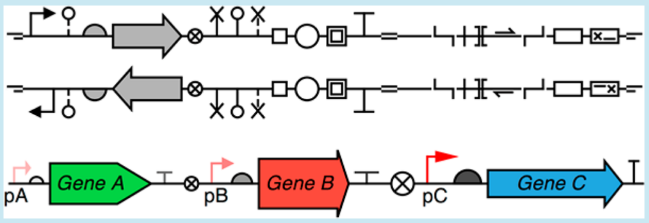
DNAplotlib: Programmable Visualization of Genetic Designs and Associated Data

Bryan S. Der,[†,#] Emerson Glassey,[†,#] Bryan A. Bartley,[‡] Casper Enghuus,[§] Daniel B. Goodman,[‖,⊥] D. Benjamin Gordon,[†] Christopher A. Voigt,[†] and Thomas E. Gorochowski[*,†]

[†]Synthetic Biology Center, Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, United States
[‡]Department of Bioengineering, University of Washington, Seattle, Washington 98195, United States
[§]MIT Microbiology Program, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, United States
[‖]Department of Genetics, Harvard Medical School, Boston, Massachusetts 02115, United States
[⊥]Wyss Institute for Biologically Inspired Engineering, Harvard Medical School, Boston, Massachusetts 02115, United States

Supporting Information

ABSTRACT: DNAplotlib (www.dnaplotlib.org) is a computational toolkit for the programmable visualization of highly customizable, standards-compliant genetic designs. Functions are provided to aid with both visualization tasks and to extract and overlay associated experimental data. High-quality output is produced in the form of vector-based PDFs, rasterized images, and animated movies. All aspects of the rendering process can be easily customized or extended by the user to cover new forms of genetic part or regulation. DNAplotlib supports improved communication of genetic design information and offers new avenues for static, interactive and dynamic visualizations that map and explore the links between the structure and function of genetic parts, devices and systems; including metabolic pathways and genetic circuits. DNAplotlib is cross-platform software developed using Python.



KEYWORDS: *visualization, standardization, SBOLv, biodesign automation, synthetic biology*

Engineering disciplines rely on standardized pictorial representations of parts and their interconnections to create schematics that clearly communicate how they are pieced together and to enable the reliable construction of large complex systems. In bioengineering, DNA sequences are often synthesized to create genetic constructs that probe or perturb the natural function of a cell or implement novel capabilities. Unlike more established engineering disciplines, the way that a genetic design is visually represented can vary significantly between laboratories and across different areas of the field. This leads to ambiguities that hinder data exchange, understanding, and the effective reuse of this research.

The Synthetic Biology Open Language Visual[1] (SBOLv) initiative was started to help alleviate this problem, defining a set of agreed symbols for commonly used genetic elements. In addition, other schemes such as the Systems Biology Graphical Notation[2] (SBGN) have been developed to more broadly standardize the graphical notation used to describe biological processes. The importance of these standardized approaches has also been recognized by publishers, with a major synthetic biology journal (ACS Synthetic Biology) adopting the use of SBOLv symbols when presenting genetic design information.[3]

Although these initiatives will help accelerate adoption of these standards, they rely on the availability of supporting tools to enable the production of compliant diagrams. Some tools do exist to generate SBOLv visualizations from genetic design information, either through graphical point-and-click interfaces (*e.g.*, VectorNTI,[4] TinkerCell,[5] GenoCAD,[6] DeviceEditor[7] and SBOL Designer) or text-based inputs (*e.g.*, Pigeon[8] and VisBOL[9]). These are effective for small numbers of constructs, but lack the ability to easily process large design libraries, are difficult to integrate into existing analyses, and offer only limited customization of the visualizations produced. An ability to tune how each genetic element is displayed (*e.g.*, the size, shape and color) based on its characterized performance[10] would enable clearer communication of key design features and enable an effective comparison of multiple designs. No tools currently support this capability.

To address these limitations, we developed DNAplotlib, a computational toolkit that enables the highly customizable visualization of standardized genetic designs in a programmable way (Figure 1). DNAplotlib is written in the Python programming language and makes extensive use of the matplotlib[11] graphics library to produce high-quality output in the form of vector-based PDFs, rasterized images and animated movies

Special Issue: IWBDA 2016

Received: September 10, 2016
Published: October 17, 2016

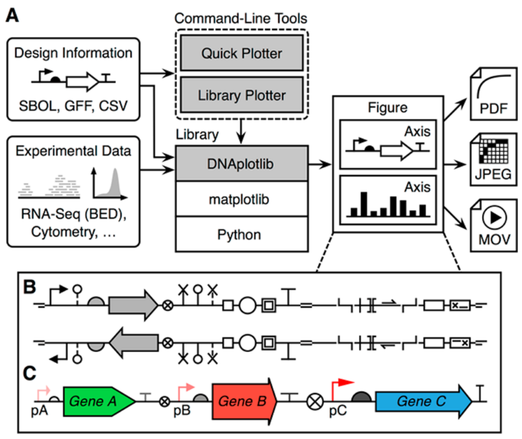


Figure 1. Overview of DNAplotlib. (A) Schematic of the visualization pipeline and supporting libraries. Genetic designs are provided as SBOL,[14] GFF or CSV files, or created through direct calls to the DNAplotlib library. Associated experimental data relating to individual parts or entire designs (*e.g.*, RNA-seq transcription profiles in the BED format[24]) can also be provided to influence properties of the visualization. Shaded boxes denote elements included as part of the library. (B) DNAplotlib supports the complete set of standardized SBOLv[1] parts in both forward and reverse orientations. (C) The size, color, shape and labeling of all genetic parts can be customized to convey associated part information, *e.g.*, promoter strength.

(Figure 2; Movie S1). Python was chosen due to its broad and growing use in the analysis of biological data, its ability to effectively "glue" together many different computational tools to create complex workflows,[12,13] and for being highly portable across all major operating systems. To simplify the process of generating visualizations in code, numerous helper functions are included to load genetic design information from files in the Synthetic Biology Open Language (SBOL),[14,15] General Feature Format (GFF), and Comma Separated Values (CSV) formats (Supporting Information). The full set of SBOLv parts[1] are available (Figure 1B) and users can easily extend existing functionality to cover new types of part or regulation and apply their own visual annotations at precise points within a genetic construct (Figures 2 and 3). Built-in parts also offer a broad range of customization options, enabling the visual communication of other characteristics such as measured performance (*e.g.*, promoter or terminator strength) that go beyond the part type alone (Figure 1C).

At the core of DNAplotlib is the main rendering pipeline (Figure 4A). This is implemented within the DNARenderer object and executed using the renderDNA(...) function. To tailor the rendering of each type of part or regulation arc, rendering functions are provided to the DNARenderer as dictionaries (part_renderers and reg_renderers in Figure 2A) with the part or regulation type mapping to the associated rendering function. Standard built-in functions can be chosen that cover the full range of SBOLv parts (Figure 1B; Supporting Information), or the user can specify their own, which may include new types of part or regulation not currently available (*e.g.*, recombinases, see Figure 3). To create a visualization, designs are provided in the form of a list where each element is a dictionary defining the part at that position in the design as well as other design information such as orientation, length and styling options. The use of a dictionary data type to store this information allows for varying numbers and types of option to be easily accommodated. Visualizations are automatically generated by scanning this list and, for each element, calling the associated function for the part type encountered. If an unrecognized part type or attribute is met, this element is ignored to ensure that multiple rendering functions with differing levels of functionality do not break the entire pipeline. Regulation is handled in a similar way with start and end points provided, in addition to styling options. Regulation links are automatically routed to minimize overlapping regions. All rendering is performed using a matplotlib axis object, which enables genetic designs to be directly incorporated into existing plotting routines (*e.g.*, bar charts and scatter plots, see Figure 2).

All built-in part and regulation renderers can be customized through the use of predefined options (Figures 4B and 5). To customize part appearance, a dictionary called opts is added to the specific part or regulation element that needs customizing. The opts dictionary defines a mapping between a customization option and the value it should take. These options are automatically sent to the relevant rendering function by the DNARenderer object when the part or regulation arc is drawn. Options not used by the renderer are ignored. A full table of all options, their format, and the elements that are compatible are shown in Figure 5.

The programmable nature of DNAplotlib opens up many unique capabilities not possible with other tools. For example, genetic circuits are composed of many parts whose regulation leads to numerous internal states of gene expression. Illustrating these and the strengths of regulation present is a challenge as the complexity of a circuit grows. Similarly, the construction of large variant libraries that explore a potential genetic design space has become commonplace as DNA synthesis costs have fallen and assembly methods have improved.[16−19] Visualizing a large number of internal circuit states or design variants manually is both time-consuming and error-prone. However, a simple computer program can be written to rapidly and accurately enumerate these, and DNAplotlib used to automate the visualization of key design features, part attributes and the internal regulatory links that are present (Figure 2A,B). This is made possible by direct programmable access, which also allows for tight integration into existing analysis workflows with minimal effort. DNAplotlib is already used within the genetic circuit design automation program Cello[20] to visualize candidate designs. Furthermore, the ability to automate the generation of large numbers of visualizations containing small variations in regulation opens up new opportunities to produce animated visualizations that convey the dynamics of a system (Movie S1). This is useful for genetic devices such as oscillators[21] whose output naturally varies in time, having no single steady state.

Another feature differentiating DNAplotlib is the inclusion of "trace-based" symbols for promoters, ribosome binding sites (RBSs), genes, terminators, and user-defined regions that goes beyond the SBOLv standard. These symbols take inspiration from genome browsers,[22,23] aiming to display not only functional information about the part type encoded at a particular point in a design, but also to provide a physically accurate representation of its position and extent within the DNA. This allows for a direct comparison to experimental data (Figure 2C) or other designs (Figure 2D) at a base pair resolution. This is achieved by either extending the length of gene and user-defined element symbols, or having filled rectangular regions cover the backbone of the construct for the length of a promoter, RBS or terminator, and using standard symbols extending from these to denote the part type (see Figure 2C,D). With sequencing seeing increased use across biology and allowing for the collection of large amounts of data at this

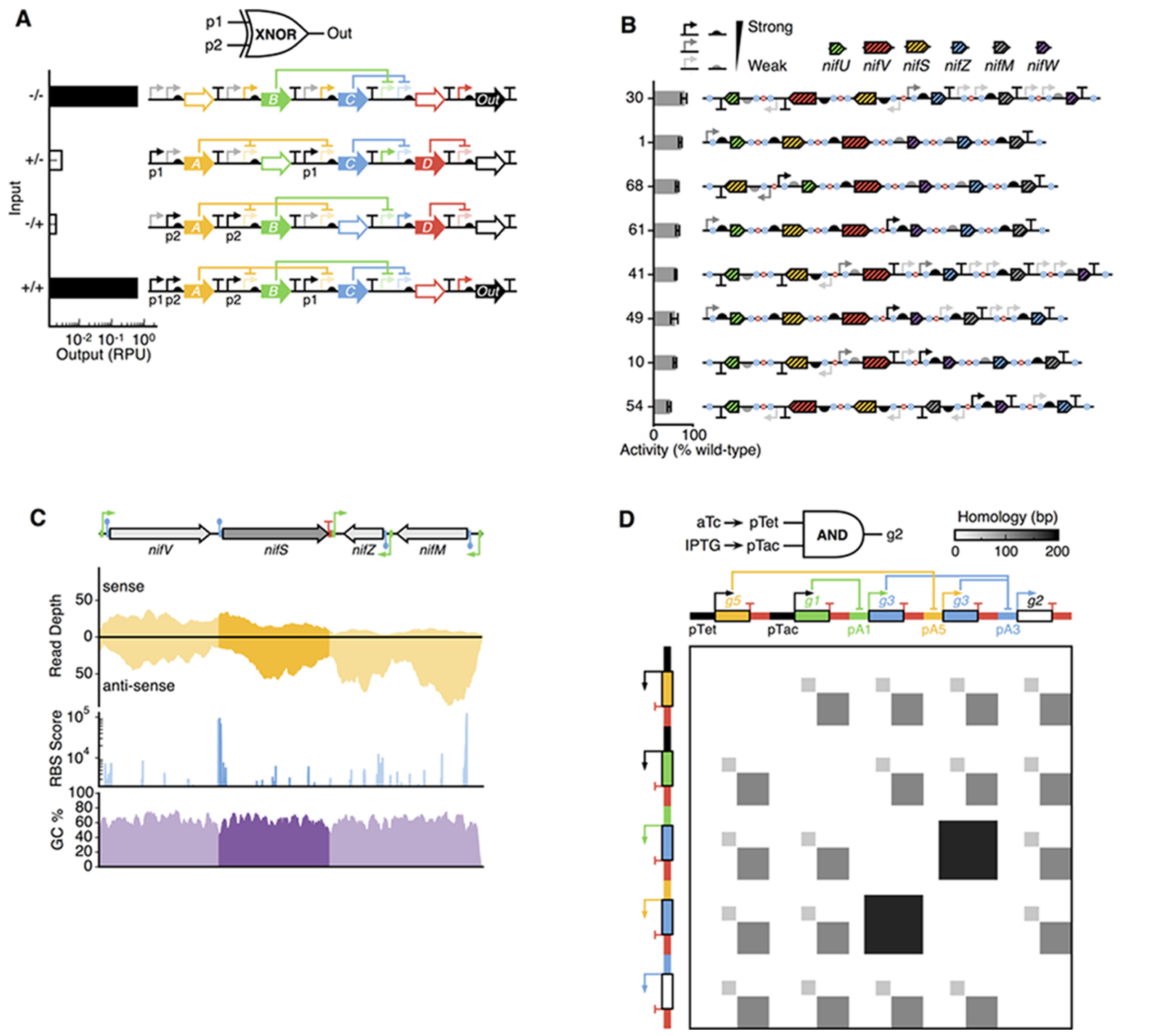


Figure 2. Examples of DNAplotlib visualizations. All are available from the project Web site. (A) Bar graph shows predicted output in relative promoter units (RPUs) for a hypothetical repressor-based XNOR genetic device designed by Cello.[20] The corresponding construct and expected state of all promoters and genes for each combination of inputs is shown to the right. Input promoters are active if black and labeled, repressible promoters are active if strongly colored, and genes are expressed if filled. Regulatory links that are present for a given set of inputs are included. (B) Selection of refactored *nif USVWZM* gene cluster designs.[25] Bar graphs represent the relative activity of the encoded synthetic nitrogen fixation pathway with error bars showing ±1 standard deviation. Numbers correspond to the variant number in the original study. In the genetic designs, promoter and RBS strengths are shown ranging from strong (black) to weak (light gray), spacer elements are blue and cloning scars are red. (C) Zoomed section of variant 75 from the *nif USVWZM* library[25] drawn using trace-based renderers to enable direct comparison of nucleotide data. Three data tracks are show: strand-specific RNA-seq read depths,[25] scores from an RBS prediction software, and GC percentage for a 50 bp centered moving window. Data for the *nif S* region has been highlighted. (D) Homology analysis of a CRISPRi circuit implementing a 2-input, 1-output AND gate.[26] The promoters pTet and pTac act as inputs and the g2 guide-RNA is the output. The same construct is plotted vertically and horizontally using trace-based renderers. Heat map shows the internal homology present ignoring homology that would be present between identical positions in each copy of the circuit. Highlighted regions show that part reuse and similarity of several regions within the guide-RNA sequences leads to potential hot-spots for recombination.

level of detail, the demand for this capability is likely to grow. To support the use of associated data at a base pair resolution, DNAplotlib includes functions to load trace files in the commonly used Browser Extensible Data (BED) format[24] (Figure 2C; Supporting Information).

Direct access to DNAplotlib from Python gives greatest flexibility when generating visualizations. However, in some cases it may be simpler for nonprogrammers to specify designs and part customizations in text-based files. These can be shared more easily, allow for better reuse of design or styling information among members of a lab, and support the wider adoption of standardized genetic designs. For this purpose, we provide two command-line interfaces. The first called "Quick Plotter" (quick.py) mimics the idea of Pigeon[8] and uses a simple syntax to define basic constructs as a single line of text. This is useful for the quick creation of small constructs with limited customization. The second called "Library Plotter" (plot_SBOL_designs.py) is designed for the visualization of

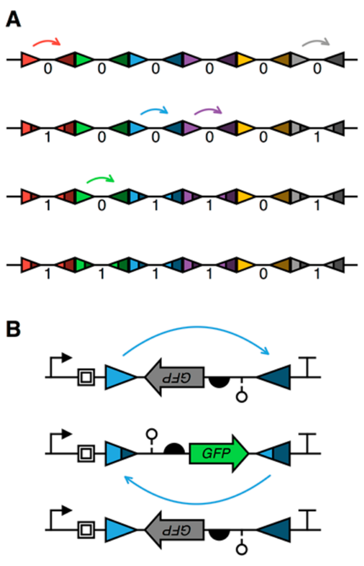


Figure 3. Extending functionality to cover new types of genetic part and regulation. Recombinase sites do not form part of the current SBOLv standard. Even so, they can be easily incorporated into DNAplotlib plots by providing custom renderers for these specific elements. (A) Array of recombinase sites implementing a 64-bit genetic memory device.[27] Current binary state is shown below each pair of recombination sites. Arrows indicate manipulations of the array at each step by integrases associated with each pair of recombination sites. (B) Illustration of a reversible recombinase NOT-gate device. In these examples, regulation arcs are used to indicate the flipping of DNA between the recombinase sites.

large design libraries. Users are required to provide several text files defining their set of parts, styling information, and the library of designs (*e.g.*, part ordering, orientation and regulatory links). These are then processed and a visualization of the full library of designs generated and saved to file.

There are also several broader functions that DNAplotlib supports. First, the visualization of genetic designs at present is a predominately manual process. Illustration tools are commonly used to draw the individual components and these are then added to existing plots of underlying data to create a final diagram. Errors in the design are time-consuming to fix and similar forms of diagram cannot be easily generated by others. While the development of a visualization using DNAplotlib might initially take a comparable amount of time, once a script is produced, it can immediately act as a template for others. For example, the novel plot shown in Figure 2D, which displays a comparison of the homology present within

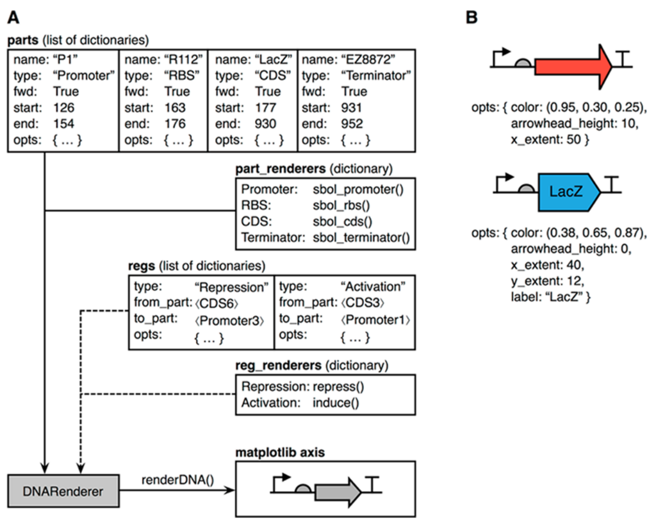


Figure 4. Data structures controlling the visualization of a genetic design. (A) DNAplotlib provides the DNARenderer object that takes design and regulatory information (parts and regs) with associated rendering functions for each element (part_renderers and reg_renderers), and then coordinates the creation of a visualization through the renderDNA(...) function. Dotted lines denote optional elements and chevrons denote part objects. All rendering is performed using a matplotlib axis to allow for the easy incorporation of other standard plotting routines. (B) The opts dictionary can be included with any part or regulatory definition to tailor the styling of the component (see Figure 5 for a full list of options). Options are shown for the coding region parts.

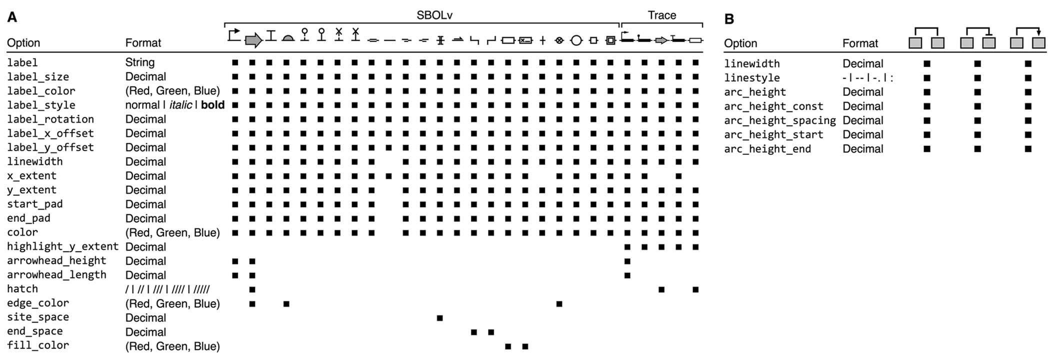
a genetic circuit, could immediately be used by others with minimal change, greatly simplifying the distribution of useful visual analyzes. Second, the maintenance of standards-compliant designs over time can be a challenge as standards often change and evolve with their field. Because DNAplotlib internally captures the visual standard, existing scripts merely need to be rerun to generate up-to-date diagrams. This provides a powerful means of ensuring the long-term applicability and relevance of visualizations developed.

DNAplotlib is released as open-source software. The project welcomes contributions from others within the community and all source code and a gallery of examples is available at the project Web site (www.dnaplotlib.org).

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge on the ACS Publications website at DOI: 10.1021/acssynbio.6b00252.



A

		SBOLv																							Trace				
Option	Format																												
label	String	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
label_size	Decimal	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
label_color	(Red, Green, Blue)	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
label_style	normal | *italic* | **bold**	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
label_rotation	Decimal	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
label_x_offset	Decimal	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
label_y_offset	Decimal	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
linewidth	Decimal	■	■	■	■	■	■	■	■	■		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
x_extent	Decimal	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		■	■	■	■	■	■		■	
y_extent	Decimal	■	■	■	■	■	■	■	■	■		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
start_pad	Decimal	■	■	■	■	■	■	■	■	■		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
end_pad	Decimal	■	■	■	■	■	■	■	■	■		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
color	(Red, Green, Blue)	■	■	■	■	■	■	■	■	■		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
highlight_y_extent	Decimal																								■	■	■	■	■
arrowhead_height	Decimal	■	■																						■				
arrowhead_length	Decimal	■	■																						■				
hatch	/ | // | /// | //// | /////		■																								■		■
edge_color	(Red, Green, Blue)		■		■																■								
site_space	Decimal													■															
end_space	Decimal															■	■												
fill_color	(Red, Green, Blue)																	■	■										

B

Option	Format			
linewidth	Decimal	■	■	■
linestyle	- | -- | -. | :	■	■	■
arc_height	Decimal	■	■	■
arc_height_const	Decimal	■	■	■
arc_height_spacing	Decimal	■	■	■
arc_height_start	Decimal	■	■	■
arc_height_end	Decimal	■	■	■

Figure 5. Customization options supported by each part and regulation type. (A) Options for all part types covering both SBOLv and trace part renderers. Black squares denote a supported option. For options with a color format, values are given as fractions of 1.0 for red, green and blue components. (B) Options for all regulation renderers.

Supplementary Text (PDF)
Movie S1: Repressilator dynamics (MOV)

AUTHOR INFORMATION

Corresponding Author
*E-mail: thomas.gorochowski@bristol.ac.uk.

Author Contributions
#B.S.D. and E.G. contributed equally to this work.

Notes
The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was supported by US Defense Advanced Research Projects Agency (DARPA) Living Foundries awards HR0011-13-1-0001 and HR0011-15-C-0084 (C.A.V., T.E.G. and D.B.G.); Office of Naval Research, Multidisciplinary University Research Initiative grant N00014-13-1-0074 (C.A.V.); NSF Synthetic Biology Engineering Research Center grant Syn-BERC EEC0540879 (C.A.V.); and National Science Foundation grant DBI-1355909 (B.A.B.).

REFERENCES

(1) Quinn, J. Y., Cox, R. S., 3rd, Adler, A., Beal, J., Bhatia, S., Cai, Y., Chen, J., Clancy, K., Galdzicki, M., Hillson, N. J., Le Novere, N., Maheshwari, A. J., McLaughlin, J. A., Myers, C. J., P, U., Pocock, M., Rodriguez, C., Soldatova, L., Stan, G. B., Swainston, N., Wipat, A., and Sauro, H. M. (2015) SBOL Visual: A Graphical Language for Genetic Designs. *PLoS Biol. 13*, e1002310.

(2) Le Novère, N., et al. (2009) The Systems Biology Graphical Notation. *Nat. Biotechnol. 27*, 735−741.

(3) Hillson, N. J., Plahar, H. A., Beal, J., and Prithviraj, R. (2016) Improving Synthetic Biology Communication: Recommended Practices for Visual Depiction and Digital Submission of Genetic Designs. *ACS Synth. Biol. 5*, 449.

(4) Lu, G., and Moriyama, E. N. (2004) Vector NTI, a balanced all-in-one sequence analysis suite. *Briefings Bioinf. 5*, 378−388.

(5) Chandran, D., Bergmann, F. T., and Sauro, H. M. (2009) TinkerCell: modular CAD tool for synthetic biology. *J. Biol. Eng. 3*, 19.

(6) Czar, M. J., Cai, Y., and Peccoud, J. (2009) Writing DNA with GenoCAD. *Nucleic Acids Res. 37*, W40−47.

(7) Chen, J., Densmore, D., Ham, T. S., Keasling, J. D., and Hillson, N. J. (2012) DeviceEditor visual biological CAD canvas. *J. Biol. Eng. 6*, 1−12.

(8) Bhatia, S., and Densmore, D. (2013) Pigeon: a design visualizer for synthetic biology. *ACS Synth. Biol. 2*, 348−350.

(9) McLaughlin, J. A., Pocock, M., Misirli, G., Madsen, C., and Wipat, A. (2016) VisBOL: Web-Based Tools for Synthetic Biology Design Visualization. *ACS Synth. Biol. 5*, 874−876.

(10) Canton, B., Labno, A., and Endy, D. (2008) Refinement and standardization of synthetic biological parts and devices. *Nat. Biotechnol. 26*, 787−793.

(11) Hunter, J. D. (2007) Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng. 9*, 90−95.

(12) Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., and de Hoon, M. J. (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics 25*, 1422−1423.

(13) Sanner, M. F. (1999) Python: a programming language for software integration and development. *J. Mol. Graphics Modell. 17*, 57−61.

(14) Galdzicki, M., Clancy, K. P., Oberortner, E., Pocock, M., Quinn, J. Y., Rodriguez, C. A., Roehner, N., Wilson, M. L., Adam, L., Anderson, J. C., Bartley, B. A., Beal, J., Chandran, D., Chen, J., Densmore, D., Endy, D., Grunberg, R., Hallinan, J., Hillson, N. J., Johnson, J. D., Kuchinsky, A., Lux, M., Misirli, G., Peccoud, J., Plahar, H. A., Sirin, E., Stan, G. B., Villalobos, A., Wipat, A., Gennari, J. H., Myers, C. J., and Sauro, H. M. (2014) The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat. Biotechnol. 32*, 545−550.

(15) Roehner, N., Beal, J., Clancy, K., Bartley, B., Misirli, G., Grunberg, R., Oberortner, E., Pocock, M., Bissell, M., Madsen, C., Nguyen, T., Zhang, M., Zhang, Z., Zundel, Z., Densmore, D., Gennari, J. H., Wipat, A., Sauro, H. M., and Myers, C. J. (2016) Sharing Structure and Function in Biological Design with SBOL 2.0. *ACS Synth. Biol. 5*, 498−506.

(16) Casini, A., Storch, M., Baldwin, G. S., and Ellis, T. (2015) Bricks and blueprints: methods and standards for DNA assembly. *Nat. Rev. Mol. Cell Biol. 16*, 568−576.

(17) Engler, C., Gruetzner, R., Kandzia, R., and Marillonnet, S. (2009) Golden gate shuffling: a one-pot DNA shuffling method based on type IIs restriction enzymes. *PLoS One 4*, e5553.

(18) Appleton, E., Tao, J., Haddock, T., and Densmore, D. (2014) Interactive assembly algorithms for molecular cloning. *Nat. Methods 11*, 657−662.

(19) Weber, E., Engler, C., Gruetzner, R., Werner, S., and Marillonnet, S. (2011) A Modular Cloning System for Standardized Assembly of Multigene Constructs. *PLoS One 6*, e16765.

(20) Nielsen, A. A. K., Der, B. S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E. A., Ross, D., Densmore, D., and Voigt, C. A. (2016) Genetic circuit design automation. *Science 352*, aac7341.

(21) Elowitz, M. B., and Leibler, S. (2000) A synthetic oscillatory network of transcriptional regulators. *Nature 403*, 335−338.

(22) Kent, W. J., Sugnet, C. W., Furey, T. S., Moskin, K. M., Pringle, T. H., Zahler, A. M., and Haussler, D. (2002) The Human Genome Browser at UCSC. *Genome Res. 12*, 996−1006.

(23) Robinson, J. T., Thorvaldsdóttir, H., Winckler, W., Guttman, M., Lander, E. S., Getz, G., and Mesirov, J. P. (2011) Integrative genomics viewer. *Nat. Biotechnol. 29*, 24−26.

(24) Quinlan, A. R., and Hall, I. M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics 26*, 841−842.

(25) Smanski, M. J., Bhatia, S., Zhao, D., Park, Y., L, B. A. W., Giannoukos, G., Ciulla, D., Busby, M., Calderon, J., Nicol, R., Gordon, D. B., Densmore, D., and Voigt, C. A. (2014) Functional optimization of gene clusters by combinatorial design and assembly. *Nat. Biotechnol. 32*, 1241−1249.

(26) Nielsen, A. A., and Voigt, C. A. (2014) Multi-input CRISPR/Cas genetic circuits that interface host regulatory networks. *Mol. Syst. Biol. 10*, 763.

(27) Yang, L., Nielsen, A. A., Fernandez-Rodriguez, J., McClune, C. J., Laub, M. T., Lu, T. K., and Voigt, C. A. (2014) Permanent genetic memory with > 1-byte capacity. *Nat. Methods 11*, 1261−1266.