SketchML::JSON

A JSON Implementation of SketchML

Part I: Introduction

Motivation

In order to accommodate the increasing number of lab projects, input devices, and libraries being used, we wish to unify all of the our sketch storage formats under a single, consolidated standard. This standard will be written in JSON for the primary reason that it will be easily supported by NoSQL databases like MongoDB (XML is always saved as a blob, even in MongoDB unfortunately). However, because there is an existing standard library, SketchML, that has been carefully constructed and reviewed, we would like to use it as the basis of the JSON implementation. For this particular standard, very few changes are necessary between the XML and JSON implementations, the most prominent being that JSON does not differentiate between sub-elements and attributes.

Overview of SketchML

The original SketchML standard is written for XML. The basic construct is a <sketch> which contains a master list of unconstrained <point> objects. Points are then referenced via their id in <shape> objects. Shapes can be Strokes, Lines, or more complicated types, but their main commonality is a set of <arg> elements that contain the list of references to points. An additional feature of SketchML that is particularly useful is the <edit> element. This element allows types of actions to be defined and contains a set of <arg> objects that reference the affected shapes or points. The <edit> list can be used to replay sketches or build snapshots at a given time; it also removes the need to ever delete any objects since they can be “removed” from the sketch through an <edit> without actually being deleted. One more excellent feature of the <edit> list is that since it only contains references to objects, the <sketch> itself is more or less standalone. So all the strokes and points can be read directly from the SketchML file to display the sketch whether or not the <edit> elements are parsed. Parsing the <edit> elements simply provides an additional level of control with replayability and snapshots.

Overview of SketchML::JSON

SketchML::JSON is essentially SketchML for JSON. As mentioned before, one of the main differences is the removal of attributes. Attributes in SketchML::JSON become sub-elements. SketchML also supports custom attributes, so a number of “attributes” (sub-elements) have been officially added to SketchML::JSON to better support newer input devices. These include “hover” and “tilt” on the <point> elements.

One other important change has been made, which is the move from unconstrained points to unconstrained strokes. Because the sampling rate is so high on modern touchscreens, the number of points collected in a single stroke can be on the order of thousands. In traditional SketchML, these are stored in a master list, but the stroke itself *repeats* every id of the points. This creates quite a bit of size and processing overhead as the number of points scales up. In SketchML::JSON, points are always stored as sub-elements of strokes (since a point cannot be created without belonging to a stroke by definition). The main question that arises here is “What about multiple interpretations of groups of points?”. Because SketchML uses free points and all other objects reference points, any number of any set of points can be associated with any shape. That level of freedom is very good, but the individual point referencing system comes at enormous size and processing costs. SketchML::JSON offers a similar approach that provides all of the power with much less overhead. Rather than points being unconstrained, they are subelements of strokes. Strokes become the unconstrained primitive which provide the ability for multiple interpretations. However, it is important to be able to collect groups of points that do not necessarily comprise a full stroke in order to allow any number of any set of points to be recognized as being part of any shape. This functionality is provided through the “substroke” primitive which references start and stop indices from a stroke object. The “substroke” is really the only added primitive in the SketchML::JSON implementation. It was not added as a shape type because it has the special characteristics of indexing into a stroke object, and it is not a recognized shape, as shape objects are in SketchML::JSON. This is how sketch objects are represented in the SRL Sketch Shape Library, so it is a cross-compatible and more scalable way of managing multi-interpretable ranges of points.

One change in philosophy has been made in reference to a shape. in SketchML::JSON a shape by definition is recognized. An unrecognized shape can not exist. A shape is defined by its interpretation. With this change we have expanded the properties related to the recognition of a shape. These properties are: interpretationId, recognizerId, interpretation (formally type), complexity, isEndState, confidence, and isForced.

We have added two more properties that sketches can have: devices and pens. These are in addition to the existing “sketcher” property. Each stroke can have a device, pen, and sketcher. The reasoning for this that the variety of devices have increased with phones, tablets, laptops, watches and even more. There also exist large devices that can support multiple pens and multiple authors at the same time. Computers also support touch events, pens, and mouse inputs at the same time in the same sketch.

A Basic Example

An example of a simple, complete sketch in SketchML is shown below.

**<sketch id="UUID" units="pixel">**

**<sketcher>**

**<id>UUIDa</id>  
 <dpi x="72" y="72"/>  
 <nickname>john</nickname>  
</sketcher>  
<study>SOUSA Post-Reviw</study>  
<domain>Chemistry</domain>  
<point x="10" y="100" pressure="1" time="39612905340" id="UUID0"/>  
<point x="10" y="102" pressure="1" time="39612905343" id="UUID1"/>  
<shape id="UUID2" time="39612904343" author="UUIDa" type="Stroke" height="3" width="1">  
 <arg type="Point">UUID0</arg>  
 <arg type="Point">UUID1</arg>  
</shape>  
<shape id="UUID3" time="39612904350" author="UUIDa" type="Line" height="3" width="1">  
 <arg type="Stroke">UUID2</arg>  
</shape>  
<edit id="UUIDe1" time="39612904343" type="CreateStroke">  
 <trigger type="InputUp">UUIDt1</trigger>  
 <arg type="CreateStroke">UUID2</arg>  
</edit>  
<edit id="UUIDe2" time="39612904350" type="RecognizeLine">  
 <trigger type="Recognized">UUIDt2</trigger>  
 <arg type="RecognizeLine">UUID2</arg>  
</edit>**

**</sketch>**

This is a sample translation that does NOT represent the final Standard

**{  
 "sketch": {  
 "id": "UUID",  
 "units": "pixel",  
 "sketcher": {  
 "id": "UUIDa",  
 "dpi": {  
 "x": "72",  
 "y": "72"  
 },  
 "nickname": "john"  
 },  
 "study": "SOUSA Post-Reviw",  
 "domain": "Chemistry",**

**"strokes" : {**

**"UUID2": {**

**"id": "UUID2",  
 "time": "39612904343",  
 "author": "UUIDa",**

**"height": "3",  
 "width": "1",  
 "points": [  
 {  
 "x": "10",  
 "y": "100",  
 "pressure": "1",  
 "time": "39612905340",  
 "id": "UUID0"  
 },  
 {  
 "x": "10",  
 "y": "102",  
 "pressure": "1",  
 "time": "39612905343",  
 "id": "UUID1"  
 }  
 ]**

**}**

**},  
 "shapes": {  
 "UUID3": {  
 "id": "UUID3",  
 "time": "39612904350",  
 "author": "UUIDa",  
 "type": "Line",  
 "height": "3",  
 "width": "1",  
 "arg": {  
 "type": "Stroke",  
 "id": "UUID2"  
 }  
 }  
 },  
 "edit": [  
 {  
 "id": "UUIDe1",  
 "time": "39612904343",  
 "type": "CreateStroke",  
 "trigger": {  
 "type": "InputUp",  
 "id": "UUIDt1"  
 },  
 "arg": [  
 {  
 "type": "CreateStroke",  
 "id": "UUID2"  
 }  
 ]  
 },  
 {  
 "id": "UUIDe2",  
 "time": "39612904350",  
 "type": "RecognizeLine",  
 "trigger": {  
 "type": "Recognized",  
 "id": "UUIDt2"  
 },  
 "arg": {  
 "type": "RecognizeLine",  
 "id": "UUID2"  
 }  
 }  
 ]  
 }  
}**

While this example may give the impression that SketchML::JSON is less space efficient than SketchML, the JSON representation will scale very well as the number of points increases to thousands. Note also that every single element has been place on a separate line; it can be compacted as much as the XML version and is actually more storage efficient. This is also more human readable and the benefits of NoSQL databases become immediately available.

Part II: Format Specification

Guidelines

In SketchML::JSON, the root object is a “sketch”. The “sketch” contains a master list of strokes called “strokes”, which themselves contain lists of the points called “points”. In JSON, it’s very easy to use lists of objects since the values just need to be surrounded with square brackets [ and ]. The “substrokes” and “shapes” primitives reference “strokes” by id. Substrokes access regions of strokes and “shapes” access entire substrokes or strokes. A detailed view of each element and its supported values follows.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Required** | **Type** | **Description** |
| **Top level object** (has no name) | | | |
| sketches | one or more | list | If multiple sketches exist this will be the top level object. If only a single sketch exist then these values may not exists |
| updates | zero or more | list | editing commands (erase, undo, etc.) |
| **sketch** | | | |
| id | required | UUID | UUID for sketch |
| unit | zero or one | string | defaults to “pixel”. a string of either “pixel” or “himetric” to indicate the unites for all the point. “himetric” is the standard format in c# |
| sketcher | zero or more | list | the person/people who drew the sketch |
| device | zero or more | list | the device/devices from which the sketch was created |
| pen | zero or more | list | the name of the stylus (could also be finger) from which the sketch was created |
| study | zero or one | string | the study for which the sketch was created |
| domain | zero or more | list | list of strings of the domain that the sketch is a part of |
| points | zero or more | list | list of points that can be referenced by strokes or substrokes, key is id |
| strokes | one or more | list | list of the strokes in the sketch, key is id |
| substrokes | zero or more | list | list of substrokes that reference strokes by index, key is id |
| shapes | zero or more | list | higher level shapes, key is id |
| speech | zero or more | map | speech actions (phrase, word, silence, etc.), key is id |
| mediaInfo | zero or more | list | reference to an external media file |
| attributes | zero or one | object | object mapping a set of attributes to their values (see Appendix E)  you are allowed to use custom attributes  all attributes should be saved in an decoding and recoding |
| **sketcher** | | | |
| id | required | UUID | the ID of the sketcher |
| nickname | zero or one | string | an abbreviation for the ID of the sketcher |
| **device** | | | |
| id | required | UUID | the ID of the device |
| dpi | zero or one | object | the DPI (dots per inch) of the device |
| os | zero or one | string | the name of the operating system |
| penEnabled | zero or one | boolean | whether or not a stylus is enabled for the device |
| touchEnabled | zero or one | boolean | whether touch input is enabled in this device |
| mouseEnabled | zero or one | boolean | whether mouse input is enabled for this device |
| physicalKeyboard | zero or one | boolean | whether a keyboard is connected to the device |
| **pen** | | | |
| id | required | UUID | the ID of the pen |
| productName | zero or one | string | product name of pen (Cintiq, Surface Pro 3, etc.) |
| brandName | zero or one | string | brand of the pen (Wacom, Microsoft, etc.) |
| description | zero or one | string | a written description of the pen |
| isPenFinger | zero or one | boolean | whether the pen is the user’s finger |
| isPenMouse | zero or one | boolean | whether the pen is the user’s mouse |
| penDigit | zero or one | number | if you have multiple pens and if you have finger digits, this identifies which it is  (see appendix B) |
| **strokes** | | | |
| id | required | UUID | id of the stroke |
| points\* | required | list | list of points in the stroke.  \*This will either be a list of point objects OR a list of points in the point cloud. It can NOT be both. |
| time | required | timestamp | time of the last point in the stroke |
| author | zero or one | UUID | id of the sketcher who created the stroke |
| pen | zero or one | UUID | id of the pen that created the stroke |
| device | zero or one | UUID | id of the device on which the stroke was drawn |
| name | zero or one | string | name or alias of stroke |
| attributes | zero or one | object | object mapping a set of allowed attributes to their values (see Appendix E) |
| boundingBox | zero or one | object | object containing the upper left and lower right points of the bounding box |
| isUserCreated | zero or one | boolean | whether the user created the stroke |
| **substrokes** | | | |
| id | required | UUID | id of substroke |
| stroke | required\* | UUID | UUID of the stroke being indexed into \*either this is required or the points key must be used |
| startIndex | required\* | integer | index of first point in stroke being referenced  \*either this is required or the points key must be used |
| stopIndex | required\* | integer | index of last point in stroke being referenced  \*either this is required or the points key must be used |
| points | one or more\* | list | A list of UUID that links to points in the point cloud.  \*either this is required or the stroke, startIndex, and stopIndex keys must be used |
| name | zero or one | string | name or alias of substroke |
| attributes | zero or one | object | object mapping a set of allowed attributes to their values (see Appendix E) |
| **shapes** | | | |
| id | required | UUID | id of the shape |
| time | required | timestamp | time of the creation of the shape (typically by recognition) |
| subElements | required | list | a list of strokes, substrokes, or shapes |
| interpretation | required | string | type of shape (circle, square, truss, arrow, etc.) |
| confidence | required | double | how confident we are of an interpretation. [when converting from formats that do not store this please hardcode this value as 1] |
| alias | zero or more | object | the aliases of the shape (names for components that are used in the language) |
| interpretationId | zero or one | UUID | the id of the specific interpretation. This is to differentiate the same interpretation that has been recognized in a different way |
| recognizerId | zero or one | UUID | the id that uniquely which recognizer created this interpretation |
| complexity | zero or one | double | how complex |
| isEndState | zero or one | boolean | whether or not more interpretations can be applied |
| isForced | zero or one | boolean | whether an existing interpretation is overridden |
| description | zero or one | string | description of the shape |
| domain | zero or one | string | domain of the recognized shape |
| attributes | zero or one | object | object mapping a set of attributes to their values (see Appendix E)  you are allowed to use custom attributes  all attributes should be saved in an decoding and recoding |
| boundingBox | zero or one | object | bound box of the shape |
| isUserCreated | zero or one | boolean | true if a user created the shape |
| **Update** | | | |
| id | required | UUID | id of the edit |
| time | required | timestamp | time of the edit |
| trigger | required | object | trigger that caused the edit |
| commands | one or more | list | The list of commands that all happened at the same time |
| **Command** | | | |
| id | required | UUID | id of the edit |
| type | required | string | on of the defined types of edits (see Appendix C) |
| arguments | one or more | object | arguments as defined by the edit type (see Appendix C) |
| **speech** | | | |
| id | required | UUID | id of the speech |
| author | zero or one | UUID | id of the author of the speech |
| startTime | required | timestamp | start time |
| endTime | required | timestamp | stop time |
| type | required | string | type of speech (word, phrase, silence, etc.) |
| source | zero or one | UUID | id of the source for the speech (reference to a mediaInfo) |
| score | zero or one | double | acoustic score of the word |
| word | zero or one | string | the word itself |
| wordForm | zero or one | integer | form of the word |
| **mediaInfo** | | | |
| id | required | UUID | id of the mediaInfo element |
| filename | required | string | filename (location) of the media |
| startTime | required | timestamp | start time of the mediaInfo |
| type | required | string | type of mediaInfo (audio, video, etc.) |
| **dpi** | | | |
| x | required |  |  |
| y | required |  |  |
| **points** | | | |
| id | required | UUID | id of the point |
| x | required | list | list of doubles indicating x locations; keeps a history of all x’s (newest last) |
| y | required | list | list of doubles indicating y locations;  keeps a history of all y’s (newest last) |
| time | required | long | the time the point was created |
| name | zero or one | string | name or alias of the point |
| pressure | zero or one | double | the pressure of the point (0 to 1) |
| tiltx | zero or one | double | amount of tilt on the input in the x direction |
| tilty | zero or one | double | amount of tilt on the input in the y direction |
| isHover | zero or one | boolean | true is the point was created hovering above the screen. |
| **boundingBox** | | | |
| id | required | UUID | id of the bounding box |
| topLeftCorner | required | object | the first point defining the bounding box. This is the top left corner by a user facing the screen |
| bottomRightCorner | required | object | the second point defining the bounding box.  This is the bottom right corner by a user facing the screen |
| name | zero or one | string | the name or alias of the bounding box |
| **alias** | | | |
| type | required | string | example: “Point”, “Stroke” |
| name | required | string | name of the alias (e.g. “head”, “shaft”) |
| **trigger** | | | |
| type | required | string | type of trigger (see Appendix D) |
| id | required | UUID | UUID of the trigger |

Part III: Official Format Example

First note is that this implements every key possible at least once and looking at the standard you can develop a subset of keys.

The situation:

A user draws 3 strokes that become interpreted as lines and into a pyramid.

The user then draws a stroke then deletes that stroke

{

“sketches” : [

{

“id”: “sketchId1”,

“unit”: “pixel”,

“sketcher”: [

{

“id”: “sketcher1”

},

{

“id”: “sketcher2”

}

],

“device”: [

{

“id”: “1ac96f”,

“dpi”: {

“x”: 50,

“y”: 60

},

“os”: “Apple Newton OS 3000”

},

{

“id”: “device2”,

“penEnabled”: true,

“touchEnabled”: true,

“MouseEnabled”: true,

“physicalKeyboard”: false

}

],

“pen”: [

{

“id”: “UUUIDs”,

“productName”: “fancypen5000”,

“brandName”: “pens-are-good”,

“description”: “this is a actually a pencil”,

},

{

“id”: “IDUU”,

“isPendFinger”:true,

“penDiget” : 2

}

],

“study” : “science”,

“domain”: [

“music-theory”

],

Appendix A: Fields in the SRL Sketch Shape Library

**Supporting Objects:**

SrlAuthor:

id: UUID

name : string

SrlDevice:

id: UUID

dpiX: double

dpiY: double

penEnabled: boolean

touchEnabled: boolean

mouseEnabled: boolean

physicalKeyboard: boolean

SrlPen:

id: UUID

penId: string

brand: string

description: string

isPenFinger: boolean

isPenMouse: boolean

penDigit: number

**Virtual Objects:**

SrlPoint:

pressure: double

tiltX: double

tiltY: double

xList: List<double>

yList: List<double>

isHover: boolean

*[inherited from AbstractSrlComponent]*

id: UUID

time: long

name: string

SrlBoundingBox:

topLeftCorner: SrlPoint

bottomRightCorner: SrlPoint

*[inherited from AbstractSrlComponent]*

id: UUID

time: long

name: string

SrlConvexHull:

points: List<SrlPoint>

*[inherited from AbstractSrlComponent]*

id: UUID

time: long

name: string

**Object:**

SrlShape:

*[Interpretation fields]*

interpretationId: UUID

recognizerId: UUID

interpretation: string

complexity: double

isEndState: boolean

confidence: double

isForced: boolean

*[Other fields]*

description: string

subShapes: List<SrlObject>

*[inherited from SrlObject]*

domain: string

attributes: Map<String, Object>

boundingBox: SrlBoundingBox

isUserCreated: boolean

convexHull: SrlConvexHull

*[inherited from AbstractSrlComponent]*

id: UUID

time: long

name: string

SrlText:

*[everything from SrlShape]*

*[Other fields]*

textValue: String

isHandWritten: boolean

SrlStroke:

*[Other fields]*

points: List<SrlPoint>

author: SrlAuthor

pen: SrlPen

device: SrlDevice

*[inherited from SrlObject]*

domain: string

attributes: Map<String, Object>

boundingBox: SrlBoundingBox

isUserCreated: boolean

convexHull: SrlConvexHull

*[inherited from AbstractSrlComponent]*

id: UUID

time: long

name: string

SrlSubStroke:

*[everything from Stroke]*

*[Other fields]*

startIndex: int

endIndex: int

parentStroke: SrlStroke

Appendix B: Digit Description

*/\*\**

*\* the digit of the pen that created the stroke.*

*\* For instance if a device supports multiple pens this will differentiate them even if the author is the same.*

*\* A more common example is if the user is using touch input and this will decide which finger it is.*

*\* Typically the digit is the first digit that is touched and the second one is the second digit on the screen.*

*\* also typically the first digit is the index finger. The second digit is the middle finger.*

*\* Note: this is not perfect but instead is just a guess about how it should work.*

*\* (we have numbered them below for each hand separately as if they belong to two different people).*

*\* <pre>*

*\* \_\_ \_\_ \_\_ \_\_*

*\* / V \ \_ \_ \_ \_ / V \*

*\* | | |/ V \ / V \| | |*

*\* | | | | || | | | |*

*\* | 4 |3 |2 |1 || 1| 2| 3| 4 |*

*\* | | | | || | | | |*

*\* | | | | || | | | |*

*\* | | | | || | | | |*

*\* \_\_\_\_\_ | | || | | \_\_\_\_\_*

*\* | 5 ''-/ || \-'' 5 |*

*\* \\_ || \_/*

*\* ''-\_ \ || / \_-''*

*\* \ ) /\ ( /*

*\* \ / \ /*

*\* | | | |*

*\* </pre>*

*\* (<a href="https://gist.github.com/sublee/600592/">code to create ascii hand</a>)*

*\*/*

Appendix C: Command Element Details

Commands are very strict into official supported types. If a command type is not officially supported it could lead to data not being saved in the same format.\*

The reason for this is that a command on a system where it is not supported could lead to vastly different sketches than on a system where the command is supported.

That being said this list should be added to (after discussion) and all known creators of parsers should be told about the changes and what new commands they need to support.

**List of supported command types:**

|  |  |
| --- | --- |
| **Type** | **Description** |
| ADD\_STROKE | Adds a new stroke to the sketch |
| ADD\_SHAPE | Adds a new shape to the sketch |
| PACKAGE\_SHAPE | moves an element from a shape or a sketch into a different shape or sketch |
| REMOVE\_OBJECT | removes an object from the sketch |
| ASSIGN\_ATTRIBUTE | adds an attribute to an element |
| REMOVE\_ATTRIBUTE | removes an attribute of an element |
| MARKER\* | A marker actually does not cause any changes to the sketch but instead contains data about events.  An example is saving the sketch |
| CLEAR | empties the sketch of all shapes and stroke |
| CREATE\_SKETCH | defines a sketch being created. |
| SWITCH\_SKETCH | switches what sketch edits apply to |
| UNDO | tells the stack pointer to look back one edit |
| REDO | tells the stack pointer to look forward one edit |

\*all marker data must persist between every encode and decode

Key table:

|  |  |  |
| --- | --- | --- |
| **Command Name In Bold** | | |
| argument name | argument type | argument description |

**List of command types and their arguments**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Required** | **Type** | **Description** |
| **ADD\_STROKE** | | | |
| id | required | UUID | The id that links to the stroke being added |
| **ADD\_SHAPE** | | | |
| id | required | UUID | The id that links to the shape being added. |
| shapeObject | zero or one | object | [optional] this is the original shape. This is used in case the interpretation of the shape gets forced into a different shape. In this case the recognizer id is equal to the id of the sketcher. |
| **PACKAGE\_SHAPE** | | | |
| oldContainerId | required | list | A list that contains the ids linking to the old container. The last id in this list is the UUID of the old container. The other ids are of its parental hierarchy. See examples below for more details |
| newContainerId | required | list | A list that contains the ids linking to the old container. The last id in this list is the UUID of the old container. The other ids are of its parental hierarchy. See examples below for more details |
| movedElements | required | list | A list that contains the ids linking to the elements that are being moved. |
| **REMOVE\_OBJECT** | | | |
| id | required | UUID | The UUID of the element being removed from the sketch. |
| **ASSIGN\_ATTRIBUTE** | | | |
| id | required | UUID | The UUID of the element that is affected |
| key | required | string | The value of this argument is the key of the attribute being added. |
| value | required | any | The value of this argument is the value of the attribute being added. |
| **REMOVE\_ATTRIBUTE** | | | |
| id | required | UUID | The UUID of the element that is affected |
| key | required | string | The value of this argument is the key of the attribute being added. |
| **MARKER** | | | |
| markerType | required | string | Currently supported markers are: “SUBMISSION”, “FEEDBACK”, “SAVE”, “SPLIT”. See below for examples. |
| arguments | zero or more | object | extra data needed for a marker. |
| **CLEAR** | | | |
| ids | zero or more | list | optional list of ids of the elements being cleared. This should clear the entire sketch as it currently is shown at the time this edit occurs. But that may not contain all elements in the lifetime of the sketch. This list is meant to capture that discrepancy |
| **CREATE\_SKETCH** | | | |
| id | required | UUID | This is the id of the sketch that is being created. |
| sketchType | zero or one | string | the type of sketch, may be used for special situations. example types: “RESPONSE”, “TEXT” |
| x | zero or one | int | its location in relation to the parent sketch. [-1 means it is the same value as the parent sketch] |
| y | zero or one | int | its location in relation to the parent sketch. [-1 means it is the same value as the parent sketch] |
| width | zero or one | int | its size in relation to the parent sketch. [-1 means it is the same value as the parent sketch] |
| height | zero or one | int | its size in relation to the parent sketch. [-1 means it is the same value as the parent sketch] |
| domain | zero or more | list | list of strings of the domain(s) that the sketch is a part of. [Taken from the main table] |
| background | zero or one | string | a string representing the background. [A formal standard list of background names have not been composed yet! Please contribute this list] |
| **SWITCH\_SKETCH** | | | |
| id | required | UUID | The id of the sketch that is being switched to. |

Appendix D: Trigger types

|  |  |
| --- | --- |
| **Type** | **Description** |
| onHover | Input device is hovering hovering over the screen |
| onClick | A click has been performed |
| onRecognize | A shape has been recognized from the current sketch |
| onDrag | Dragging a stroke or shape across the screen |
| onDragStart | Beginning of a drag edit |
| onDragEnd | Ending of a drag edit |
| onPenDown | Input device is touching the screen |
| onPenUp | Input device has been lifted from screen |

Note that this list is not comprehensive. Any number of triggers could be added and implementations of SketchML::JSON should have the ability to ignore edit and trigger types they don’t understand.

Appendix E: Official Attributes

attributes are properties of a shape or stroke that may not already be listed in table. This is to allow for instant extension of what data is supported without requiring a change to the parser.

This list can be extended for any use case that is deemed necessary.

Not every object will have all or any of these attributes.

Every attribute is completely optional.

This list is specifically created to show support for the existing SketchML standard. We believe these values do not require 1st class support and instead should just be a list of attributes that may or may not exist. Additionally this list the types of these attributes so that everyone knows what is the expected types of these keys.

|  |  |  |
| --- | --- | --- |
| Key | type | description |
| **Text related keys** |  |  |
| text | string | the text content of a shape |
| isHandWritten | boolean | true if the text of the shape or any sub-shape was handwritten |
| **Graphical related keys** |  |  |
| color | string | hex value of the color |
| laysInk | boolean | if true render the stroke, if false don’t |
| penTip | string | type of tip used to display stroke, ex: rectangle, ball |
| thickness | int | thickness in pixels to display the stroke |
| raster | string | “MaskPen” or “CopyPen” on how to render the ink |
| texture | string | the type of texture being used to render the stroke |
| **Mathematical related keys** |  |  |
| height | double | height of the shape |
| width | double | height of the shape |
| area | double | area of the shape |
| orientation | double | The angle in degrees that the shape was drawn in relative to its description in the language file |
| **Pointer** |  |  |
| p1 | string | UUID of the first point in the stroke |
| p2 | string | UUID of the endpoint in the stroke |
| leftx | double | left most x of the object |
| topy | double | top most y of the object |
| control1 | string | points for dragging (UUID) |
| control2 | string | points for dragging (UUID) |
| start | string | from SketchML |
| end | string | from SketchML |
| **Other** |  |  |
| source | string | from SketchML (Maybe device? Check up on these) |