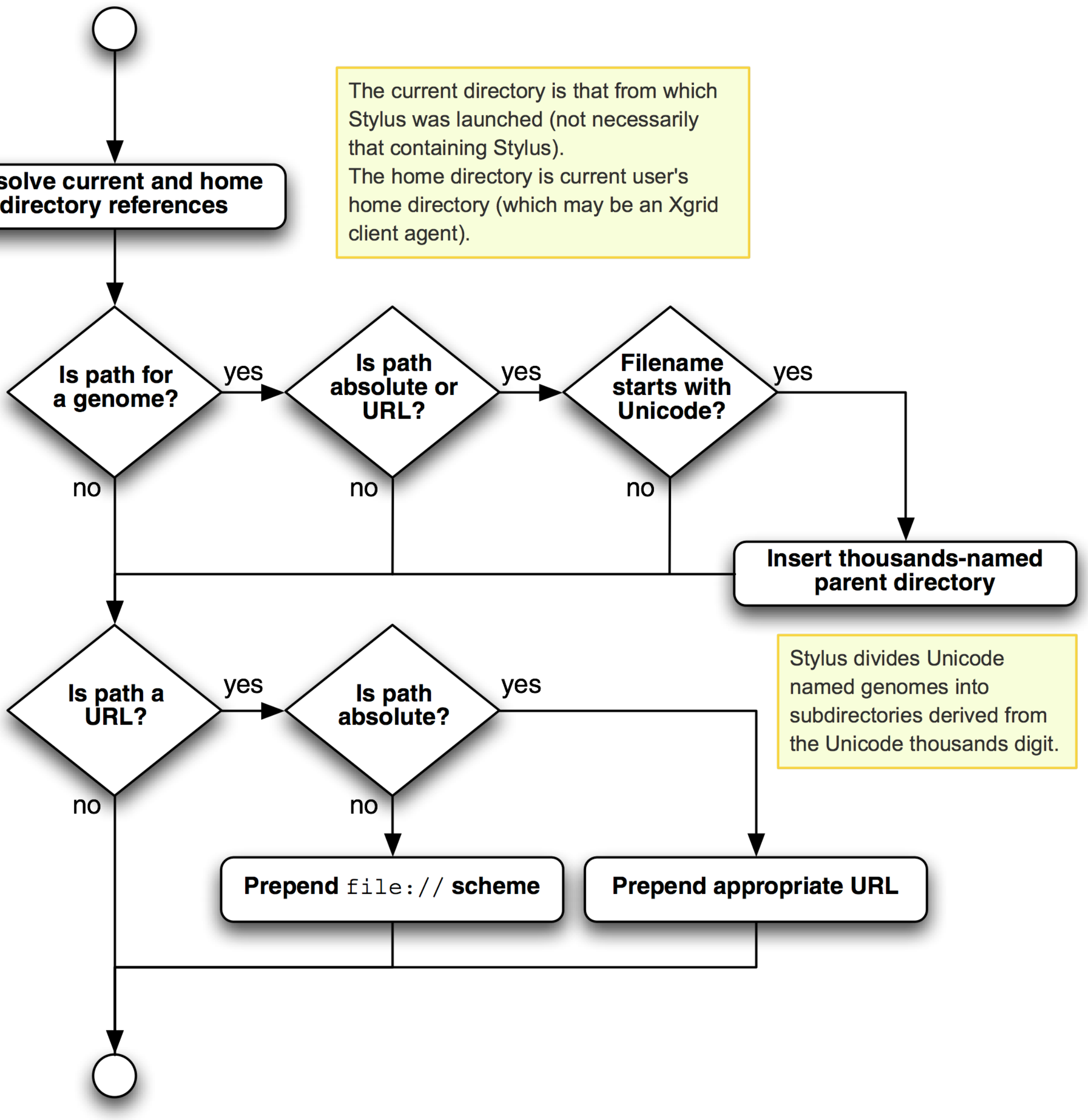
API	Description
`errorToString`	Convert a *Stylus* error code to a human-readable string
`getLastError`	Retrieve the last *Stylus* error as a human-readable string
`getLastErrorCode`	Retrieve the last *Stylus* error code
`getVersion`	Retrieve the *Stylus* versions string
`setScope`	Set the *Stylus* Han definition and XML URL scope
`setGlobals`	Set the *Stylus* global constants used (see `stylus.xsd`)
`getLogLevels`	Retrieve, as a string array, the recognized *Stylus* log levels
`setLogLevel`	Set the *Stylus* log level
`getTraceRegions`	Retrieve, as a string array, the recognized *Stylus* trace regions
`setTrace`	Set *Stylus* trace regions, category levels, and initial trial
`getRecordDetails`	Retrieve, as a string array, the recognized *Stylus* genome recording options
`setRecordRate`	Set the *Stylus* record rate, detail, and location
`setGenome`	Load the passed genome (from a string) into *Stylus*
`getGenome`	Retrieve the active genome (as a string) from *Stylus*
`getGenomeBases`	Retrieve the bases of the active genome (as a string) from *Stylus*
`executePlan`	Execute the passed (as a string) plan against the loaded genome
`getStatistics`	Retrieve the current statistics from *Stylus*
`getState`	Retrieve the current genome state (e.g., dead, alive)
`getFailure`	Retrieve the last failure code
`getFailureReason`	Retrieve the last failure reason code (specific to the failure code)
`getFailureDescription`	Retrieve the last failure as a human-readable string

[28] The *Stylus* Python interface is built using SWIG (see `http://www.swig.org/`). Porting the *Stylus* Python interface to other scripting languages, such as Perl, should be possible with minimal effort. See the `stylusengine.i` file for details.

5.3 Path Resolution Flowchart

The *Stylus* scripts make various decisions when resolving supplied paths to genomes, plans, or directories, allowing for flexible names. The following flowchart details these decisions.

Figure 1: Path Resolution Flowchart


Resolve current and home directory references
The current directory is that from which Stylus was launched (not necessarily that containing Stylus).
The home directory is current user's home directory (which may be an Xgrid client agent).
Is path for a genome?
yes
no
Is path absolute or URL?
yes
no
Filename starts with Unicode?
yes
no
Insert thousands-named parent directory
Stylus divides Unicode named genomes into subdirectories derived from the Unicode thousands digit.
Is path a URL?
yes
no
Is path absolute?
yes
no
Prepend file:// scheme
Prepend appropriate URL