ACS Synthetic Biology

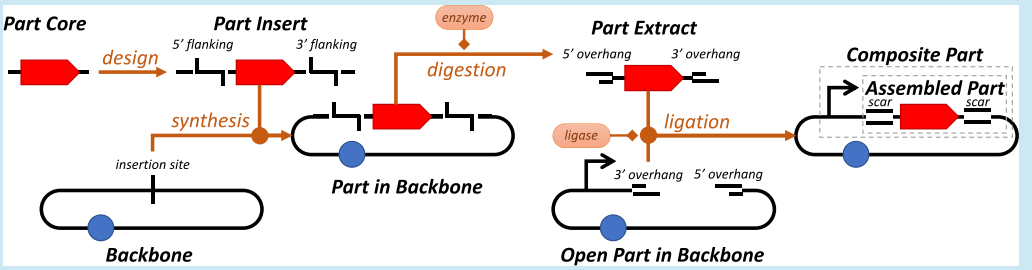
pubs.acs.org/synthbio Research Article

Standardized Representation of Parts and Assembly for Build Planning

Published as part of ACS Synthetic Biology virtual special issue " IWBDA 2022 ".

Jacob Beal,* Vinoo Selvarajah, Gaël Chambonnier, Traci Haddock, Alejandro Vignoni, Gonzalo Vidal, and Nicholas Roehner

Cite This: *ACS Synth. Biol.* 2023, 12, 3646−3655 Read Online



ABSTRACT: The design and construction of genetic systems, in silico, in vitro, or in vivo, often involve the handling of various pieces of DNA that exist in different forms across an assembly process: as a standalone "part" sequence, as an insert into a carrier vector, as a digested fragment, etc. Communication about these different forms of a part and their relationships is often confusing, however, because of a lack of standardized terms. Here, we present a systematic terminology and an associated set of practices for representing genetic parts at various stages of design, synthesis, and assembly. These practices are intended to represent any of the wide array of approaches based on embedding parts in carrier vectors, such as BioBricks or Type IIS methods (e.g., GoldenGate, MoClo, GoldenBraid, and PhytoBricks), and have been successfully used as a basis for cross-institutional coordination and software tooling in the iGEM Engineering Committee.

KEYWORDS: *synthetic biology, terminology, assembly, engineering design, SBOL*

INTRODUCTION

The creation of the first recombinant DNA molecules a half-century ago opened a new era for biology based on the design and construction of custom genetic constructs. Nowadays, one of the common tasks in synthetic biology is to plan, execute, and document the assembly of various shorter "building block" pieces of DNA into larger constructs. For example, a fragment of DNA may be synthesized as an insert into a vector backbone, digested out of that backbone, and assembled together with other fragments to produce a final construct.

Despite this commonality, however, and despite the standardization of some assembly methods and the development of new software and molecular tools for manipulating genetic sequences, it is often unclear which stage of a construction process is actually being described by a given sequence. This often creates significant confusion between practitioners as they communicate about parts and sequences and build plans, leading to frequent confusion, errors, difficulty in sharing information, delays, and waste.

We address these problems with a proposed standard terminology intended to allow unambiguous descriptions of DNA sequences and build plans through every stage of the design, synthesis, and assembly of a genetic construct. Specifically, we target the representation of build plans that make use of DNA assembly based on digestion and ligation, supporting at least BioBricks Assembly[1] and Type IIS assemblies like GoldenGate,[2] MoClo,[3] GoldenBraid,[4] and PhytoBricks.[5]

We have further mapped this terminology into a concrete representation using Synthetic Biology Open Language (SBOL) version 3,[6] a standard meant to ease the exchange of information about genetic designs throughout the design−build−test−learn cycle. SBOL3 provides all of the representational elements necessary for a precise description of genetic elements and of the construction of larger sequences from smaller sequences. Previous uses of SBOL for representing parts and assemblies, however, have lacked a systematic grounding in terminology and

Received: July 12, 2023
Revised: October 27, 2023
Accepted: October 30, 2023
Published: November 13, 2023

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655

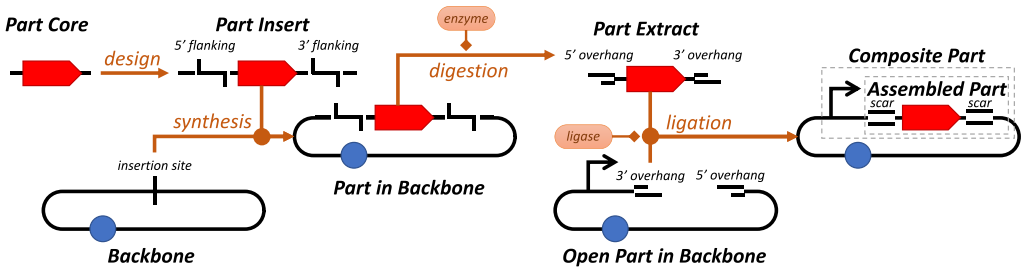


Figure 1. Illustration of build terminology with respect to a typical digestion/ligation build workflow: a *part core* is extended with flanking sequences needed for assembly to create a *part insert* that can be synthesized or assembled into an *insertion site* on a *backbone* to produce a *part in backbone* ready for assembly. Digestion produces a *part extract* that can be ligated together with an *open backbone* or *open part in backbone* and possibly other part extracts to produce a *composite part* in the backbone, including the original part core within an *assembled part* in its final context.

a set of relationships between stages, such as the one we now present.

In summary, the terminology presented here should be useful to practitioners no matter what tools or representations they may be using, while representation in SBOL3 provides complete formal details to allow software engineers to construct compatible tools. Finally, we illustrate the utility of these contributions with descriptions of how they can be applied to common synthetic biology workflows and a brief case study describing the adoption and use of both the terminology and the SBOL3 representation to support cross-institutional collaborations within the International Genetically Engineered Machines (iGEM) Engineering Committee.

RESULTS

Terminology. To develop a terminology for build planning, we began by considering a range of frequently used digestion/ligation build workflows, including BioBricks Assembly,[1] GoldenGate,[2] MoClo,[3] GoldenBraid,[4] and PhytoBricks.[5] Terms were developed with the aim of cleaving as closely as possible to pre-existing patterns in descriptions and discussions of "parts" and "assembly" among practitioners, with adjustments to eliminate ambiguity. The terminology was then circulated for comments by other practitioners, first privately and then publicly, and was adjusted iteratively to address issues raised in comments.

The resulting build terminology, illustrated in Figure 1 and detailed below, is centered on three main concepts: *part*, *backbone*, and *assembly*. The terminology also originally included a definition for a *device* as a functional mechanism in some biological context, but that definition proved unnecessary for the discussion of build planning and has since been more fully developed in a compatible fashion within the framework of "functional synthetic biology".[7] We therefore use the term "device" only as an informal or intuitive notion in this presentation and refer the reader to the other article[7] if a more formal or precise definition is desired.

Part.

- A *part* is any single continuous linear DNA construct that has a completely specified sequence and an explicit assembly interface that allows it to be assembled with other parts. The specific DNA sequences associated with the assembly interface are expected to change in a predictable manner from stage to stage in an assembly process (e.g., following digestion with restriction enzymes), so a part is always described with a modifier to explicitly indicate which stage is being described (e.g., part in backbone, part extract). Note that the assembly interface of a part might not actually contain any base pairs, if a part is intended to be concatenated as part of a synthesis order, but this needs to be explicitly declared and not assumed.
- If no assembly interface is defined, then the DNA element is a *part core.* In many cases, a part core may also be a device with a function that can be defined simply (e.g., promoter, CDS, and terminator), but part cores can potentially also be more complex devices, such as a whole functional unit or even an entire gene cluster (this is why "part core" is used rather than other potential alternatives such as "basic part"). The key distinction is in whether an assembly interface is referenced in the design. Note that this also means that a part core may include an undesirable "hidden" assembly interface (e.g., unlabeled flanking sequences or restriction sites on a poorly annotated sequence) and that any part can be transformed into a part core by discarding associated assembly interface information.
- A construct created by combining two or more parts through a plan for assembly is a *composite part.* Note that samples of a composite part need not actually be produced by its designated assembly process: for example, the composite part might be constructed directly via synthesis, including the scars that would have been formed through the planned assembly. Likewise, a composite part can be reduced to a noncomposite part by removing assembly information.
- In the postassembly context of a composite part, an *assembled part* is the original part core plus any 5′ and/or 3′ flanking *Scar* sequences that are produced by the combination of flanking sequences in the process of assembly. Critically, note that discussion of an assembled part refers only to one portion of the composite part—the portion corresponding to one of the part extracts that was combined to make the composite. For example, in Figure 1, the composite part contains two assembled parts: the assembled version of the promoter and the assembled version of the coding sequence. The notion of an assembled part thus supports the discussion of how a part has been changed by the assembly process, e.g., through the trimming of flanking sequences or the introduction of scars.

Backbone.

- A *backbone* is a DNA construct into which parts are intended to be inserted at one or more designated insertion sites, in order to meet the requirements of an assembly. In general, precisely one part can be inserted at

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655

any given insertion site (though this part might be a composite part formed by ligation of multiple part extracts). In many cases, a backbone will be a circular plasmid with precisely one insertion site, but other types of vectors are possible as well, such as plasmids with multiple insertion sites, linear plasmids, viral replicons, or nonreplicating flanking adapters.

- A part core, plus any 5′ and 3′ flanking sequences, that is prepared for insertion into a backbone's insertion site is referred to as a *part insert*. Note that some backbones come with "built in" flanking sequences, in which case the part insert may be nothing more than the part core itself.
- When a part insert is added to a backbone, the overall DNA construct, i.e., the backbone with at least one occupied insertion site, is referred to as a *part in backbone*. The process of inserting a part insert into a backbone can, as a side effect, lead to the release of a *drop-out sequence*, which is a portion of the backbone at an insertion site that is removed as part of the process of inserting a part at that site. Some backbones include drop-out sequences, while others do not.
- In the inverse direction, the extraction of a part from a part in backbone generates a *part extract*, corresponding to the part core plus any 5′ and/or 3′ flanking sequences involved in the assembly. Note that the flanking sequences might not have been present in the part insert, if the backbone had "built in" flanking sequences as described above.
- Complementary to a part extract, an *open backbone* is a backbone that has been cut open to allow it to incorporate a part extract, and an *open part in backbone* is the same thing for a part in backbone. Note that the process used for incorporating a part extract into an open backbone or open part in backbone (e.g., ligation during assembly) will often not be the same as the process for inserting a part insert into a backbone (e.g., as part of a synthesis order).

Note that the backbone of the parts that are being assembled may or may not be the same as the backbone of the composite part in the backbone, depending on the specifics of the assembly process. For example, in a typical BioBricks assembly, one part in the backbone is cut to produce a part extract, while the another is cut to form an open part in the backbone, and their combination forms a composite part in the backbone of the second part. For a typical MoClo Level 1 assembly, on the other hand, the backbone of the product differs from the backbone of the inputs, as the assembly cuts a set of part extracts out of parts in Level 0 backbones and then combines them with an open Level 1 backbone to form the composite part in backbone. Note also that in many cases, the composite part in the backbone may itself be used as an input to another assembly, forming an even larger composite part, e.g., using MoClo Level 1 products for a MoClo Level 2 assembly.

Assembly. An *assembly* is a plan for combining a set of input parts in order to produce an output of either a single composite part or a library of composite parts. The inputs and output may or may not include backbones depending on the specifics of the assembly. An assembly plan should contain enough information to allow it to be executed by the selection of appropriate laboratory protocols.

Representation in SBOL3. All of the concepts captured by the terminology above can also be represented explicitly using SBOL, which enables the development and use of software tools for working with parts and assembly plans in silico. Specifically, we use the current version, SBOL3,[6] which allows for a cleaner and more succinct representation than would be possible with prior versions of SBOL.

Review of Key SBOL3 Concepts and Notation. We begin with a brief review of key SBOL3 concepts and notation used in the representation. SBOL types, predicates, and values will be shown in monospace font, while examples in figures will be shown using simplified UML diagrams. For additional detail, see the SBOL 3.1 specification.[8]

SBOL3 uses an object model based on the Resource Description Framework (RDF),[9] a representation of knowledge in the form of subject/predicate/object graphs. In the RDF, objects are named and referenced using URLs (or more generally IRIs). Ontology terms, also in the form of URLs, are used to express information about the types and roles of biological entities and interactions. SBOL makes use of a number of standard biological ontologies, including the Systems Biology Ontology (SBO)[10] for describing material types and interactions, and the Sequence Ontology (SO)[11] for providing information about sequence features. Terms from ontologies will be indicated by prefixing them with the abbreviation for the ontology (e.g., `SBO:DNA` is the Systems Biology Ontology term for DNA). Note that whenever an ontology term is specified, it should be taken to indicate either that term or any child term: for example, saying that a replicating backbone should have a `role` of a `SO:vector_replicon` means that it can also be a `SO:plasmid_vector` since the Sequence Ontology defines every `SO:plasmid_vector` to be a type of `SO:vector_replicon`.

The SBOL classes that are used here for representing parts and assemblies are as follows:

- A Component represents either a physical entity, such as the design for a DNA construct, or a functional entity, such as a network of chemical reactions. The type property indicates what sort of entity the Component is, a `role` property can provide information on its expected biological function (e.g., promoter, CDS, and restriction site), and a link to a `Sequence` object can provide DNA sequence information.
- Structural information about a `Component` is encoded using several kinds of `Feature`:
  - A `SequenceFeature` indicates a `role` for some `Location` in the `Sequence` of the `Component`, e.g., a restriction site.
  - A `SubComponent` indicates the inclusion of an `instanceOf` a `Component` within a design. Context-specific roles can be added by giving the optional `roleIntegration` property the value `mergeRoles`, e.g., to mark the part insert in a part in backbone construct. If not all of the included component is present, this can be indicated using a `sourceLocation`, e.g., a MoClo Level 0 part in backbone includes all of the Level 0 destination vector except for the drop-out sequence that was replaced by the inserted part.
  - A `LocalSubComponent` can be used as a placeholder for transient constructs, e.g., a part extract produced by a digestion to be consumed by a ligation.
- Topological information is encoded using a `Constraint`, whose `restriction` property indicates

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655

the relationship between a subject and an object Feature, including ordering (strictlyPrecedes), containment (contains) and junctions (meets, overlaps).

- Reactions are encoded using an Interaction, where each involved species is described with a Participation object indicating the role of the participant species, e.g., a digestion reaction in which an enzyme releases a part extract from a part in backbone.
- The input and output species of a network of reactions can be indicated using an Interface, e.g., two input parts assembled to form an output composite part.
- The provenance ontology PROV-O[12] is used to record how entities were produced: a prov:Activity can indicate that a Component was created (prov:wasGeneratedBy) using (prov:Usage) another prov:entity in some particular way (prov:hadRole), e.g., a composite part being produced using a particular assembly plan.

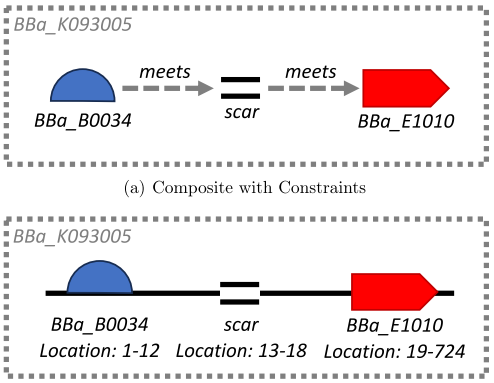
Part Cores, Composite Parts, Assembled Parts, and Scars. Any part of the core or part is represented by an SBOL component with a type property of SBO:DNA. As is standard with SBOL, the general function of the part or part core (e.g., promoter, CDS, composite function) is indicated by a Sequence Ontology term on the role property. Likewise, structural information is indicated using Feature objects, plus optional Constraint objects to indicate the order of the Features and one Sequence object if the sequence of the part is known.

For part of the core, the Sequence must be known a priori. For example, the iGEM part BBa_E1010 is a part core for an mRFP reporter, going from start codon to stop codon plus a trailing barcode. This part core, for which an SBOL representation may be found at https://synbiohub.org/public/igem/BBa_E1010, is a Component of type SBO:DNA with the role SO:CDS, and links to a Sequence with 706 base pairs.

For parts, on the other hand, Sequence might be known or might be left for later calculation from its Features and Constraints. For example, a composite part might initially be specified as a set of Constraints expressing assembly order and later have its sequence calculated from its assembly plan, while an intermediate stage such as a part extract might never actually need to have its full sequence explicitly calculated.

When representing a composite part, it is important to include the assembled parts in its description as well as any scars, such as those produced by BioBrick or MoClo assembly. An assembled part is represented by a SubComponent with an instanceOf property linking to the Component representing the part that was used as an input for the assembly, and a sourceLocation property used to select only that portion of the input part that is actually present in the composite. A scar, on the other hand, is represented by a SequenceFeature with role SO:restriction_enzyme_assembly_scar that either meets or overlaps with an assembled part on both its 5′ and 3′ sides (as indicated by either Constraints or Locations).

For example, BBa_K093005 is a composite part combining the BBa_B0034 ribosome entry site and the BBa_E1010 mRFP coding sequence. It is represented by a Component of type SBO:DNA with the generic "composite part" role SO:engineered_region. Figure 2 shows how the structure of this composite part can be represented using either Constraint



(a) Composite with Constraints

(b) Composite with Locations

Figure 2. Composite parts, such as this representation of the iGEM composite part BBa_K093005, which consists of the BBa_B0034 ribosome entry site followed by the BBa_E1010 mRFP coding sequence, can be represented using SBOL Constraint relations to express the order of their features (a) and/or specific Location values if the composite Sequence has been calculated (b). Additional data model details are shown in Figure S1.

or Location, as well as the inclusion of a SequenceFeature of type SO:restriction_enzyme_assembly_scar for the BioBricks RBS/CDS "TACTAG" scar left between the two parts as a side effect of BioBricks assembly.

Finally, a composite part also includes at least one prov:wasGeneratedBy link to a prov:Activity describing an assembly plan (see below). Note that if there is no assembly plan, then the part is not a composite part and that there can potentially be multiple options for assembly plans since prov:wasGeneratedBy can have multiple values.

Backbones, Insertion Sites, and Drop-Out Sequences. A backbone, like a part, is represented by a SBOL Component with a type property of SBO:DNA. Unlike a part, however, a backbone does not necessarily need to be linked to a Sequence since a backbone is not always retained in the final construct and can sometimes be used "blindly" like a reagent in the assembly process. It is still preferable to have a Sequence for a backbone, however, in order to facilitate better planning, debugging, and quality control.

The role of a backbone depends of its nature: a replicating backbone should use SO:vector_replicon or one of its children (e.g., SO:plasmid_vector), while a nonreplicating backbone, such as used in linear fragments with flanking sequences for restriction, is simply an SO:engineered_region. As with any Component, other information about the structure of the backbone, including insertion sites and drop-out sequences, is indicated with Feature and Constraint objects.

In particular, a backbone must have at least one insertion site, which is indicated using a SequenceFeature with the role SO:insertion_site. Note that circular plasmids are often described with the insertion site at the start/end of the sequence, in which case it is generally preferable to indicate its location as the origin (zero), rather than at the end of the sequence. Likewise, a drop-out sequence should be indicated by a Feature with a role of SO:deletion, e.g., SubComponent for expressing a selection marker. For

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655

simplicity, it is preferable to place the insertion site corresponding to a drop-out sequence at its start, rather than inside of it or at its end, since then the coordinate of the insertion site will not change after the drop-out sequence is removed.

Finally, note that some backbones may include "built in" flanking sequences for assembly on either side of their insertion sites, while other backbones will assume these should be incorporated in the part insert. If there are flanking sequences, they can be indicated with a `SequenceFeature` or `SubComponent` with the role `SO:restriction_enzyme_region`. More specifically, in the case of most digestion-based assembly methods, the roles will typically be `SO:restriction_enzyme_recognition_site` and `SO:sticky_end_restriction_enzyme_cleavage_site`. For example, Figure 3 shows an example of a backbone, the iGEM plasmid vector pSB1C3, which is a high-copy plasmid that includes the prefix and suffix of the BioBrick RFC 10 assembly standard (BBa_G00000 and BBa_G00001, respectively) and an insertion site between these flanking sequences, designated as the zero coordinate of the circular plasmid sequence.

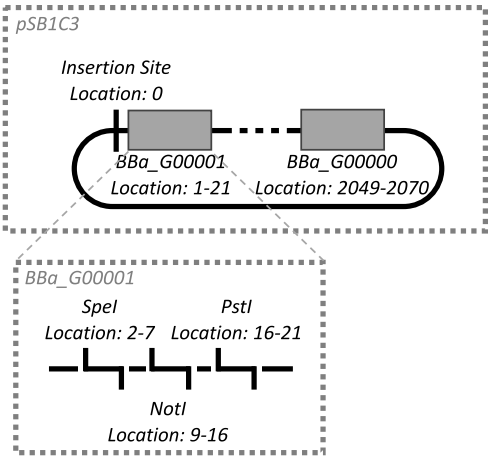


Figure 3. Example of a backbone: the iGEM pSB1C3 plasmid vector is a backbone with its insertion site defined as its origin, and BioBricks flanking sequences BBa_G00000 and BBa_G00001 on either side of the origin, each of which includes enzyme recognition and cutting sites. For simplicity, the contents of BBa_G00000 are omitted, as are all other features besides the insertion site and flanking sequences. Additional data model details are shown in Figure S2.

Part in Backbone, Part Inserts, and Part Extracts. A part in the backbone is represented by a `Component` with at least two features: a `SubComponent` for the part insert and another `SubComponent` for the backbone, along with `Constraint` and/or `Location` information to place them with respect to one another. The `SubComponent` for the part insert should also indicate that it is the part insert by adding a `role` of `SO:engineered_insert` and a `roleIntegration` value of `mergeRoles`. When there is a drop-out sequence in the backbone, its `SubComponent` uses the `sourceLocation` property to exclude the drop-out portion of the backbone sequence from the part in backbone.

If the backbone includes flanking sequences, as in the pSB1C3 example above, then the part insert can simply be a part core since the backbone is providing the assembly interface. If the backbone does not include flanking sequences, however, then the part insert needs to provide the assembly interface and should be a `Component` with role `SO:engineered_insert` that includes both the part core and its flanking sequences as `Feature` objects, using the same roles for the flanking sequences as described above with backbones.

For example, Figure 4 shows a part in backbone constructed by using the part core BBa_E1010 as a part insert into the pSB1C3 plasmid vector. Note that in this case, the part insert does not on its own have any assembly interface but rather gains it from the context of its insertion into the pSB1C3 backbone.

Complementarily, the representation of a part extract from a part in the backbone is much like that of a part insert. If the part extract has no additional flanking sequences added to it, then the part extract is simply the `Component` for the part core. More typically, however, it will be flanked by 5′ and 3′ sticky overhangs, which is represented by a `Component` that includes both the part core and these flanking sequences as `Feature` objects with the corresponding SO terms. For

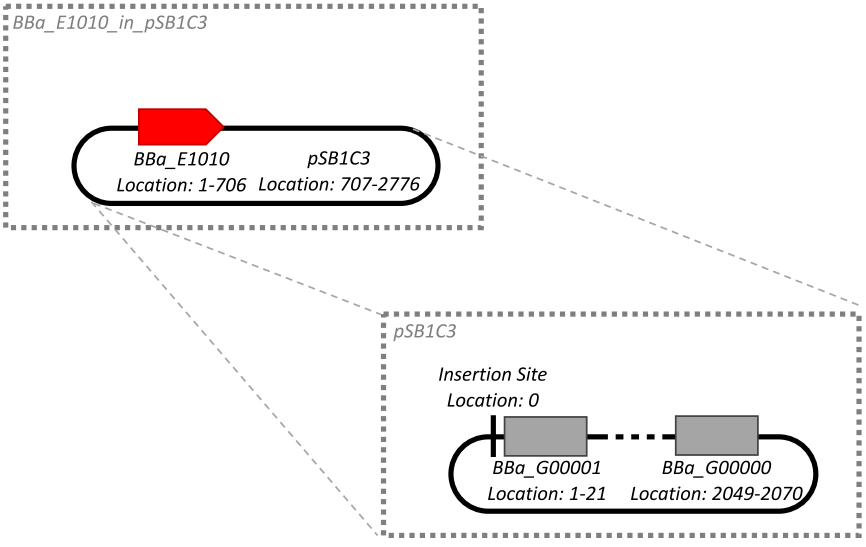


Figure 4. Example of a part in the backbone constructed by inserting part core BBa_E1010 into the pSB1C3 backbone, omitting substructure details for simplicity. Additional data model details are shown in Figure S3.

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655

example, Figure 5 shows the part extract produced by using the XbaI and *Pst*I restriction enzymes to digest the BBa_E1010 part

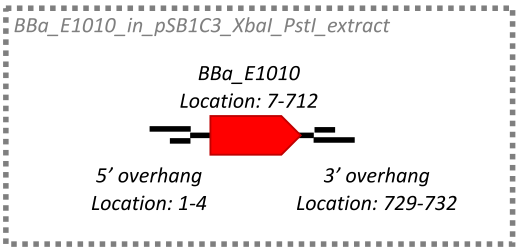


Figure 5. Example of a part extract for the digest of BBa_E1010 out of the part in backbone in Figure 4, in which enzyme digestion causes the part core to become flanked with 5′ and 3′ sticky overhangs. Additional data model details are shown in Figure S4.

core out of the part in the backbone shown in Figure 4. Finally, an open backbone and open part in backbone are represented in the same way as a part extract, except that an open backbone includes a `SubComponent` that is a backbone and an opened backbone includes both a backbone `SubComponent` and also a `SubComponent` that is an `SO:engineered_insert`.

Assembly. An assembly plan, describing the steps to construct a composite part from input parts, may be represented in SBOL using a `prov:Activity` object to link between the `Component` for the composite part and a "reaction network" `Component` that describes the sequence of reactions for producing the composite part from a set of input parts.

A `prov:Activity` object is a rather general way of describing the history of materials, so several property values need to be set in order to allow it to express specifically the idea that the reaction network is the assembly plan for the composite part. To indicate that the `prov:Activity` provides an assembly plan for the composite part, it is given two `type` values (`sbol:design` and `sbol:assemblyPlan`) and linked from the `prov:wasGeneratedBy` property of the composite part. To link to the reaction network, the `prov:Activity` links to a `prov:Usage` that says the reaction network `Component` provides the design for this assembly plan: specifically, the `prov:Usage` has a `prov:entity` property linking to the reaction network and a `prov:hadRole` property with the value `sbol:design`.

To represent the network of assembly reactions, the assembly plan `Component` includes a `SubComponent` for the composite part or core and another `SubComponent` for each part or core that is planned for inclusion in the composite. These `SubComponent` objects are the `Interface` for the assembly, with the composite designated as the `output` and its components designated as the `input`. For assemblies like BioBricks or the various Type IIS assemblies, however, the actual reactions will generally start with parts in backbone, which are first digested and then ligated to produce the composite part in backbone. For this reason, if the `input` and `output` are expressed as part cores rather than parts in backbone, then each part core should be linked to its corresponding part in backbone using a `Constraint` with `type` value `contains`, so that it is easy to trace from the abstract part core inputs to the parts in the backbone that are actually needed for the reactions.

The actual assembly reactions are represented with a set of `Interaction` objects describing digestion to produce part extracts, open backbones, and/or open parts in backbones, followed by ligation of these products to produce one or more composite parts. Specifically, a digestion step is represented by an `Interaction` of type `SBO:cleavage`. Each input vector or enzyme is linked to the digestion using a `Participation` with a `role` property of `SBO:reactant` and a `participant` property linking to the `Feature` for the vector or enzyme. Likewise, the part extracts, open backbones, or open part in backbones produced by the digestion are linked using a `Participation` with a `role` of `SBO:product`. A ligation step is the same as a digestion step, except that the `type` of the interaction is `SBO:ligation` and part extracts, open backbones, and/or open part in backbones are the reactants, while the composite part is the product. These reactions can be validated by checking whether the features with `SO:restriction_enzyme_region`

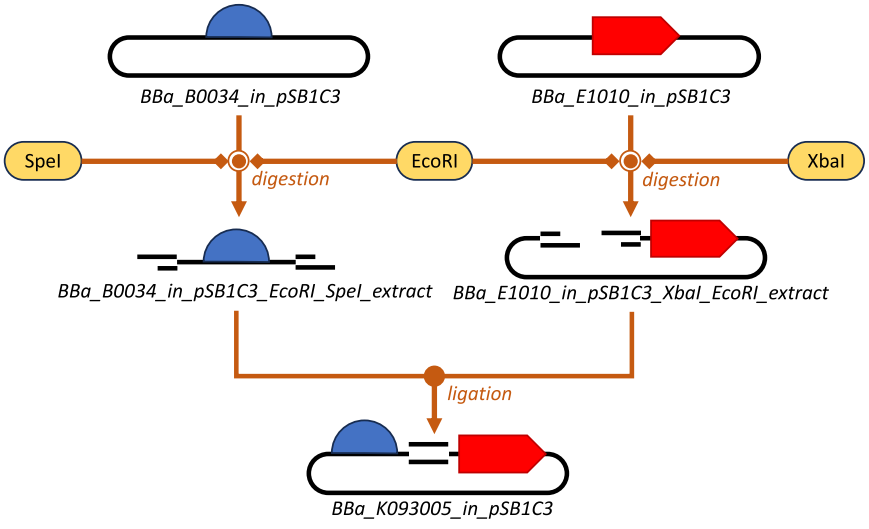


Figure 6. Assembly example UML diagram with flanking sequences in the Backbone and Locations. The contents of BBa_G00000 and BBa_G00001 are purposefully omitted for simplicity of presentation. Objects are color-coded based on what they represent: orange is for the assembly plan, blue is for backbone, green is for part in backbone, yellow is for part core, pink is for part extract, and magenta is for open part in backbone. Additional data model details are shown in Figure S5.

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655

in the reactants and products have appropriate patterns for the enzymes.

For example, Figure 6 shows how composite BBa_K093005 can be produced through a BioBricks assembly of BBa_B0034 and BBa_E1010. Specifically, extraction of BBa_B0034 from its the pSB1C3 backbone is represented by a digestion with the *Eco*RI and SpeI restriction enzymes as participants, while the opening of the BBa_E1010 part in pSB1C3 to accept an insertion is represented by a digestion with the *Eco*RI and XbaI restriction enzymes as participants. The resulting part extract and open part in backbone, with their appropriate overhangs, are then represented as participants in a ligation to form BBa_K093005 in the pSB1C3 backbone.

Note that while many composite parts can be described with just one digestion/ligation stage, an assembly plan may have any number of digestion and ligation stages to produce the ultimately intended composite part. Likewise, an assembly plan may produce multiple composite parts outputs (e.g., in the case of shared intermediates or library assembly), in which case the `Interface` will have multiple `output` values.

Representation Use Patterns. The representation described above enables straightforward automation for a number of common synthetic biology workflows with DNA designs. Here, we exemplify these patterns by providing specifics about how the representation can be used to support several common workflows.

Maintaining a Catalog of Available Parts in Backbones. A catalog of available parts in backbones, such as would be found in the iGEM distribution, can be implemented in terms of two SBOL `Collection` objects, one for all of the part cores and one for all of the parts in the backbone (possibly annotated with additional information, such as the location that the part in the backbone should be able to be found in a distribution plate). When planning a design or assembly, the part in the backbone collection can then be queried to find all of the part in backbones containing a given part core by searching the part in the backbone collection for all `Components` that contain a `SubComponent` with role `SO:engineered_insert` whose `instanceOf` property either links to the desired part core or else links to a part insert `Component` that contains it as a `SubComponent`.

Exporting Sequences for Synthesis. Given a `Collection` of parts in backbone, the sequences that need to be synthesized into each backbone can be exported by searching each part in backbone `Component` for all of the `SubComponent` objects whose effective role in the context contains a value of `SO:engineered_insert`. Each such sequence needs to be synthesized and inserted at the adjacent insertion site on the backbone.

Checking Assembly Compatibility for a Part in Backbone. To check the assembly compatibility for a part in the backbone, one needs to check that the part has appropriate flanking sequences and that there are no illegal restriction sites in the part itself as well as in the backbone.

A generic assembly template can be generated as an abstract `Component` with digestion and ligation reactions by using specific restriction enzymes. The parts in the backbone to be assembled, however, are only fully specified in the `SequenceFeature` objects for the flanking sequences that contain the restriction enzyme recognition sites, joined to undefined `LocalSubComponent` feature for the backbone and part core with `Constraint` objects with `type` value `meet`. Likewise, the composite part has a fully specified `SequenceFeature` for the flanking sequences and scars and a `Constraint` with the `typo` value `meet` linking each scar to an adjacent undefined `LocalSubComponent` feature for assembled parts.

To check assembly compatibility for a part in the backbone, one can create an assembly `Component` with `SubComponent` objects for the part and the generic assembly template and add a `Constraint` of type `equals` between the part in backbone and an abstract part in backbone in the template. Once this is done, attempts are made to calculate the sequence of the `Component`: if there is a conflict between the part in the backbone and the flanking sequences in the generic assembly template, then the part in the backbone does not have compatible flanking sequences. Likewise, many assemblies require specific antibiotic resistance genes, which can also be annotated on the abstract part in the backbone in the template and checked for when resolving the `equals` constraint.

Once the sequence has been calculated, check for illegal restriction sites by scanning for all locations with sequences that match the recognition sites in the flanking sequences of the generic assembly template. Any such location that is not fully contained within an appropriately typed `Feature` for a flanking sequence means that the part in backbone is not compatible with the assembly. The fully computed assembly `Component` can then serve as a validation record for assembly compatibility if desired.

Designing a Composite Part via an Assembly Plan. To design a composite part by assembling together parts, use a generic assembly template when checking assembly compatibility. In this case, however, all of the parts to be assembled should be specified, not just the one being checked. Following the equality constraints should then allow the sequence of the composite part to be calculated.

Case Study: Use by the iGEM Engineering Committee. The iGEM competition is a yearly international event in which hundreds of teams of students around the world do synthetic biology projects and then meet to share their projects and be judged at the iGEM Giant Jamboree. One of the key supports for these team projects is the iGEM Distribution, a kit shipped to participating teams that contains, among other things, a large curated collection of DNA parts that teams can make use of in their projects. In 2021, the iGEM Foundation decided to change the way this collection was designed, including a complete rebuild of all parts for the iGEM 2022 Distribution, recurating the parts into a more organized system of "packages," and aiming to improve accessibility, quality, reproducibility, and traceability of kit materials. For this effort, the iGEM Foundation enlisted the aid of the iGEM Engineering Committee, a volunteer organization that supports iGEM by developing and promoting effective practices for engineering biology, along with supporting protocols and tools. The iGEM Engineering Committee is an international collaboration with more than 50 participating members from academic and industrial institutions in dozens of countries (including the authors of this manuscript) and representing diverse areas of synthetic biology experience.

In coordinating across the many participants in the effort to curate DNA parts for the iGEM 2022 Distribution, we quickly encountered challenges due to many miscommunications between collaborators about how sequences related to our build plans. For example, did a sequence already include flanking sequences, was this what should be synthesized, or what it would look like after insertion into a backbone, etc. In order to solve these problems, we developed the shared terminology presented

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655

Part Name	Role	Design Notes	Altered Sequence	Part Description	Data Source Prefix	Data Source
SpCas9	CDS	Source = Addgene 39312	Cutout of Plasmid	Cas9 DNA nuc	Addgene	39312
dSpCas9	CDS	Source = Addgene 39318	Cutout of Plasmid	Nuclease deac	Local Sequence File	39318
hSpCas9	CDS	Source = Addgene 42230. I've added the start codon (at	Cutout of Plasmid	Humanized ve	Local Sequence File	42230
dhSpCas9	CDS	Source = Addgene 61422. Added the start codon (atg) a	Cutout of Plasmid	Humanized ve	Local Sequence File	61422
hLwCas13a	CDS	Source = Addgene 90097. Added stop codon (taa).	Cutout of Plasmid	Humanized ve	Local Sequence File	90097
dhLwCas13a	CDS	Source = Addgene 91905. Added stop codon (taa). Ther	Cutout of Plasmid	Nuclease deac	Local Sequence File	91905
LbCpf1 or LbCas12a	CDS	Source = Addgene 69988. Added stop codon (taa).	Cutout of Plasmid	Humanized ve	Local Sequence File	69988
dLbCpf1 or dLbCas12a	CDS	Source = Addgene 140220. Added stop codon (taa)	Cutout of Plasmid	DNA nuclease	Local Sequence File	140220
dSpCas9_RFC10	CDS	Source = Bba_K2558003	Direct	Cas9 DNA nuc	iGEM registry	K2558003
ABE8.20-m	CDS	Source = Addgene 136300.	Cutout of Plasmid	Evolved SpCa	Local Sequence File	136300
ABE8e TadA-8e V106W	CDS	Source = Addgene 138495. Added annotations based of	Cutout of Plasmid	Evolved SpCa	Local Sequence File	138495
ABEmax	CDS	Source = Koblan et al. (doi: 10.1038/nbt.4172) supplem	Direct	Evolved and optimized SpCas ade		ABEmax
BE4max	CDS	Source = Koblan et al. (doi: 10.1038/nbt.4172) supplem	Direct	Evolved and optimized SpCas cyti		BE4max
SaABE8e TadA-8e V106W	CDS	Source = Addgene 138509. Added annotations based of	Cutout of Plasmid	Evolved SaCa	Local Sequence File	138509
PE2	CDS	Source = Addgene 132775. Added annotations based of	Cutout of Plasmid	SpCas based	Local Sequence File	132775

(a) Example Part Core Specifications

Part/Library Nam	Final Product	Backbone/I	Constraints	Part 1	Part 2	Part 3
Cas variants	TRUE	pSB1C5		D1005	SpCas9, dSpCas9, hSpCas9, dhSpCas9, hLwCas13a, dhLwCas	D1006
Anti-CRISPR	TRUE	pSB1C5		D1005	AcrIIA2_mammalian, AcrIIA2_bacterial	D1006
Nickases	TRUE	pSB1C5		D1005	Cas9n_D10A, Cas9n_H840A	D1006

(b) Example Build Plan Specifications

Figure 7. Example materials from the Excel workbook specifying the CRISPR-Cas collection in the iGEM 2022 Distribution: (a) part cores, with each row specifying one part core, including its name, role, design notes, and accession for retrieval, and (b) build plans, with each row specifying a library of part cores to be flanked with BioBrick assembly adapters (D1005 and D1006) to form a part insert that is to be synthesized and inserted into the pSB1C5 backbone to produce a library of part in backbone plasmids.

above, which has rapidly become the *lingua franca* of communication for collaboration on the iGEM Distribution, as well as for other projects of the iGEM Engineering Committee such as interlaboratory studies, development of new organism-focused part kits, protocol development, and laboratory automation efforts.

The SBOL representation for the terminology was also put to use by the committee in developing the iGEM 2022 Distribution and in the creation of a set of automation scripts for curation assistance and error checking. Specifically, the iGEM 2022 Distribution was implemented using a GitHub repository (https://github.com/iGEM-Engineering/iGEM-distribution), in which each package of parts is curated by manually populating an Excel workbook, typical extracts from which are shown in Figure 7. In the Excel workbook, one sheet contains information about each part core and backbone in the collection, such as its name, role, design notes, and an accession number that will allow the part core or backbone to be retrieved from a local file or imported from a public data source such as SynBioHub[13] or NCBI. The other sheet contains build plan information that incorporates part cores into collections of part inserts, composite parts, and parts in backbone.

In the typical usage pattern, each part of the core is flanked with adapters to make it an assembly-compatible insert, composed together into more complex constructs if desired, and then inserted into a plasmid backbone. The scripts developed by the committee export the material from the sheets into an SBOL representation, using the guidelines presented above to represent the intended collection of parts and their associated build plans, supplemented with the part core and backbone information retrieved from external repositories as needed. The build plans are automatically checked for errors in completeness or coherence and exported into summary README files in Markdown for human auditing, with no proposed change allowed to be merged into the iGEM Distribution plan until it is certified error free and a human reviewer has approved. Finally, an SBOL representation of the complete iGEM Distribution is generated, and the build plans are used to export it into two different forms: a build input in the form of a FASTA synthesis order file and backbone insertion instructions and a build output in the form of GenBank files for the final part in backbone plasmids that are actually shipped in the kit.

During the preparation of the iGEM 2022 Distribution, this shared terminology and supporting SBOL-based automation allowed the iGEM Engineering Committee to collectively curate a large number of parts at a sustained and rapid pace. Following work on preliminaries and early tests during 2021, the main period of development for the iGEM 2022 Distribution took place from January 1st, 2022 to February 16th, 2022, when the first synthesis order was tagged for release. During that period, contributions were made by 15 collaborators at 11 institutions in 8 countries, collectively building the distribution via 87 pull requests (an average of nearly two per day) and 571 commits. At release time, the iGEM 2022 Distribution contained a build plan for 346 parts in the backbone organized into 13 packages, summing to a 493 kilobase synthesis order FASTA, along with another 569 automation-generated intermediates and composites at other stages of the build plan.

All of the Excel workbooks for the iGEM 2022 Distribution, along with the associated build plans and final genetic constructs are available on GitHub at https://github.com/iGEM-Engineering/iGEM-distribution, and Figure 7 shows an excerpt from the workbook for the CRISPR-Cas collection. Note that in the case of this specific build, however, the actual production was done solely by synthesis, and so the scars are inserted into composite constructs in order to ensure that sequences are base-for-base equivalent to the products of a BioBricks assembly (thus ensuring compatibility of data taken from those composites when thus previously assembled), rather than as an actual necessity of a physical assembly process.

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655

The automation tools, scripts, and configuration data used in the production of the iGEM 2022 Distribution are also publicly available in this same repository and in the publicly available libraries that it imports along with the documentation that was used to educate participants in the project on how to use the tools. These materials should be able to serve as a template for others wishing to make use of these same methods, e.g., by forking the repository, deleting the iGEM parts, and using the Excel template provided to build their own collections of parts and composites.

Since the production of the iGEM 2022 Distribution, the iGEM Engineering Committee has continued to make use of both the terminology and tooling to improve the iGEM Distribution and to improve its processes, including automatic synthesizability checks, increased modularity of package development, making scripts more reusable, and breaking the monolithic distribution into a simpler-to-maintain library and dependency model, all of which are currently being deployed in the development of future materials for iGEM collections.

DISCUSSION

We have presented both a set of terminology for describing DNA parts and a set of practices by which that terminology can be formalized into SBOL3 representations of genetic parts at various stages of design, synthesis, and assembly, along with examples of both patterns for using these methods in common synthetic biology workflows and a case study of their usage to support complex international collaborations within the iGEM Engineering Committee.

A standard is only as valuable as its adoption, of course, and the work described above is still too recent to know how widely it will be adopted. So far, at least, the reception of this work has been positive and indicates that the terminology and representation have a good chance of spreading organically through the community. In addition to its ongoing use by the iGEM Engineering Committee, as described above, the terminology and representation have also been circulated to a number of other organizations in the wider synthetic biology community, where the terminology has generally been received as useful and intuitively comprehensible. The representation has also been formally endorsed by the SBOL standards community, which has adopted it as a "best practice" ("BP011: Representation of Parts and Devices for Build Planning") recommended method for describing parts and assembly plans in SBOL. Looking beyond communication to software tooling, a full supporting Python API is in the process of being implemented for the SBOL Utilities library,[14] and multiple groups are investigating its use for coordinating assembly reactions utilizing laboratory automation, either directly into various liquid-handler APIs or via the cross-platform LabOP protocol representation[15]. We thus anticipate that both the terminology and the representations presented in this article will spread more broadly through the synthetic biology community and will be adopted into other tools that include assembly planning.

Extending beyond the scope of this article, while the current terminology and representation have been worked out specifically with regard to Type IIS and BioBricks assembly methods, the same basic framework should extend to other contexts as well. Genomic integration, for example, should be able to be described and represented in the same way as the insertion of a part into a backbone, simply by substituting the genome for the backbone, with the key difference being the description of the locus of insertion. Likewise, the terminology and representation are also likely to extend well to other assembly methods, such as Gibson Assembly[16] or Ligase Cycling Reaction Assembly,[17] though certain details will likely need to be adjusted. Finally, while the terminology and representation presented here have so far been applied only in the context of DNA, they may prove to be useful as a basis for developing analogous and compatible vocabularies and representations for other polymeric molecules such as RNA, proteins, lipids, or polysaccharides.

METHODS

Automation tooling for the iGEM 2022 Distribution case study was implemented using the SBOL3 standard in Python with the SBOL-utilities and pySBOL3 libraries.[14] Import automation was used to access materials from SynBioHub,[13] NCBI GenBank, and the iGEM Parts Registry. Workflows were implemented using GitHub Actions and executed on GitHub cloud resources automatically using push and pull request triggers. Full details are available publicly in the implementation git repository on GitHub at https://github.com/iGEM-Engineering/iGEM-distribution.

ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acssynbio.3c00418.

UML data models for examples (PDF)

AUTHOR INFORMATION

Corresponding Author

Jacob Beal − *Intelligent Software & Systems, Raytheon BBN Technologies, Cambridge, Massachusetts 02138, United States;* orcid.org/0000-0002-1663-5102; Email: jakebeal@ieee.org

Authors

Vinoo Selvarajah − *iGEM Foundation, Cambridge, Massachusetts 02139, United States*

Gaël Chambonnier − *Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, United States;* orcid.org/0000-0002-5606-2089

Traci Haddock − *Asimov, Inc., Boston, Massachusetts 02215, United States*

Alejandro Vignoni − *Synthetic Biology and Biosystems Control Lab, Institut d'Automàtica i Informàtica Industrial, Universitat Politècnica de València, Valencia 46022, Spain;* orcid.org/0000-0001-9977-7132

Gonzalo Vidal − *Interdisciplinary Computing and Complex BioSystems (ICOS) Research Group, School of Computing, Newcastle University, Newcastle upon Tyne NE1 7RU, U.K.*

Nicholas Roehner − *Intelligent Software & Systems, Raytheon BBN Technologies, Cambridge, Massachusetts 02138, United States;* orcid.org/0000-0003-4957-1552

Complete contact information is available at: https://pubs.acs.org/10.1021/acssynbio.3c00418

Author Contributions

Conceptualization, methodology, and investigation: all authors. Writing: original draft: J.B. and G.C.; and review and editing: all authors.

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

A.V. is funded by MCIN/AEI/10.13039/501100011033 grant no. PID2020-117271RB-C21. G.V. is supported by the Newcastle University School of Computing. J.B. and N.R. were supported in part by AFRL and DARPA contract FA8750-17-C-0184. This document does not contain technology or technical data controlled under either the U.S. International Traffic in Arms Regulations or the U.S. Export Administration Regulations.

■ REFERENCES

(1) Shetty, R.; Lizarazo, M.; Rettberg, R.; Knight, T. F. Assembly of BioBrick standard biological parts using three antibiotic assembly. In *Methods in enzymology*; Elsevier, 2011; Vol. *498*, pp 311−326.
(2) Engler, C.; Kandzia, R.; Marillonnet, S. A one pot, one step, precision cloning method with high throughput capability. *PLoS One* **2008**, *3*, No. e3647.
(3) Weber, E.; Engler, C.; Gruetzner, R.; Werner, S.; Marillonnet, S. A modular cloning system for standardized assembly of multigene constructs. *PLoS One* **2011**, *6*, No. e16765.
(4) Sarrion-Perdigones, A.; Falconi, E. E.; Zandalinas, S. I.; Juárez, P.; Fernández-del-Carmen, A.; Granell, A.; Orzaez, D. GoldenBraid: an iterative cloning system for standardized assembly of reusable genetic modules. *PLoS One* **2011**, *6*, No. e21622.
(5) Cai, Y.-M.; Carrasco Lopez, J. A.; Patron, N. J. Phytobricks: manual and automated assembly of constructs for engineering plants. *Methods Mol. Biol.* **2020**, *2205*, 179−199.
(6) McLaughlin, J. A.; Beal, J.; Mısırlı, G.; Grünberg, R.; Bartley, B. A.; Scott-Brown, J.; Vaidyanathan, P.; Fontanarrosa, P.; Oberortner, E.; Wipat, A.; et al. The Synthetic Biology Open Language (SBOL) version 3: simplified data exchange for bioengineering. *Front. Bioeng. Biotechnol.* **2020**, *8*, 1009.
(7) Aldulijan, I.; Beal, J.; Billerbeck, S.; Bouffard, J.; Chambonnier, G.; Ntelkis, N.; Guerreiro, I.; Holub, M.; Ross, P.; Selvarajah, V.; et al. Functional synthetic biology. *Synth. Biol.* **2023**, *8*, ysad006.
(8) Buecherl, L.; Mitchell, T.; Scott-Brown, J.; Vaidyanathan, P.; Vidal, G.; Baig, H.; Bartley, B.; Beal, J.; Crowther, M.; Fontanarrosa, P.; et al. Synthetic biology open language (SBOL) version 3.1.0. *J. Integr. Bioinf.* **2023**, *20*, 20220058.
(9) McBride, B. The resource description framework (RDF) and its vocabulary description language RDFS. In *Handbook on ontologies*; Springer, 2004; pp 51−65.
(10) Courtot, M.; Juty, N.; Knüpfer, C.; Waltemath, D.; Zhukova, A.; Dräger, A.; Dumontier, M.; Finney, A.; Golebiewski, M.; Hastings, J.; et al. Controlled vocabularies and semantics in systems biology. *Mol. Syst. Biol.* **2011**, *7*, 543.
(11) Eilbeck, K.; Lewis, S. E.; Mungall, C. J.; Yandell, M.; Stein, L.; Durbin, R.; Ashburner, M. The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biol.* **2005**, *6*, R44.
(12) Lebo, T., Sahoo, S., McGuinness, D., Belhajjame, K., Cheney, J., Corsar, D., Garijo, D., Soiland-Reyes, S., Zednik, S.; Zhao, J. PROV-O The PROV Ontology, 2013. https://www.w3.org/TR/prov-o/ (accessed July 1, 2023).
(13) McLaughlin, J. A.; Myers, C. J.; Zundel, Z.; Mısırlı, G.; Zhang, M.; Ofiteru, I. D.; Goni-Moreno, A.; Wipat, A. SynBioHub: a standards-enabled design repository for synthetic biology. *ACS Synth. Biol.* **2018**, *7*, 682−688.
(14) Mitchell, T.; Beal, J.; Bartley, B. pySBOL3: SBOL3 for Python Programmers. *ACS Synth. Biol.* **2022**, *11*, 2523−2526.
(15) Bartley, B.; Beal, J.; Rogers, M.; Bryce, D.; Goldman, R. P.; Keller, B.; Lee, P.; Biggers, V.; Nowak, J.; Weston, M. Building an Open Representation for Biological Protocols. *ACM J. Emerg. Technol. Comput. Syst.* **2023**, *19*, 1−21.
(16) Gibson, D. G.; Young, L.; Chuang, R.-Y.; Venter, J. C.; Hutchison, C. A.; Smith, H. O. Enzymatic assembly of DNA molecules up to several hundred kilobases. *Nat. Methods* **2009**, *6*, 343−345.
(17) Kok, S. d.; Stanton, L. H.; Slaby, T.; Durot, M.; Holmes, V. F.; Patel, K. G.; Platt, D.; Shapland, E. B.; Serber, Z.; Dean, J.; et al. Rapid and reliable DNA assembly via ligase cycling reaction. *ACS Synth. Biol.* **2014**, *3*, 97−106.

https://doi.org/10.1021/acssynbio.3c00418
ACS Synth. Biol. 2023, 12, 3646−3655