Actions in Schema.org - Draft 3 (June 2013)

Authors: yaar@google.com, steve.macbeth@microsoft.com, goto@google.com
Last modified: 6/27/2013 by yaar@google.com

Status: This document relates to Actions in Schema.org Proposal (May 2013): http://www.w3.org/wiki/File:ActionsinSchema.org2013-05-11.pdf. Based on community feedback, the Schema.org team decided on 6/4/2013 to stage the development of the proposal by first standardizing "completed" actions. Addressing "potential" actions is deferred to a subsequent proposal.

Changes from ActionsinSchema.org2013-05-11.pdf includes:

1. Scope of document limited to "completed" actions.
2. Extensibility is designed around RDF metadata.
3. Exhaustive list of core actions.
4. Introduction of Action.object, target & result properties which are "rdfs:subPropertyOf"-ized in sub-actions.

Table of Contents:

Overview

This document proposes the introduction of "verbs" to the schema.org vocabulary, in the form of a new **Class** of **Things**, called **Actions**. This proposal includes:

- Base type Thing > Action
- Core set of Action sub-types.
- Mechanism for defining custom actions.

Motivations:
The use case this proposal addresses is the recording of actions performed as they are performed on things. Use cases include:

1. A confirmation email describing the completion of an action on a website.
2. A activity stream for a user describing a history of actions performed by the user.

While not in the scope of this proposal, Actions are designed to also address:

- Declaring the actionable purpose of web forms and hyperlinks.
- Actionable data: An event could describe how to RSVP to the event. A movie could describe how to buy tickets for the movie. A website could describe how to register for the site.
- Delegation of action execution between multiple applications.

These are planned to be covered in a follow-up proposal.

Basic Example - Activity Streams:

A web page may present activities performed by a user over a certain span of time. For instance, the user "John" may have "read a book", "shared an article" and "bought a camera" activities. The following markup may be used to describe these activities inline:

```
<html>
...
   <div itemscope itemtype="http://schema.org/ReadAction">
      <span itemprop="performedBy">John</span> read
      <span itemprop="object" itemtype="http://schema.org/Book">
      The Lord of the Rings</span>
   <div>
   <div itemscope itemtype="http://schema.org/ShareAction">
      <span itemprop="performedBy">John</span> shared an
      <a itemprop="object"
         href="http://www.nytimes.com/2013/06/09/article34.html">article</a>
      with <span itemprop="sharedWith">Jane</a>.
   <div>
   <div itemscope itemtype="http://schema.org/BuyAction">
      <span itemprop="performedBy">John</span> bought a
      <div itemprop="bought" itemscope itemtype="http://schema.org/Product">
         <span itemprop="name">Canon PowerShot S110</span>
      </div>
   <div>
...
</html>
```

New Type: Thing > Action

A base type for all schema.org Actions. Whereas most of schema.org types represent "Nouns", an action represents a "Verb", making it possible to describe various activities performed on schema.org types.

An action (past, present, future or potential) performed by some entity; typically on some other 'object', sometimes 'performedWith' another person or tool, and having some 'target' and/or 'result'. Specific action sub-type documentation specifies the exact meaning of such Action roles in each case.

The Action type is not intended to be used directly since it doesn't describe the specific action that was performed. Sub-classes of actions should be used instead.

Property	**Type**	**Description**
name	Text	(inherited from Thing) The name of the action.
description	Text	(inherited from Thing) A short description of the action.
image	URL	(inherited from Thing) URL of an image of the action.
url	URL	(inherited from Thing) URL of the action.
performedBy	Thing	Whoever/whatever performed the action.
performedWith	Thing	People or tools that participated in performing the action which are not the primary performer specified by performedBy.
startTime	Datetime	When the action was performed: start time. This is for actions that span a period of time.
endTime	Datetime	When the action was performed: end time. This is for actions that span a period of time.
location	Place	Where the action was performed.
object	Thing	The main item directly affected by the action. The precise meaning of the property is dependent on the action sub-type. Will often be the things whose relation to the action answers the question "what did the action was performed on". Sub-actions may introduce more specific *object* properties that are **rdfs:subPropertyOf** *object* to eliminate ambiguity.
target	Thing	The target item of the activity. The precise meaning of the property is dependent on the action sub-type. Will often be the things whose relation to the action is described with the English propositions "to", "in", "with" or "for". Sub-actions may introduce more specific *target* properties that are **rdfs:subPropertyOf** *target* to eliminate ambiguity.
result	Thing	Item that is the result of the performed action. The precise meaning of the property is dependent on the action sub-type. Sub-actions may introduce more specific *result* properties that are **rdfs:subPropertyOf** *result* to reduce ambiguity.
updatedTime	Datetime	When the action information was last updated

createdTime	Datetime	When the action information was created. It might be different from the action's start or end time if the action's information was reported at a different time.
creator	Thing	The application, service or business which created the action.

Note: The Action type is based on the Event Ontology[1] as well as Activity objects modeled by ActivityStrea.ms[2].

Action Sub-Types

Following is a core list of Action sub-types proposed for schema.org. The list has been compiled based on the base list of verbs in ActivityStrea.ms[3].

Some of the actions in the list below have specialized properties. These properties are denoted by a $^{\text{object}}$, $^{\text{target}}$ or $^{\text{resul}}$ superscript, in which case they are **rdfs:subPropertyOf** *Action.object*, *Action.target, Action.result* respectively.

Action	**Additional Properties**
AcceptAction	
AccessAction	
AcknowledgeAction	
AddAction	added$^{\text{object}}$ (Thing), addedTo$^{\text{target}}$ (Thing)
AgreeAction	
AppendAction	appended$^{\text{object}}$ (Thing), appendedTo$^{\text{target}}$ (Thing)
ApproveAction	
ArchiveAction	
AssignAction	assigned$^{\text{object}}$ (Thing), assignedTo$^{\text{target}}$ (Thing)
AtAction	atLocation$^{\text{object}}$ (Place)
AttachAction	attached$^{\text{object}}$ (Thing), attachedTo$^{\text{target}}$ (Thing)
AttendAction	

[1] http://motools.sourceforge.net/event/event.html
[2] http://activitystrea.ms/specs/json/1.0/
[3] http://activitystrea.ms/specs/json/schema/activity-schema.html#verbs

AuthorAction	
AuthorizeAction	
BefriendAction	
BikeAction	
BorrowAction	
BuildAction	
BuyAction	bought^{object} (Thing), order^{result} (Order[4]), recipient^{target} (Person or Org)
CancelAction	
CheckInAction	checkedInto^{object} (Thing)
CloseAction	
CompleteAction	
CommentAction	comment^{result} (Comment)
ConfirmAction	
ConsumeAction	
CreateAction	created^{result} (Thing)
DeleteAction	deletedFrom^{target}
DeliverAction	
DenyAction	
DisagreeAction	
DiscoverAction	
DislikeAction	
DownloadAction	
DriveAction	vehicle^{object} (Thing), course^{target} (Thing), source (Place), destination (Place), averageSpeed (Text), passenger (Person)
EditAction	
ExperienceAction	

[4] From http://www.w3.org/wiki/WebSchemas/OrdersSchema

FindAction	
FollowAction	
GiveAction	
HostAction	
IgnoreAction	
InsertAction	
InstallAction	
InteractAction	interactedWith^{object} (Thing)
InviteAction	event^{target} (Event), guest^{object} (Person)
JoinAction	
LeaveAction	
LikeAction	
ListenAction	listenedTo^{object} (MusicRecording or AudioObject), device (Thing)
LogInAction	service^{object} (Text)
LogOutAction	service^{object} (Text), sessionLength (Number)
LoseAction	
OpenAction	
PickUpAction	
PlayAction	
PresentAction	
QualifyAction	
ReadAction	
ReceiveAction	
RejectAction	
RemoveAction	deletedFrom^{target} (Thing)
RenewAction	

RentAction	
ReplaceAction	replaced^{object} (Thing), replacedBy (Thing)
RequestAction	
ReserveAction	reservation^{result} (Reservation[5])
ResolveAction	
RetractAction	retracted^{object} (Thing), retractedFrom^{target} (Thing)
ReturnAction	returned^{object} (Thing), returnedTo^{target} (Thing)
ReviewAction	review^{result} (Review)
RideAction	
RsvpAction	event^{object} (Event), guest (Person), rsvpStatus^{result} (RsvpStatus), comment (Comment), bringingOthers (Number), bringingKids (Number)
RunAction	
SatisfyAction	
SaveAction	
ScheduleAction	
SearchAction	query (Text)
SellAction	
SendAction	
ShareAction	sharedWith^{target} (Thing)
ShipAction	
SponsorAction	
SubmitAction	
SubscribeAction	
TagAction	tag (Text)
TerminateAction	

[5] From http://www.w3.org/wiki/WebSchemas/ReservationsSchema

TieAction	
TravelAction	
UpdateAction	
UseAction	
ViewAction	
VisitAction	
WalkAction	
WantAction	
WatchAction	
WinAction	

Custom Actions

Whereas a performed action cannot be described by any of the core action sub-types listed above, it is possible to customize existing actions, or even declare new custom actions.

Custom action are declared by introducing a type that extends an existing action. It is required that the new type URL points at a downloadable RDF[6] descriptor for the action. The RDF describes a type that is a subClassOf one the core Schema.org Action types. The RDF may specify additional properties, and possibly mark them as rdfs:subPropertyOf existing properties of Action.

Example: Netflix "Add To Queue" Action

Netflix allows its users to add movies to their movie rentals queue, and would like to model that as a custom action. Such action could extend AddAction, which is semantically similar. Here is how the definition could be:

- Action: Thing > Action > AddAction > AddToQueueAction
- Description: "The act of adding a movie to a netflix movie queue"
- Action Type: http://schemas.netflix.com/AddToQueueAction
- Properties: movie (Movie), queuePosition (Integer)

For example, *"John added the movie 'The Matrix' to his Netflix queue at position #1 at 6/12/2013 9:46pm PST"* would be encoded as follows (in JSON-LD format):

```
{
  '@vocab': 'http://schema.org/',
  '@type': 'http://schemas.netflix.com/AddToQueueAction',
  'performedBy': { '@type': 'Person', 'email': 'john@acme.com' }
  'actionTime': '2013-06-12 21:46:00 PST',
  'movie': { '@type': 'Movie', 'name': 'The Matrix' }
  'queuePosition': '1',
}
```

To declare the AddToQueueAction, Netflix will have to host the following RDF metadata at *http://schemas.netflix.com/AddToQueueAction*:

```
<rdf:RDF>
    <!-- declaration of AddToQueueAction type -->
    <rdf:Description rdf:about="http://schemas.netflix.com/AddToQueueAction"/>
        <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
        <rdfs:label xml:lang="en">Add To Netflix Queue</rdfs:label>
```

[6] http://www.w3.org/RDF/ and http://www.w3.org/TR/rdf-schema/. Note that schema.org also allows type definition via http://schema.org/Class and http://schema.org/Property, but these are not yet fully featured.

```
        <rdfs:comment xml:lang="en">
        The act of adding a movie to a netflix movie queue
        </rdfs:comment>
        <rdfs:subClassOf rdf:resource="http://schema.org/AddAction"/>
    </rdf:Description>

    <!-- declaration of 'movie' property -->
    <rdf:Description rdf:about="http://schemas.netflix.com/movie"/>
        <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
        <rdfs:label xml:lang="en">Movie</rdfs:label>
        <rdfs:domain rdf:resource="http://schemas.netflix.com/AddToQueueAction"/>
        <rdfs:range rdf:resource="http://schema.org/Movie"/>
        <!-- Declaration of the movie property as the 'object' of the action: -->
        <rdfs:subPropertyOf rdf:resource="http://schema.org/object"/>
    </rdf:Description>

    <!-- declaration of 'queuePosition' property -->
    <rdf:Description rdf:about="http://schemas.netflix.com/queuePosition"/>
        <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
        <rdfs:label xml:lang="en">Queue Position</rdfs:label>
        <rdfs:subPropertyOf rdf:resource="http://schema.org/target"/>
        <rdfs:domain rdf:resource="http://schemas.netflix.com/AddToQueueAction"/>
        <rdfs:range rdf:resource="http://wwww.w3.org/2001/XMLSchema#integer"/>
    </rdf:Description>

</rdf:RDF>
```

Examples

Sentence	**Schema**
John bought a book	`{ @type: BuyAction performedBy: "John" object: { @type: Book } }`
John read a book	`{ @type: ReadAction performedBy: "John" object: { @type: Book } }`
John listened to a music on an ipod	`{ @type: ListenAction performedBy: "John" object: { @type: MusicRecording } device: "iPod" }`
John reviewed an article resulting in a review	`{ @type: ReviewAction performedBy: "John" object: { @type: Article, url: ... } review: { @type: Review, ... } }`
John created a playlist	`{ @type: CreateAction performedBy: "John" created: { @type: Playlist } }`
John added a song to a playlist	`{ @type: AddedAction performedBy: "John" added: { @type: Song, ... } addedTo: { @type: Playlist, ... } }`
John watched a movie	`{ @type: WatchAction performedBy: "John" object: { @type: Movie, ... } }`

John commented on an article	`{`
	`  @type: CommentAction`
	`  performedBy: "John"`
	`  target: { @type: Article, ... }`
	`  comment: { @type: Comment, ...}`
	`}`
	
	`-or-`
	
	`{`
	`  @type: CommentAction`
	`  performedBy: "John"`
	`  comment: {`
	`    @type: Comment,`
	`    about: {@type: Article, ... }`
	`  }`
	`}`

John reviewed are restaurant	`{`
	`  @type: ReviewAction`
	`  performedBy: "John"`
	`  object: { @type: Restaurant, ... }`
	`  review: { @type: Review, ...}`
	`}`
	
	`-or-`
	
	`{`
	`  @type: ReviewAction`
	`  performedBy: "John"`
	`  review: {`
	`    @type: Review,`
	`    about: {@type: Restaurant, ... }`
	`  }`
	`}`

John liked article	`{`
	`  @type: LikeAction`
	`  performedBy: "John"`
	`  object: { @type: Article, ... }`
	`}`

John saved a recipe	`{`
	`  @type: SaveAction`
	`  object: { @type: Recipe, ... }`
	`}`

John drove his kids to school	`{`
	`  @type: DriveAction`
	`  performedBy: "John"`
	`  passenger: { "Kid1", "Kid2" }`
	`  destination: "School"`
	`}`

John drove the Golden Gate Bridge in his BMW at 60mph	`{`
	`  @type: DriveAction`
	`  performedBy: "John"`
	`  course: "Golden Gate Bridge"`
	`  vehicle: "BMW"`
	`  averageSpeed: "60mph"`
	`}`

John ate a cookie	`{`
	`  @type: EatAction`
	`  performedBy: "John"`
	`  object: "Cookie"`
	`}`