[Supplemental Material:]

**Scalable Open-Source Codes:**

**First Method - Nanoribbons Code:** For this method, the “bulk” silica sample contains coordinates in the form of *x y z a*, where *x y z* represent the position of the atom and *a* indicates the type of atom (1 for Si, and 2 for O). Using the equation of a helix in Cartesian space a line is defined that travels through the bulk silica file. The parameterized Cartesian equation for a helix is: *x*(*t*) = *R* \*cos(*t*); *y*(*t*) = *R* \*sin(*t*); *z*(*t*) = *bt*, in which *R* is the radius of the helix, and *b* is the pitch constant defined by *b* = 2/*p*, where *p* is the pitch. Solving this for *t* in terms of *z,* *x* and *y* can be obtained in terms of *z*. Given an atom in the initial bulk model, the code determines whether the atom lies in the path of the helix. This produces a curled line of atoms that fit the helix based on the given parameters of pitch and radius; however it does not create a bodied nanoribbon. In order to create a nanoribbon, the program processes each atom in the “bulk” sample, calculating the shortest distance of the atom to the desired helix in the x- and y- plane. This distance is used to select this atom depending on whether or not it lies within the desired helix. If the distance is within the desired radius of the nanoribbon then the atom is added to the output data model, otherwise it is discarded. For the development of the silica nanoribbons the AWK computer language was used, which is run from a Unix command line in an Ubuntu 10.1 environment. The script runs a comparative function on each input line of a “bulk” atomistic data file, and then generates a new output data file containing the coordinates of the atoms that lie within the desired nanoribbon dimensions. This script can also be customized to generate files suitable for input to a visualization program. When running the visualization program19 the “point coordinate file” input option was used, atom types were distinguished by color, and an axis grid border was selected for the field.

**Second Method - Nanosprings Code:** NanospringCarver (v. 0.5 beta) was written in the C++ language. First the “bulk” silica glass model is read into NanospringCarver. For this method, each atom is described by a unique identifier followed by its type (1 for Si, 2 for O), then its Cartesian (*x*, *y* and *z*) coordinates. A parameter file is created (containing the radius of the helix *R*, the wire radius *r*, the pitch *p*, and the minimum distance between two atoms in the bulk model *d*). The minimum distance *d* is used to define the parametric step used in the iterative step-by-step construction of the helix. A “pruning” computational process is carried out next, eliminating atom positions that should not be considered if they lie outside the larger bounds of the nanospring, as illustrated by the light blue sections in **Figure 2**. The *x* and *y* coordinates of the atoms are used to determine if candidate atoms are outside of these bounds. If the candidate atom has an *x* or *y* coordinate which has an absolute value greater than (*R*+*r*), the atom lies outside the outer bounds. If the absolute value of either *x* or *y* atomic coordinate is less than [(*R*-*r*)/(sqrt(2))] then the atom lies interior to the inner spring boundary and is discarded. The atoms that fall outside of the bounds are not considered in any further calculations. A routine is run to evaluate the remaining atoms based on the parameterized equation of the spring. Points on the desired spring are calculated from *z*=*bt*, where *t* starts at *t\_min*=(*z\_min*/*b*) and ends at *t\_max*=(*z\_max*/*b*). Each time the routine is run, a point on the spring is calculated, starting with *t* =*t\_min* and subsequently by incrementing *t* by *dt* – a function of the minimum distance (*d*) between two points in the parameter file. For each calculated spring point position, comparisons are run on all the remaining atoms to determine if they lie within a distance *r* of this position along the helix, and if so are stored into an output data model. **Figure 3** shows the resulting selection process during the creation of an atomistic silica nanospring model.

[Insert here: Figure S1. Computational method to create nanosprings (NanospringCarver program).]

[Insert here: Figure S2. Illustration of the step-by-step atom selection process at different stages in the creation of a nanospring using the NanospringCarver program. Parts (A-D) indicate 25%, 50%, 75% and 100% completion of this process (visualized in IFrIT). The trimmed model area is shown using gridlines.]