

# 1 Language Definition File XML Specification

The language definition file (lang\_def.xml) describes two parts:

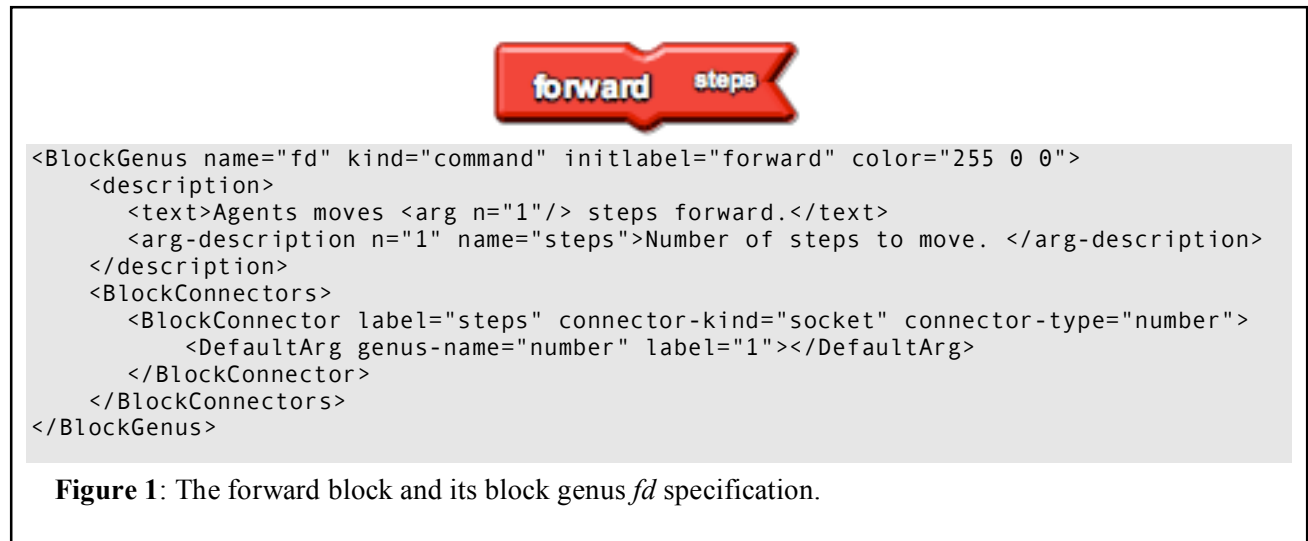
- block language
- Workspace, which is the programming environment for the specified block language.

(Note: The lang\_def.dtd files specifies rules, elements, attributes within lang\_def.xml.)

## 2 Specifying the Block Language

### 2.1 Block Genus

A block genus describes the properties that define a common set of blocks. For example, *fd* is a block genus that describes all forward blocks in Starlogo. For each block in your block language, you must specify a BlockGenus.



Block Genus Attributes	Description
name	Unique name for genus
kind	Block genus class. Currently, codeblocks uses three kinds: command, data, variable
initlabel	The initial label for a block in this genus.
color	The RGB color for a block in this genus.
editable-label	If “yes”, block label is user-editable. Unless specified in block genus, editable-label = “no” by default (i.e. genus <i>fd</i> is not editable.)
label-unique	If “yes”, every block instance of this genus must have a unique block label. Set to “no” by default.
is-label-value	If “yes,” its block label determines the block value. This attribute is applicable to data blocks (i.e. number, sting, true, false). Set to “no” by default.
label-prefix	A string to always precede its block label

label-suffix	A string to always succeed its block label
sockets-expandable	If “yes,” sockets expand whenever a block is connected to them. Set to “no” by default.
is-starter	If “yes,” this block genus begins block stacks of code. Set to “no” by default.
is-terminator	If “yes,” this block genus ends block stacks of end. Set to “no” by default.

### 2.1.1 Description

The description element specifies the formatting and text of a user-friendly description of the blocks within this genus.

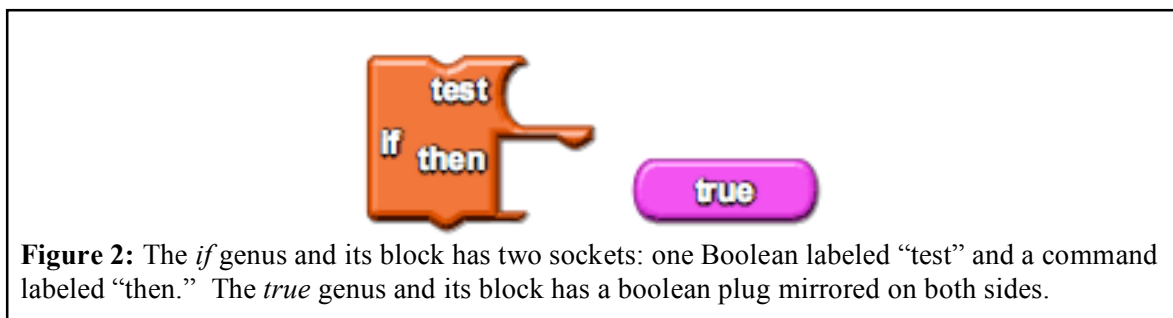
(Note: this is not completely specified yet.)

### 2.1.2 BlockConnectors

The BlockConnectors specifies the properties of all the connecting points of a block genus. There are four kinds of connectors, differing in their meaning and location:

- Plug: Returns a value. Resides on the left side of a block.
- Socket: Reads in a value or branches the flow of execution. Resides on the right side of a block.
- Before: Connects to previous command. Resides on the top side of a block.
- After: Connects to the next command. Resides on the bottom side of a block.

Each connector has a type that determines its shape and the data (number, string, boolean, polymorphic) or command type that it reads/returns/connects to. *Before* and *After* connectors are always of type command. A *Socket* may be of any type. Similarly, a *Plug* returns any type except command.



Sockets and Plugs are specified in the xml file. Before and After connectors are dynamically generated, depending if the genus kind is command or if the genus is a starter or terminator.

BlockConnector Attributes	Description
label	Label that resides by the connector on the block. For example, <i>steps</i> is a label of <i>forward</i> ’s socket.
label-editable	If “yes,” connector label is user-editable. Set to “no” by default.
connector-kind	Determines if connector is a “plug” or a “socket.”


	Before and After connectors are dynamically generated depending on genus kind, starter, or terminator properties.
connector-type	Data (number, string, boolean, polymorphic) or command type.
position type	Determines the position of this connector. single: either left or right depending on kind mirror: plug is mirrored on the right side of block bottom: sockets are placed in the bottom of the block

Some blocks have default arguments or blocks that are connected to its connectors when the block is dragged onto the block canvas. For example, the forward block has a default argument, the number block with its value set to 1. Default arguments can be convenient, especially if the default argument is a common choice to connect to a block.

Default Argument Attributes	Description
genus-name	the block genus name of the default argument
label	the label (if genus is editable) of the argument

### 2.1.3 BlockStubs (Advanced)

Some block genres may have dynamically generated blocks when its block instance is dragged onto the block canvas. These dynamically generated blocks are called Stubs, and they reference the parent block they were created from. Examples of stubs are getters and setters for variables. The block genus *agent-var-boolean* specified below has two stubs: a getter and a setter.



```

<BlockGenus name="agent-var-boolean" kind="variable" initlabel="agent boolean"
editable-label="yes" color="65 105 225">
...
  <BlockConnectors>
    <BlockConnector connector-kind="socket" connector-type="boolean">
    </BlockConnector>
  </BlockConnectors>
  <Stubs>
    <Stub stub-genus="setter"></Stub>
    <Stub stub-genus="getter"></Stub>
  </Stubs>
  <LangSpecProperties>
    <LangSpecProperty key="scope" value="agent"></LangSpecProperty>
  </LangSpecProperties>

```

**Figure 3:** The genus *agent-var-boolean* has two stubs, a getter and a setter.

Currently there are five stubs available, each with its own genus:

- getter: a data block returns the value of its parent block.

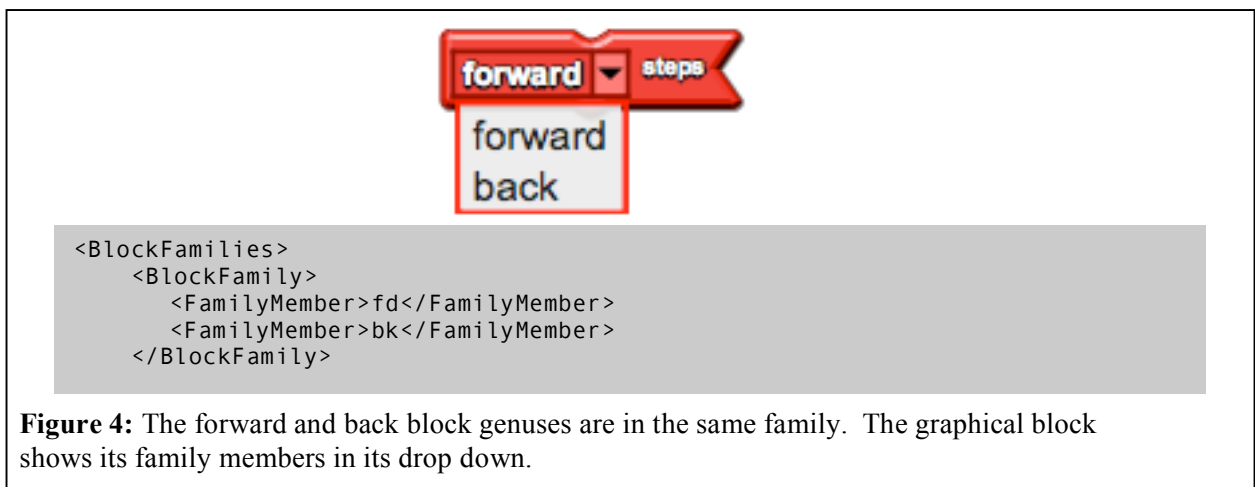
- setter: a command block that sets the value of its parent block
- inc: a command block that increments the value of its parent block by a fixed value.
- caller: a command block that executes the commands enclosed by its parent. This stub is used by procedure blocks to create their call blocks.
- agent: a data block that returns the value its parent block for a particular agent

#### 2.1.4 LangSpecProperties

The Codeblocks library is language independent and not all properties specified in the language definition file can cover all the properties needed by a block language. Each LangSpecProperty has a key-value pairing that when loaded is saved within the properties HashMap in the BlockGenus class.

## 2.2 BlockFamilies

Block families provide additional convenience for the user by grouping block genres that have similar functionality into families. When blocks are rendered in the workspace, blocks with families have an additional drop down box included in its graphical block. Family members are included in this drop down box.



## 3 Specifying the Workspace

### 3.1 BlockDrawerBars and BlockDrawer

BlockDrawerBars manage BlockDrawers, which are block containers. BlockDrawerBars contain a series of buttons, each button toggling the visibility of one block drawer.

There are four types of block drawers, each differing in behavior and functionality.

- Default: contains one instance of each block contained. Blocks may be dragged and dropped within the drawer and to and from the block canvas or other block drawers.
- Factory: contains blocks that can produce an infinite amount of block instances. When a block is picked from a factory drawer, a new instance is created. When a block is dragged back to a factory drawer, the block is deleted. The blocks contained within this drawer never move, but only produce new blocks.
- Page: contains blocks for a particular page

- Custom: Similar to the default block drawer, except that this drawer may be saved with the project for future use.

<b>BlockDrawer Attributes/Elements</b>	<b>Description</b>
name	the name of this drawer. The name is displayed in the button corresponding to this drawer
type	drawer type: default, factory, page, custom. Set to “default” by default.
is-open	If “yes,” drawer is displayed at workspace startup. Set to “no” by default.
button-color	The color of the corresponding button
BlockGenusMember	the genus name a block within this drawer
Separator	Formatting property. If specified between BlockGenusMember elements, a line is drawn between the genus members in the graphical drawer.
NextLine	Formatting property. If specified between BlockGenusMember elements, the genus member after the NextLine element is drawn in the succeeding row in the drawer.

### 3.2 Pages

Pages divide up and organize the block workspace. Pages may have associated block drawers that contain blocks that are “special” to that page.

<b>Page Attributes</b>	<b>Description</b>
page-name	name of this page. Name is drawn on the page.
drawer-name	drawer associated with this page