Appendix A

This appendix describes how the Descript dialect *differs* from the D dialect.

Syntax of Descript

Descript has all of the syntactical clauses of, in the same order as (with one exception), D. But, instead of symbols and columns, it has **label**s. The only symbols allowed are arithmetical.

General Formats

Every line in a product, resource, or interval definition is a descriptive statement. There are up to three functional segments in each statement: the **component segment**, the **source segment**, and the **management segment**.

Each label is delimited by a colon and a space.

```
label: element
```

Multiple elements are separated by a semicolon and a space.

```
label1: element1; label2: element2
```

When a label is part of an element of another label, the first label does not need a colon.

```
label1 element1label: element1
```

A comment is denoted by a statement starting with the label `comment`.

```
comment: In normal writing, an asterisk indicates a note following its subject.
```

Some statements start with a keyword or keyphrase, delimited by a semicolon and a space, for context.

```
keyword; label: element
```

Management Definitions

An implicit library specification has the label `library` and the library name, followed by the label `product set` and one or more set names.

```
library: D : Math; product set: Circle
```

Each product part completion and resource part access implicit interval specification begins with the label `store` and the product name or resource name, respectively. This is followed by the label `part` and the part name. The intervals for completions and accesses can be centrally referenced, with the labels `completion` and `access`, respectively. (This statement's clause order is different from D.)

```
store: Keyed Report; part: Keyed Report Line;
  completion: Keyed Report Line Product;
  access: Keyed Report Line Resource
```

An in-set alias entry has the label `alias` and the alias interval set name, followed by the label `translation` and the translation.

```
alias: Option Selection; translation: selectOption
```

Major Definitions (Header Statements)

A characteristics definition is indicated by the label `characteristics`.

```
characteristics: temporary
```

A product's interval definition is denoted by a header statement that has the interval name following the label `interval`. It's followed by the interval's return component definition, which is the label `return type`, followed by a component type. If there is no return component, then that is denoted by a `return type` label with no return type.

```
interval: Picture Validation
  return type: bool
```

and

```
interval: Context Acquisition
  return type:
```

For a managed interval, each component definition is begun by the label `label`, for the interval's invocation, then each management component is used as a label. The component type follows the label `type`.

```
interval: Picture Validation;
  label: Width; type: int; label: Height; type: int
  return type: bool
```

Minor Definitions (Body Statements)

The first (component) segment can be a component definition, a component reference, or a product part reference.

The second, or source, segment can be a component reference (or an explicit value), a combination, or an interval reference. These are not allowed when the first segment has a product part reference.

The third segment—the management segment—can contain an incrementation designation, which is a resource reference or a resource part reference, or it can contain an arithmetic expression or an interval reference.

When either (or both) of the first two segments is non-null, and the third is a resource (or part) reference, this is a compound statement.

Component Definitions

A component definition has the label `component` and the component name, followed by the `type` label. A component type is optionally after the `type` label. An initial value is then optionally preceded by the label `value`.

```
component: Response; type:;
```

is equivalent to

```
component: Response; type: string; value: null
```

A component group definition has the label `group` and the component group name. This is followed by the group member component definitions, followed by a statement with just the keyphrase `group end`, all of which can be indented. Because of the allowed indentations here, each definition must be followed by a semicolon.

```
group: Keyed Report Line
    component: Sequence Number; type: smallint; value: 1
    type: smallint
    component: Department Number; type: smallint
    group end
```

A component series definition follows the group definition format, with the label `series` and the keyphrase `series end`, but the series name is optional; and an optional size name, preceded by the label `size name` and a size value, preceded by the label `size value`, are part of the series definition statement.

```
series: Report Page Spacing; size name: Lines Per Page;
  size value: 60;
  component: Character Count; type: smallint; value: 2;
  series end
```

and

```
series:; size name: Estimated Maximum Employees; size value: 100;
  component: Employee Number; type: int; series end
```

String definitions can follow the series format, except here simply the type string (or nothing) is the member definition, and the optional `value` clause applies to the entire series.

```
series: Status Message; size name: Characters Per Line;
  size value: 132;
  value: "Products were completed without incident."
```

translates to

```
series: Status Message; size value: 132; component: character;
  type: char;
  value: "Products were completed without incident."
```

Component References

A component reference has the `component` label and the component name.

```
component: Response
```

Any component reference can list multiple components, separated by commas.

```
component: Coordinates
```

can be

```
component: Last X, Last Y
```

A unique group member can be referenced by just its name. Any nonunique member must be preceded by all of its immediate owner levels until one of those is unique. This reference must begin with the `group` label and each level must be preceded by the label `sublevel`.

```
group: Original Report Line; sublevel: Print Control;
  component: Another Component
```

A series member reference must begin with the `series` label followed by all of its series levels, separated by commas, then the member. No specified series levels indicate all of the occurrences of the member.

```
series: Week Number, Employee Counter; component: Hours Worked
```

and

```
series:; component: Hours Worked
```

When a series is in a group, or vice versa, a reference to a low-enough member requires a mixture of syntax. The `series` label is needed any time the level type switches to series. The `group` label is needed any time the level type switches to group. In a string reference, the character member is implied.

```
series: Response; group: Character Number
```

Source References

A source reference has the same labels as the component references, but the first label is prefixed with the label `source`. For source components, the standard is actually just the `source` label.

```
source component: Response
```

translates to

```
source: Response
```

```
source group: Original Report Line; sublevel: Print Control;
  component: Another Component
```

and

```
source series: Week Number, Employee Counter;
  component: Hours Worked
```

Combinations

Source combinations have just the `source` label.

```
component: Group Mark Column; source: Group End Column +
```

and

```
component: Group Begin Column; source: + Group Mark Column
```

and

```
component: Column Counter; source: +
```

and

```
component: Range Message; source: "The number is between"
  Low Number " and " High Number "."
```

Product Part References

A product part completion begins with the `completion` label and ends with the keyphrase `completion end`. Its simplest form has no product part processing statements.

```
completion: Report Line
comment: all of the components populated
completion end
```

and

```
comment: components populated in various places
completion: Report Line; completion end
```

Interval References

An interval reference is delineated by the label `jump`, followed by the interval label, followed by the interval name. Function intervals and extension intervals are referenced with their respective keyword preceding the interval label, just as in the definition. The interval label is only for clarifying.

```
jump: Attribute Check
```

and

```
component: Pixel Count; jump function: Picture Size
```

A managed interval reference has the management component names as labels.

```
component: Goal Number; type:;
  jump: Random Number; Low Number: 1; High Number: 100;
  Decimal Places: 0
```

Interval reference nesting is very clearly identified by a two-dimensional step. For the nested portion, in place of the component is the keyword `nested`. For the interval reference portion, in place of the nested interval reference is the keyword `nesting`.

```
nested; jump: string length; String: Original Value
component: Component Value; jump: string segment;
  String: Report Layout;
  Begin Position: Column Counter; End Position: nesting
```

Specific Usages

A specific usage is described with the label `for` followed by a condition.

```
jump: Header Skip; for: Print Control = Page Break
```

The condition controls are the labels `and` and `with`. Parentheses are used for grouping.

```
jump: Group Code Process; Group Type: Group Type;
  for: Group Fields;
  with: (Column Counter = Group Begin Column;
   and: Column Counter = Group End Column +)
```

A parallel specific usage condition is designated by each of the involved statements starting with the keyword `parallel`.

```
parallel; jump: Main Line Process; for: Special Attribute
  = Main Line Attribute
parallel; jump: Redefine Process; for: Special Attribute
  = Redefine Attribute
parallel; jump: Table Process; for: others
```

Resource Part References

A resource part access has the access label.

```
access: Report Line
```

Repetitions

A repetition is described with the appended label `for every`, followed by a condition.

```
for every: Print Control \= Double Space;
  access: Report Original Line
```

Instead of a resource access, the Rrepetitions notation can be an arithmetic operator and incrementation amount or a nonreturn interval reference.

```
for every: Column Counter \> Line Width; source: Column Counter + 1
```

Specific Blocks

Specific usages and incrementations can have immediate follow-up steps. Each of these starts with the keyphrase `follow up`.

```
component: Source Code; source: Empty Layout Message;
  for: Source Code = Not Written
follow up; part: Source Code
```

Status Management

Status monitoring has two forms: single statement monitoring and multiple statement monitoring. The multiple form starts with the keyphrase `monitor on` for an on or the keyphrase `monitor off` for an off, and the single form starts with the keyphrase `monitor next`. These keyphrases can be the keywords `on`, `off`, and `next`, respectively.

```
monitor next; part: Unexpected End of File;
  for: status = End of File
```

and

```
on; part: Unexpected End of File; for: End of File
```

and

```
off; for: End of File
```

Interval Orientation

The first line is simply the label `parent` and the name of the parent class.

```
parent: Shape
```

An implementation of an interface-only has the label "*interface*", followed by one or more interface-only class names, separated by commas.

```
interface: Error, Message
```

An object definition has the component label and the label `object`—and, optionally, the label `locater`—and the object name. This is followed by the `type` label—and, optionally, the label `class`—and the class name. A generalized object is denoted by the label `ancestor` and an ancestor name, between the two other entities. This is followed by a reference to an initialization interval, implicitly of the class. A specializable type is denoted by the keyword `specializable` preceding the the `type` label.

```
component object: Shape; ancestor: Shape; type: Circle;
  jump: initialization; Radius: Radius
```

An object reference is denoted by the `object` label followed by the object name. A class reference is denoted by the `class` label followed by the class name. A library reference has the `library` label followed by the library name. Again, any interval reference begins with the `jump` label; here, the `interval` label is required.

```
component: Radius; jump object: Focus Circle; interval: Radius
```

and

```
component: Area; jump library: D : Math; class: Circle;
  interval: Area; Radius: Radius
```

A type change is denoted by a component reference followed by the label `new type` and the new type, followed by the source.

```
component: Focus Shape; new type: Circle; source: Previous Shape
```

Interaction Properties

The usability is defined with the label `usability`.

```
component: Pi; type:; usability: any; value: 3.14159
```

and

```
interval: Area; label: Radius; type: float
  return type: float; usability: any
```

The rigidity is defined with the label `rigidity`.

```
component: Pi; type:; usability: any; rigidity: execution;
  value: 3.14159
```

and

```
interval: Area; label: Radius; type: float
  return type: float; usability: any; rigidity: execution
```

A variable component's usability can be only self, so the `self` keyword is not specified. Implicit standard access and update intervals can be designated with the `out` and `in` keywords, respectively, separated by a comma, as the usability.

```
component: Radius; type: float; usability: in, out; rigidity: variable;
```

and

```
component: Radius; type: float; usability: in; rigidity: variable;
```

and

```
component: Radius; type: float; usability: out; rigidity: variable;
```

and

```
component: Radius; type: float; usability:; rigidity: variable;
```

A block interval can be designated as the usability, with the added `jump` label and referencing interval name.

```
interval: Area; label: Radius; type: float
  return type: float; usability: jump: Shape Statistics; rigidity: execution
```

Advanced Elements

Chaining is possible with the two-dimensional nesting step. For the object creation portion, in place of the object name is the `nested` keyword. For the interval reference portion, in place of the object creation is the `nesting` keyword.

```
nested; jump class: Circle; interval: initialization; Radius: New Radius
jump object: nesting; interval: Rendering
```

Locations cannot be compared with the equal sign; they must be compared with either of two generic intervals: "same object as", which is implemented (in generic) with the equal sign, or "equivalent to", which is an empty interval, because it's different for each class. For ease of use, these generic intervals can be referenced with standard label syntax.

```
component: Same Values Count; source: +;
  for object: Current Object = object: Desired Object
```

is illegal.

```
component: Same Values Count; source: +;
  for object: Current Object; equivalent to: Desired Object
```

is clear and is equivalent to

```
component: Same Values Count; source: +;
  for jump object: Current Object; interval: equivalent to;
  other object: Desired Object
```

and

```
component: Same Object Count; source: +;
  for object: Current Object; same object as: Desired Object
```

is equivalent to

```
component: Same Object Count; source: +;
  for jump object: Current Object; interval: same object as;
  other object: Desired Object
```

An inner class is defined with a header statement of the label `inner class` and the inner class name. It is referenced with the label `inner` and the inner class name.

```
inner class: Mouse Monitor
```

and

```
jump inner: Mouse Monitor
```

and

```
jump class: Inquiry Form; inner: Mouse Monitor
```

An interval location is copied with the label `location` added to the `interval` label.

```
component object: Action Monitor; source interval location: Action Performed
```

Aspect Orientation

The called interval has inlet names, each preceded by the label `inlet`, to mark the inlet. And the call is continued with the keyword `injections`. The injection begins with the label `injection` and injection name followed by the statements to inject. And the call is ended with the keyphrase `injections end` on a line by itself.

```
interval: Server Interval
comment: some core statements
inlet: Security -- Login
comment: more core statements
```

```
interval: Client Interval
jump object: Server; interval: Server Interval; injections
injection: Security -- Login
jump object: Third-party; interval: User Validation
injections end
```

Injections can be in an interval outside of an interval reference. Any noninjection code must be marked by a preceding line that is just the `injections end` keyphrase. They can stand alone or in a group, in an injection set, denoted by a file name prefixed with `injection set`. It is referred to (functionally copied), anywhere in a class or other injection set, with the label `injection set` and the nonprefixed injection set name.

```
injection set: Security
```

Reserved Words

Obviously, Descript relies very little on punctuation and very much on words, so it has many more reserved words than D.

Labels

Individual Identifiers

access	inner class	product set
alias	interface	return type
ancestor	interval	rigidity
characteristics	jump	size name
class	label	size value
comment	library	source
completion	locater	store
component	location	sublevel
injection	new type	translation
injection set	object	type
inlet	parent	usability
inner	part	value

Controls

and	for	same object as
equivalent to	for every	set return

Keyphrases

Scopes

completion end	monitor next	next
follow up	monitor off	off
group end	monitor on	on
injections	nested	parallel
injections end	nesting	series end

Controls

return	set return

Modifiers

addition	function	removal
all opposite	incomplete	replacement
aspect	interface only	restart
extension	last	specialized
finish	or	start
first	others	

Property Values

any	execution	self
ascendants	library	temporary
definition	permanent	variable

General Values

false	null	parent
initialization	object	true

Appendix B

This appendix describes the *additional* features of the Desc dialect.

Syntax of Desc

The simplest explanation of Desc's syntax is that it allows the column separators and punctuation of D, and the labels and added keyphrases of Descript. The punctuation can be used *in place* of any of the corresponding keywords, but they cannot both be used for the same element. In Desc, the *reserved* labels do not require the colon, and all of the reserved words and managed interval labels can be abbreviated. The abbreviations can be as short as the least number of leading letters that distinguish each reserved word. Additionally, there are specific abbreviations, which especially facilitate the most-used reserved words.

Also, in Desc, the layered interval references don't need inner left punctuation; the left angle bracket is still required, though. Even with this shortcut, the layers are still visually obvious.

```
Radius | <Focus Circle) Radius>
```

and

```
Area | <D : Math} Circle] Area : Radius>
```

So, Desc can be as long as pure Descript and as short as shorter than D—D with abbreviated keywords (and symbols). It's actually all three dialects—and all of the possible combinations of them.

In addition to all of the elements of D and Descript, Desc has one other element, because of the ability to abbreviate interval labels. In the managed interval definition, each management component can have a specific abbreviation defined, in parentheses or with the label `abbreviation`, following the component name. While this is a nice feature, it's also important for ongoing development; when a new label makes an existing label's leading-letter abbreviation indistinguishable, and that abbreviation is widely used, the specific abbreviation can be assigned to the existing label, and no further changes will be required.

```
<Picture Validation : Width (W) [int] | : Height (H) [int] |>
  [bool]
```

is equivalent to

```
interval: Picture Validation; label: Width; abbreviation: W;
  type: int; label: Height; abbreviation: H; type: int;
  purpose return type: bool
```

Reserved Words

In the following sections, the distinguishing letters are in italics. Also, the second column lists the specific abbreviations. Italics in an abbreviation mean that it can be as short as those letters. The same abbreviation can be for both a label and a keyphrase; this is possible because of the context (placement) in a statement.

Labels

Individual Identifiers

*ac*cess	
*al*ias	
*anc*estor	
*cl*ass	
*ch*aracteristics	
*comm*ent	cm
*compl*etion	
*compo*nent	*c*om
*g*roup	
injection	inj
*injection s*et	injs
*inl*et	
inner	inn
*inner c*lass	inc
*interf*ace	
*interv*al	*i*nter
*j*ump	
*la*bel	l
*li*brary	
*locate*r	lo
*locati*on	

*n*ew type
*o*bject
*pare*nt
part
*p*roduct set ps
*pu*rpose
*ret*urn type rt
*ri*gidity
*se*ries
*size n*ame sn
*size v*alue sv
*so*urce s
*st*ore
*su*blevel
*tr*anslation
*ty*pe
*u*sability
*v*alue

Controls

and *a*n
*eq*uivalent to
for *f*o
*for e*very fe
*sa*me object as
*w*ith

Keyphrases

Scopes

*c*ompletion end ce
*fo*llow up
*g*roup end ge
injections injs
*injections e*nd injse
*monitor n*ext mx
*monitor of*f mf
monitor on mn

*nest*ed	ne
*nest*ing	ni
*ne*xt	nx
*o*ff	
on	
*para*llel	pa
*ser*ies end	se

Controls

*ret*urn	r
set return	sr

Modifiers

*ad*dition	
*al*l opposite	ao
*asp*ect	
*ext*ension	
*fin*ish	
*fir*st	
*inc*omplete	
*int*erface only	ifo
*la*st	
or	
*o*thers	
*rem*oval	rm
*rep*lacement	rp
*res*tart	
*st*art	

Property Values

any
*asc*endants
*def*inition
*exe*cution
*li*brary
*per*manent
self
*te*mporary
*v*ariable

General Values

*fa*lse
*ini*tialization
*nu*ll
*ob*ject o
*pare*nt p
*tr*ue

An Example of Desc

The following is the same fairly simple interval set as the example of D. Various formats are used in the short code, to demonstrate the flexibility, but, with Desc, any company should set consistency standards.

Listing B-1. *The Guessing Game, in D*

```
> Application

-- permanent --
<application : parameter count [int] : parameters <> [string]>
  [] any
(Guessing Game) [Guessing Game] | <initialization>

-- temporary --

<Guessing Game>
  [Guessing Game]
Another Round | "y"
Response [string] variable
| <Round> | for every Response = Another Round

<Round>
Goal Number [smallint] | <[Number] Random Number : 1 : 100 : 0>
Player Number [] variable | 0
| <To Screen : "Guess my number.">
Low Number [] variable | 1
High Number [] variable | 100
| <Clue> | for every Player Number \= Goal Number
| <To Screen : "Right.  Do you want to play again?">
Response | <From Screen>

<Clue>
| <To Screen : "The number is from "
  Low Number " to " High Number ".  Your guess:">
Player Number | <From Screen>
| <To Screen : "Higher."> | for Player Number < Goal Number
+ Low Number | Player Number + | for Player Number \< Low Number
| <To Screen : "Lower."> | for Player Number > Goal Number
+ High Number | Player Number - | for Player Number \> High Number
```

Listing B-2. *The Guessing Game, in Desc*

```
ancestor Application

-- permanent
<application : parameter count [int] : parameters <> [string]>
  [] any
(Guessing Game) [Guessing Game] | <initialization>

char: temporary

<initialization>
  purpose: [Guessing Game]
Another Round | "y"
Response; type: string
| <Round> | for every Response = Another Round

int: Round
component: Goal Number [smallint]; jump: Random Number; Low: 1 : 100 DP: 0
c: Player Number; t:y | 0
j: To Screen : "Guess my number."
comp: Low Number; ty:; v: 1
com High Number; typ; val 100
j Clue | for every Player Number \= Goal Number
| j To Screen; String: "Right.  Do you want to play again?"
c Response; j From Screen

i Clue
j To Screen : "The number is from "
  Low Number " to " High Number ".  Your guess:"
Player Number | <From Screen>
| j: To Screen : "Higher."; f: Player Number < Goal Number
+ Low Number | Player Number +; f Player Number \< Low Number
| <To Screen : "Lower."> | (Player Number > Goal Number)
f; c High Number; s Player Number -; f Player Number \> High Number
```

Appendix C

This translation key can be used just for quick reference, or in conjunction with the data-oriented dictionary.

Procedure-Oriented to Data-Oriented Translation Key

The procedure-oriented vocabulary here is most closely related to Java.

Concepts

abstract class incomplete class
abstract method interface-only interval
address location
arithmetic operation combination
array series
block follow-up steps
calculation combination
call jump
cast type change
class interval set
class member permanent characteristic
concatenation combination
data declaration component definition
data item component
data name component reference
data type component type
dynamic member temporary characteristic
for loop incrementation set
function interval
function data item interval component
function orientation interval orientation
if conditional specific usage

import library implicit reference
instance member temporary characteristic
interface interface-only class
library package
main method application interval
message interval reference
method interval
method data item interval component
object member temporary characteristic
object orientation interval orientation
package product line
parameter management component
program product set
read resource part access
record part
reference locater
static member permanent characteristic
structure group
switch parallel specific usage
tokenized string incomplete string
type component type
while loop repetition
write product part completion

Keyphrases

and with
close finish
delete removal
else for others
final execution (component), definition (interval)
if for
open start
or and
private self
protected ascendants
public any
rewrite replacement
static permanent
this object
while for every
write addition

Data-Oriented Dictionary

In the descriptions, any bold words are also defined in this dictionary. For brevity, any syntax descriptions are most closely related to D.

access

When not modified by or modifying other words, a **resource part access**.

access interval

An **interval** that allows access to a `self variable` component. An interval name that matches a noninterval component name, without parameters; or an interval name that matches a noninterval collection name, with one parameter.

addition

A normal product part completion.

alias

An entity to implicitly translate a noun-based **interval** to another language's corresponding verb-based method name. A **statement** of the alias interval name enclosed in angle brackets, followed by the translation.

```
<Option Selection> selectOption
```

alias table

A table of **aliases**, outside of any **set**.

ancestry statement

A **set statement** that describes the **set's** complete ancestry, in reverse order, for both definition and documentation. A greater-than sign at the beginning of the **statement** and preceding each successive ancestor.

```
> Shape > View
```

aspect class

A fundamental class that each other class must ascend from, to serve clearer purpose. Denoted by `aspect` as the first statement, instead of an **ancestry statement** (because the **generic class** is the only ancestor of an aspect class), the standard aspect classes are "language"; "Utility", "Mediation", and "Application"; and "View", "Model", "Security", "Logging", and "Persistence".

aspect orientation

To specifically separate (auxiliary) things like security and logging logic from the (core) business logic. Employs **inlets** into an interval, and **injections** from a referencing interval, based on matching the name of the inlet and the injection, causing only these to require a naming standard. Serves as a reverse reference for the server interval, extending a longtime technique—passing a service object—to the interval level.

block interval

A `self` interval referenced by a single interval, designated, in the interval definition, to use the referencing interval's components directly. Logically, a block of the referencing interval, also reflects more-specific usability than `self`.

body statement

A **statement** that does not define the scope of other statements. Its scope is defined by a **header statement**.

chaining

Object creation solely for the purpose of an **interval reference** in the same step. Possible with the two-dimensional **nesting statement**.

```
^ [Circle] | <initialization : Radius>
| <^ Rendering>
```

characteristic

Any entity of an **interval set**, either **permanent** or **temporary**.

characteristics set

One of two sets of entities of an **interval set**, either **permanent** or **temporary**.

column

A physical segment of a **statement**, to serve generic functionality, it's variable width, separated by a vertical slash from the adjoining column. The **component column**, the **source column**, or the **control column**.

combination

Two or more **components** combined through one or more arithmetic or string operations, the same as Java-based assignments, with a few exceptions.

```
Group Mark Column | Group End Column +
```

and

```
Group Begin Column | + Group Mark Column
```

and

```
Column Counter | +
```

and

```
Result Message | "The next valid transaction "
  "number is " Transaction Main Number & Transaction Check Digits
```

comment

A non-code **statement**. Starts with an asterisk.

```
* In normal writing, an asterisk indicates a note following its subject.
```

completion

When not modified by or modifying other words, a **product part completion**.

component

Any smallest practical piece of a **product**, **resource**, or **interval set**. Also the generic term for a group or a series.

component column

The first **column** of a **statement**.

component definition

A **statement** with the **component** name followed by brackets, optionally enclosing a **component type**, in the first column, and an initial value optionally in the second column, followed by the **interaction properties**.

```
Response [string] | null
```

component group

A set of any **components** (or **series** or other groups).

component list

A **component reference** to multiple **components**, separated by commas.

component name preword

A component name prefix separated by a space, to make less frequent purposes more immediately obvious. One of three values identifying which type of definition the component is (directly) defined in—permanent, temporary, or management—with `p`, `t`, and `m`, respectively; and one of three other values identifying how the component is used—that it's a variable, dynamic string, or incomplete string—with `v`, `d`, and `i`, respectively. The two categories can be combined into one preword, with the definition identification first.

```
v Radius [float] (in, out) variable
```

and

```
p Pi [] any execution | 3.14159
```

and

```
tv Old Length [] self variable
```

and

```
<Area : m Radius [float]>
```

component segment

In Descript, the first **segment** of a **statement**.

component series

A set of multiple occurrences of the same **component** (or **group** or other series).

component type

The format and functionality of a **component**.

compound statement

A **statement** in which either or both of the first two **columns** have something and the third does also. Part of a compound statement controls the rest of it.

constant component

A **component** with a value, specifically to allow symbolic values to not be used in **step statements**, forcing purpose to be as clear as possible.

continuation

In a multiline **statement**, an indentation on each succeeding line.

continuation mark

In a multiline **statement** that contains other multiline statements (for example, a **group definition**), a semicolon at the end of a continued line of a contained statement.

control column

The third **column** of a **statement**.

control incrementation

An **access**, or a specialized **component** incrementation or **interval reference**. Just the incrementation component followed by an arithmetic operator and an incrementation amount, or a reference to an interval that cannot return a component.

control segment

In Descript, the third **segment** of a **statement**.

durability (interaction property)

Simply defined by which characteristics set the component definition or interval definition is in.

dynamic string

A string that has an implicit construction when referenced. Compared to an **incomplete string**, it doesn't require a **component list** when referenced.

dynamic string definition

A string **component definition** with a single set of angle brackets and a standard string construction.

```
Availability Message [] | <> "The room is available on " Target Date
  ", from " Start Time " to " End Time "."
```

extension interval

A feature that functionally promotes **interval components**, specifically to allow using a referenced **interval** to create **components** for the referencing interval—effectively, a referencing interval inheriting from a referenced interval. Denoted by the left angle bracket preceded by `extension` or an exclamation mark with no space.

```
!<Context Acquisition>
```

finish

The finish to processing on a **product** or **resource**, usually implied.

follow-up step

A step that is executed to accommodate the true condition of the step before it, forming a single-level functional block.

follow-up step definition

A **statement** that starts with a plus sign.

```
[Source Code || (Source Code = Not Written)
+ Source Code | Empty Layout Message |
+ ]
```

function interval

An **interval** whose sole purpose is to produce its **return component**, denoted by the left angle bracket preceded by `function` or a question mark without a space following it.

```
?<Picture Size>
```

generalized object

An object defined as one type and initialized as a specialization of that type. Denoted by an ancestor, enclosed in brackets, between the two other entities.

```
(User Shape) [Shape] [Circle] | <initialization : Radius>
```

generic (class)

The most basic class, from which all other classes are extended.

group

When not modified by or modifying other words, a **component group**.

group alignment member

A **group member** with no member component name.

group definition

A **statement** with the component group name and the left bracket; followed by the **component definition** of each **group member** and a statement with just the right bracket, all of which can be indented.

```
Keyed Report Line [
    Sequence Number [smallint] | 1
    [smallint]
    Department Number [smallint]
    ]
```

group member

A **component** that is part of a **group**.

group member reference

For a unique group member, just its name; for a nonunique member, the member name preceded by all of its immediate owner levels' name until one of those is unique. This reference must begin with `group` or an at sign, the levels must be separated by colons, and all of it must be enclosed in angle brackets.

```
<@ Original Report Line : Print Control : Another Component>
```

header statement

A **statement** that defines the scope of **body statements**.

idiom

Code that doesn't properly translate to another language—a rare occurrence—remains intact, to help to ensure complete compatibility. Designated with a backslash in the first column; effectively a comment that gets its comment character stripped.

implementation statement

A **set statement** that defines all of the **interface-only classes** that the **set** implements. A double greater-than sign at the beginning of the **statement**, followed by one or more interface-only class names, separated by commas.

```
>> Error, Message
```

implicit access

The standard (single statement) access **interval**, designated in the **component definition**. Designated with `in` enclosed in parentheses, in place of the usability property. Can be combined with **implicit update**.

```
Radius [float] (in, out) variable
```

implicit update

The standard (single statement) update **interval**, designated in the **component definition**. Designated with `out` enclosed in parentheses, in place of the usability property. Can be combined with **implicit access**.

```
Radius [float] (in, out) variable
```

incomplete class

A class that has any, but not all, **interface-only intervals**. Identified with just `incomplete` as its second statement.

incomplete string

A string that has **component** insertion points. Compared to a **dynamic string**, its definition is functionally completed when referenced.

incomplete string definition

A string **component definition** with a set of angle brackets with nothing between them for each **component** insertion point.

```
Availability Message [] | "The room is available on <>, from <> to <>."
```

incomplete string reference

A **component list** of the string name and the insertion string components, in order.

incrementation set

A **repetition** implemented in conjunction with a **control incrementation**. Can include a **specific usage**.

```
|| {Print Control \= Double Space} Report Original Line
```

injection

A direct or indirect addition to an **interval reference**, to add logic to the referenced interval. In the direct form, the reference without a right angle bracket, its statement followed by an injection header, which has the injection name enclosed in double angle brackets, followed by the statements to inject, followed by the interval reference's right angle bracket on a line by itself.

```
<Client Interval>
| <(Server) Server Interval
<<Security -- Login>>
| <(Third-Party) User Validation>
>
```

injection set

A set of **injections** that stand alone or in a group. Denoted by a file name with a caret prefix, referred to (functionally copied) anywhere in a class or other injection set, with the injection set name, including the caret, enclosed in double angle brackets.

```
<<^ Security>>
```

inlet

A point in an **interval** where an **injection** can add logic. In the referenced interval, an inlet name, surrounded by double angle brackets, in the second column.

```
<Server Interval>
* some core statements
| <<Security -- Login>>
* more core statements
```

interaction properties

Definitions of how any entity interacts with other entities. **Usability**, **rigidity**, and **durability**.

interface-only class

A class that has all **interface-only intervals**. Identified with just `interface only` as its first **statement**.

interface-only interval

An **interval** that has no implementation. Identified with just `interface only` in its body.

interval

A description of **components** through a period of time.

interval component

A **component** that is defined in an **interval definition**.

interval definition

A **header statement** that has the **interval** name enclosed in angle brackets, followed by the interval's **return component definition**, followed by the **interaction properties**.

```
<Picture Validation>
  [bool]
```

interval locater

The explicit term for a **locater** that contains the location of an **interval**; can also be called just a locater. Copied with the standard **interval reference** enclosed in parentheses.

```
(Action Monitor) | (<Action Performed>)
```

interval orientation

The organization of groups of steps into functional units, specifically to make the groups more independent of each other, to help to minimize code changes.

interval reference (fully explicit)

A visual representation of the layering, as opposed to a stringing. Within angle brackets, with **function intervals** and **extension intervals** having their respective keywords or punctuation; an object name enclosed in parentheses and any entity name; or a line name enclosed in braces, a class name enclosed in brackets, and any entity name. With an **interval locater**, the same as with an object locater, without a further interval name.

```
Radius | ?<(Focus Circle) Radius>
```

and

```
Area | ?<[{D : Math} Circle] Area : Radius>
```

and

```
| <(Action Monitor) : Action>
```

interval set

A description of **products** and, more directly, their **components** through periods of time (**intervals**).

jump

Triggered by an **interval reference**, a change of flow from within one **interval** to the beginning of another, through that one and back to the statement that follows the reference. Delineated by the interval name enclosed in angle brackets in the second column.

```
Pixel Count | ?<Picture Size>
```

keyphrase

A keyword, or a specific set of keywords separated by spaces, that isn't a **label**. All lowercase, to easily differentiate them from developer-defined words, which are title cased.

label

A word, or a specific set of words separated by spaces, that describes what an entity is, as opposed to how it is used. These can be reserved or developer defined.

line

When not modified by or modifying other words, or when used in "in a line" (as opposed to "on a line"), a **product line**.

line implicit reference

A **statement** at the beginning of a set to allow **set references** to imply their **line reference**. A line reference, followed by every applicable referenced set name, separated by commas.

```
{D : Math} Circle
```

list

When not modified by or modifying other words, a **component list**.

locater

A **component** containing the location of another component, for an object or **interval** only, and most simply thought of as the object or interval itself. Used exactly like any other component, except its value can't be set explicitly.

major entity

An entity that defines the scope of **minor entities**, defined by a collection of **statements**. A **characteristics set** or an **interval**.

managed interval

An **interval** that operates on or from one or more **management components**.

managed interval definition

An interval definition that includes a management component definition list.

```
<Picture Validation : Width [int] | : Height [int] |>
  [bool]
```

managed interval reference (recommended format)

The **managed interval reference short format**, with a **label** before each colon, and a semicolon before each label.

```
Goal Number [] |
  ?<Random Number; Low Number: 1; High Number: 100; Decimal Places: 0>
```

managed interval reference (short format)

The interval name, followed by the **management component** values or other **component** names, each preceded by a colon.

```
Goal Number [] | ?<Random Number : 1 : 100 : 0>
```

management component

A subject or a controller of a **managed interval**.

management component definition

A list with each component definition preceded by a colon and the optional value being a default value.

```
<Picture Validation : Width [int] | : Height [int] |>
```

minor entity

Any non-**major entity**, which doesn't define the scope of other entities. A **part**, a **component**, a **group**, or a **series**.

multicomponent return

A return of multiple **components**, which can be copied to either a group or a list.

nesting

One interval reference inside of another. In the two-dimensional form, for the nested portion, in place of the component is a caret, and for the interval reference portion, in place of the nested interval reference is another caret.

```
Component Value | ?<string segment : Report Layout
  : Column Counter : ?<string length : Original Value>>
```

and

```
^ | ?<string length : Original Value>
Component Value | ?<string segment : Report Layout
  : Column Counter : ^>
```

noninterval component

A **component** that is defined outside of an **interval**.

object definition

In its basic form, the object name enclosed in parentheses, followed by the class name enclosed in brackets, in the first column, and `initialization` as an interval reference, in the second column. In other forms, defines a **generalized object** or a **specialized type**.

```
(User Shape) [Circle] | <initialization : Radius>
```

package

A basically self-sufficient logical grouping of **product lines**.

parallel

When not modified by or modifying other words, a **parallel specific usage**.

parallel specific usage

A set of **specific usages** in which only the first true condition is accommodated, designated by each of the involved statements starting with a hyphen.

```
- | <Main Line Process> | for Special Attribute = Main Line Attribute
- | <Redefine Process> | for Special Attribute = Redefine Attribute
- | <Table Process> | for others
```

part

Any most practical subset of a **product** or a **resource**.

part implicit reference

A **statement** after any **line implicit reference** statements to allow an **interval** reference to stand out as a **completion** or **access**. Optionally, the completion interval name enclosed in angle brackets in the first column; the store name and the part name, separated by a colon, enclosed in brackets in the second column; and, optionally, the access interval name enclosed in angle brackets in the third column.

```
<Keyed Report Line Product> | [Keyed Report : Keyed Report Line]
  | <Keyed Report Line Resource>
```

permanent (characteristic)

Basically, an entity that can be shared directly with other **sets**.

permanent characteristics definition

A **statement** that is a double hyphen, followed by `permanent`, optionally followed by another double hyphen.

```
-- permanent --
```

product

An output of a system; the focus of data orientation.

product line

A logical collection of **sets**, to delineate related basic functionality.

product part completion

An occurrence of an output from a system. A left bracket and **product part** name in the first column; in its recommended form, a right bracket in its own statement; in the simpler form, the right bracket immediately following the part name.

```
[Report Keyed Line
* all of the components populated
]
```

and

```
* components populated in various places
[Report Keyed Line]
```

product set

An interval set.

reference

An entity usage, as opposed to an entity definition.

reflection

Examination of the **characteristics** of an object or a class. Two parts, a "class" **interval**, which returns the name of the class of the object, and a "reflection" object, which can be created with any string, can be combined.

```
User Class [string] | <(User Object) class>
Object Documentation [reflection] | User Class
```

can be combined into

```
Object Documentation [reflection] | <(User Object) class>
```

removal

A **product part completion** that is the removal of a part.

repetition

Similar to a **specific usage**, a process step based on a condition that occurs repeatedly, most often implemented in an **incrementation set**. Described with the sequence control `for every` followed by a condition, which can also be braces surrounding the condition.

```
|| {Print Control \= Double Space} Report Original Line
```

replacement

A **product part completion** that is the replacement of a part.

resource

An input for a system, which can also be seen as an intermediate output, either from the same system or another.

resource part access

An occurrence of an input to a system. Just the name of the **resource** (or resource **part**) in the third column.

```
|| Original Report
```

restart

A **finish** and then a **start**, appropriate for getting the **part** pointer set back to the beginning.

return component

A **component** that is returned from an **interval**. Its name is the same as the interval's by default.

return component definition

Part of an **interval definition**, on its own line, the return component's name (optionally) and type enclosed in brackets in the first column, and an initial value optionally in the second column.

```
[bool] | false
```

return multicomponent definition

A **return component definition** for a **multicomponent return**, it must include names, and its individual return components are separated by commas.

```
X [int] | 0, Y [int] | 0
```

rigidity (interaction property)

Determines when the entity being defined can be updated. Defined with any one of `definition`, `execution`, or `variable`, following the **usability** specification, if it exists.

segment

In Descript, a logical segment of a **statement**, to serve generic functionality; the **component segment**, the **source segment**, or the **control segment**.

sequence modification

A **specific usage** or a **repetition—**or both, which function as a selective grouping.

series

When not modified by or modifying other words, a **component series**.

series definition

A **statement** that follows the **group definition** format, but with the series name optional, and an optional size name and a size value, separated by a vertical slash, enclosed in angle brackets, preceding the left (square) bracket.

```
Report Page Spacing <Lines Per Page | 60> [Character Count [smallint] | 2]
```

and

```
<Estimated Maximum Employees | 100> [Employee Number [int]]
```

series member

A **component** that is part of a **series**.

series member reference

`series` or a number sign, followed by all of its series level indices' name, separated by commas, then a colon; and then the member name. Or no specified series levels, indicating all of the occurrences of the member, which is useful when the series is unnamed.

```
<# Week Number, Employee Counter : Hours Worked>
```

and

```
<# : Hours Worked>
```

set

When not modified by or modifying other words, an **interval set**.

set property

A basic set definition, documentation, or a shortcut. A **line implicit reference**, a **part implicit reference**, or an **alias**.

side effect

A self-contained standard comment that doesn't affect the statement at all, to make the statement stand out visually. Denoted with a slash as the first character, which is the comment.

```
/ active body statement
```

source column

The second **column** of a **statement**.

source segment

In Descript, the second **segment** of a **statement**.

specializable type

A feature that creates an object whose exact **type** is not determined until usage. Denoted with the ancestor name enclosed in parentheses inside of the brackets. This can be of any specificity, all the way down to generic.

```
(User Shape) [(Shape)] | <initialization : Radius>
```

specific usage

A process step based on a condition, with condition controls (`and`, `with`, and `others`) that describe all of the conditions that that step applies to. In the third column, `for` followed by a condition, or a condition enclosed in parentheses.

```
| <Header Skip> | (Print Control = Page Break)
| <Layout Check : Symbol Count> | for others
```

start

The start to processing on a **product** or **resource**, usually implied.

statement

A description of any set property or characteristic of an interval set; a set statement, a header statement, or a body statement.

status monitoring

On-and-off (functional block) status management, it has two forms: single **statement** monitoring and multiple statement monitoring. In the multiple form, a left brace for an on or a right brace for an off, followed by a `for` clause that can be applied to an implied "status" component and an implied status value component name, with `status =` optional; in the single form, both braces, then the `for` clause.

```
{} Unexpected End of Resource || for status = End of Resource
```

is a single.

```
{ Unexpected End of Resource || (End of Resource)
```

is on, and

```
} || (End of Resource)
```

is off.

status table

To serve **status** monitoring, a table for developer-defined status name, type (information, warning, error, or severe), and description, manually maintained, in a file named "status table".

step

A logical step of progression through a process.

step definition

Nearly every **body statement**, a description of a logical step of progression through a process; a baseline, nonorganizational description.

temporary (characteristic)

Basically, a characteristic that can be shared directly between **intervals**.

temporary characteristics definition

A **statement** that is a double hyphen, followed by `temporary`, optionally followed by another double hyphen.

```
-- temporary --
```

transitional code

A double comment that signifies that a design decision has not been made, to facilitate concise working notes in the code. Separating two statements, `or`, which can also be a question mark.

```
Line Position | +
? Line Position | + Field Length
```

type

When not modified by or modifying other words, a **component type**.

type change

The translation of a **component** from one **type** to another. Denoted by a component reference in the first column and the new type, enclosed in brackets, preceding the source in the second column.

```
(Focus Shape) | [Circle] (Previous Shape)
```

update interval

An **interval** that allows update of a `self variable` component. An interval name that matches a noninterval component name, with parameters; or an interval name that matches a noninterval collection name, with multiple parameters.

usability (interaction property)

Determines what types of entities can use the entity being defined. Defined with any one of `any`, `line`, `ascendants`, or `self`, following the component type or the return component definition.

Index

D

N

O

P

R

S

T