Potential Actions

What follows is a series of changes to http://schema.org/Action, currently used to describe past actions, to also enable describing the capability to perform an action in the future, as well as how that capability can be exercised.

Status of this Document:

This document revises the April 7th Potential Actions draft, addressing comments received during final review <http://lists.w3.org/Archives/Public/public-vocabs/2014Apr/0065.html>.

The following changes were made:

- Revised the actionStatus mechanism. Instead of an all purpose 'action' property (inverse of 'object'), we use potentialAction for actions that are potential. This removes the need to

specify a default actionStatus and clarifies that potential action descriptions serve as templates.
- We introduce a 'target' property that indicates an EntryPoint, instead of overloading 'url' for this purpose. Using a dedicated property also allows for clearer documentation, without complicating the description of 'url'.
- Several commentators noted that ProtocolElement added no value, so we have removed it.
- Changed Entrypoint to EntryPoint (several people thought the lowercase spelling was a mistake).
- This version introduces an httpMethod property for EntryPoint.
- Added an urlTemplate property to EntryPoint, to more clearly distinguish URLs from URL Templates.
- Renamed the three ActionStatusType types from PotentialAction, ActiveAction, CompletedAction to PotentialActionStatus, ActiveActionStatus, CompletedActionStatus. This clarifies that these are not classes of Action, and avoids having 'potentialAction' and 'PotentialAction' names differ only by case.
- Replaced the '/' character used for '/input', '/output' with '-', i.e. 'xyz/input' becomes 'xyz-input'. This addresses a concern that such property names can't be serialized in W3C RDF/XML format, as well as making these annotations more distinct from other kinds of schema.org extension.

Part 1: Action status

First, we need a mechanism for differentiating potential actions from actions that have actually taken place or are even still in-progress. The expectation is that the status of an action will often be self-evident based on the usage context, so this property can often be elided. However, it may still be necessary for resolving ambiguous cases when they arise. For this we introduce a new property of Action called "actionStatus.

Thing > Action

Property	Expected Type	Description
actionStatus	ActionStatusType	Indicates the current disposition of the Action.

Thing > Intangible > Enumeration > ActionStatusType

- **PotentialActionStatus** - A description of an action that is supported
- **ActiveActionStatus** - An in-progress action (e..g, while watching the movie, or driving to a location)
- **CompletedActionStatus** - An action that has already taken place.

Example: actionStatus

```
{
  "@context": "http://schema.org",
  "@type": "WatchAction",
  "actionStatus": "CompletedActionStatus",
  "agent" : {
    "@type": "Person",
    "name": "Kevin Bacon"
  },
  "object" : {
    "@type": "Movie",
    "name": "Footloose"
  },
  "startTime" : "2014-03-01"
}
```

Part 2: Connecting Actions to Things

Frequently actions are taken or offered in the context of an object (e.g., watch this movie, review this article, share this webpage, etc.). We introduce a new property called potentialAction for describing the "prototype" of an action that can be taken on that Thing.

Thing

Property	Expected Type	Description
potentialAction	Action	Indicates a potential Action, which describes an idealised action in which this thing would play the 'object' role.

Example: Thing.potentialAction

```
{
  "@context": "http://schema.org",
  "@type": "Movie",
  "name": "Footloose",
  "potentialAction" : {
    "@type": "WatchAction"
  }
}
```

Part 3: Action EntryPoints

Potential actions are materialized via execution against the target EntryPoint of an Action.

Example: Action target URL

```
{
  "@context": "http://schema.org",
  "@type": "Movie",
  "name": "Footloose",
  "potentialAction" : {
    "@type": "WatchAction",
    "target" : "http://example.com/player?id=123"
  }
}
```

For some platforms and use cases, however, a simple URL is insufficient for formulating a request and/or processing the result, so we are introducing a new EntryPoint class for specifying that additional context beyond a URL when necessary.

Action

Property	Expected Type	Description
target	EntryPoint	An entrypoint used to execute the action.

Thing > Intangible > EntryPoint

Property	Expected Type	Description
urlTemplate	Text	An url template (RFC6570) that will be used to construct the target of the execution of the action.
encodingType	Text	Supported MIME type(s) for the request
contentType	Text	Supported MIME type(s) of the response
httpMethod	Text	An HTTP method that specifies the appropriate HTTP method for a request to an HTTP Entrypoint. Values are capitalized strings as used in HTTP.
application	SoftwareApplication	The host application

Scheme-based encoding of EntryPoint

Ideally, the simple "deep link" use cases should work with just a simple URL template. In some cases, new schemes might even be created to make that possible for some platforms, for example:

```
android-app://{package_id}/{scheme}/{host_path}
```

Example: Multiple platform URLs

Note: we expect the detail of platform-specific bindings to evolve and be clarified through implementation experience. These examples indicate the general approach rather than the exact information needs of each platform. Also note that the example syntax shown here is not strictly JSON; inline comments have been added for readability.

```
{
  "@context": "http://schema.org",
  "@type": "Restaurant",
  "name": "Tartine Bakery",
  "potentialAction": {
    "@type": "ViewAction",
    "target": [
      /* Web deep link */
      "http://www.urbanspoon.com/r/6/92204",

      /* HTTP API that returns JSON-LD */
      {
        "@type": "EntryPoint",
        "urlTemplate": "http://api.urbanspoon.com/r/6/92204",
        "contentType": "application/json+ld"
      },

      /* Android app deep link */
      "android-app://com.urbanspoon/http/www.urbanspoon.com/r/6/92204",

      /* iOS deep link */
      {
        "@type": "EntryPoint",
        "urlTemplate": "urbanspoon://r/6/92204",
        "application": {
          "@type": "SoftwareApplication",
          "@id": "284708449",
          "name": "Urbanspoon iPhone & iPad App",
          "operatingSystem": "iOS"
```

```
        }
      },

      /* Windows Phone deep link */
      {
        "@type": "EntryPoint",
        "urlTemplate": "urbanspoon://r/6/92204",
        "application": {
          "@type": "SoftwareApplication",
          "@id": "5b23b738-bb64-4829-9296-5bcb59bb0d2d",
          "name": "Windows Phone App",
          "operatingSystem": "Windows Phone 8"
        }
      }
    ]
  }
}
```

Part 4: Input and Output constraints

Additional information is often required from a user or client in order to formulate a complete request. To facilitate this process we need the ability to describe within a potential action how to construct these inputs. Since we need this capability for filling in *any* property of an Action, we introduce a notion of property annotations using a hypen ("-") delimiter. For example, by specifying a "location-input" property on a potential action we are indicating that "location" is a supported input for completing the action.

Similarly, it is also helpful to indicate to clients what will be included in the final completed version of an action, so we introduce the corresponding -output annotation for indicating which properties will be present in the completed action.

Annotation	**Expected Type**	**Description**
<property>-input	PropertyValueSpecification	Indicates how a property should be filled in before initiating the action.
<property>-output	PropertyValueSpecification	Indicates how the field will be filled in when the action is completed.

Thing > Intangible > PropertyValueSpecification

A property value specification.

Property	Expected Type	Description
valueRequired	Boolean	Whether the property must be filled in to complete the action. Default is false. Equivalent to HTML's input@required.
defaultValue	Thing, DataType	The default value for the property. For properties that expect a DataType, it's a literal value, for properties that expect an object, it's an ID reference to one of the current values. Equivalent to HTML's input@value.
valueName	Text	Indicates the name of the PropertyValueSpecification to be used in URL templates and form encoding in a manner analogous to HTML's input@name.
readonlyValue	Boolean	Whether or not a property is mutable. Default is false. Equivalent to HTML's input@readonly. Specifying this for a property that also has a value makes it act similar to a "hidden" input in an HTML form.
multipleValues	Boolean	Whether multiple values are allowed for the property. Default is false. Equivalent to HTML's input@multiple.
valueMinLength	Number	Specifies the minimum number of characters in a literal value. Equivalent to HTML's input@minlength.
valueMaxLength	Number	Specifies the maximum number of characters in a literal value. Equivalent to HTML's input@maxlength.
valuePattern	Text	Specifies a regular expression for testing literal values Equivalent to HTML's input@pattern.
minValue	Number, Date, Time, DateTime	Specifies the allowed range and intervals for literal values. Equivalent to HTML's input@min, max, step.The lower value of some characteristic or property
maxValue	Number, Date, Time, DateTime	The upper value of some characteristic or property. Equivalent to HTML's input@min, max, step.
stepValue	Number	The step attribute indicates the granularity that is expected (and required) of the value.

The minValue, maxValue and stepValue properties specify the allowed range and intervals for literal values and are equivalent to HTML's input@min, max, step. It should also be noted that if

both a property and its -input annotation are present, the value of the un-annotated property should be treated as the allowed options for input (similar to <select><option> in HTML) unless the Input indicates that the value is also readonly, in which case the value(s) should all be returned in a manner analogous to hidden inputs in forms.

Textual representations of Input and Output

For convenience, we also support a textual short-hand for both of these types that is formatted and named similarly to how they would appear in their HTML equivalent. For example:

```
"<property>-input": {
  "@type": "PropertyValueSpecification",
  "valueRequired": true,
  "valueMaxlength": 100,
  "valueName": "q"
}
```

Can also be expressed as:

```
<property>-input: "required maxlength=100 name=q"
```

URI Templates

Finally, we also allow URI templating (using RFC6570) for inlining the resulting value of -input properties into action URLs. The allowed references in the templates for substitution are dotted schema paths to the filled-in properties (relative to the Action object).

Example: Text search deep link with -input

description

```
{
  "@context": "http://schema.org",
  "@type": "WebSite",
  "name": "Example.com",
  "potentialAction": {
    "@type": "SearchAction",
    "target": "http://example.com/search?q={q}",
    "query-input": "required maxlength=100 name=q"
  }
}
```

request

```
GET http://example.com/search?q=the+search
```

Example: Product purchase API call with -output

description

```
{
  "@type": "Product",
  "url": "http://example.com/products/ipod",
  "potentialAction": {
    "@type": "BuyAction",
    "target": {
      "@type": "EntryPoint",
      "urlTemplate": "https://example.com/products/ipod/buy",
      "encodingType": "application/ld+json",
      "contentType": "application/ld+json"
    },
    "result": {
      "@type": "Order",
      "url-output": "required",
      "confirmationNumber-output": "required",
      "orderNumber-output": "required",
      "orderStatus-output": "required"
    }
  }
}
```

request

```
POST https://example.com/products/ipod/buy
```

response

```
{
  "@type": "BuyAction",
  "actionStatus": "CompletedActionStatus",
  "object": "https://example.com/products/ipod",
  "result": {
    "@type": "Order",
    "url": "http://example.com/orders/1199334"
    "confirmationNumber": "1ABBCDDF23234",
    "orderNumber": "1199334",
    "orderStatus": "PROCESSING"
  },
}
```

Example: Movie review site API with -input and -output

description

```
{
  "@context": "http://schema.org",
  "@type": "ReviewAction",
  "target": {
    "@type": "EntryPoint",
    "urlTemplate": "https://api.example.com/review",
    "encodingType": "application/ld+json",
    "contentType": "application/ld+json"
  },
  "object" : {
    "@type": "Movie",
    "url-input": "required",
  },
  "resultReview": {
    "url-output": "required",
    "reviewBody-input": "required",
    "reviewRating": {
      "ratingValue-input": "required"
    }
  }
}
```

request

```
POST https://api.example.com/review
{
  "@context": "http://schema.org",
  "@type": "ReviewAction",
  "object" : {
    "@id": "http://example.com/movies/123"
  },
  "resultReview": {
    "reviewBody": "yada, yada, yada",
    "reviewRating": {
      "ratingValue": "4"
    }
  }
}
```

response

```
{
  "@context": "http://schema.org",
  "@type": "ReviewAction",
```

```
  "actionStatus": "CompletedActionStatus",
  "resultReview" : {
    "url": "http://example.com/reviews/abc"
  }
}
```