Apache OpenWhisk Package Specification

Version 0.9, Working Draft 03, Revision 1

Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The OpenWhisk Package Specification is licensed under The Apache License, Version 2.0.

Introduction

Apache OpenWhisk™ is an open source, distributed Serverless computing project. Specifically, it is able to execute application logic (*Actions*) in response to events (*Triggers*) from external sources (*Feeds*) governed by simple conditional logic (*Rules*) around the event data.

It provides a programming model for registering and managing *Actions*, *Triggers* and *Rules* supported by a REST-based Command Line Interface (CLI) along with tooling to support packaging and catalog services.

The project includes a catalog of built-in system and utility *Actions* and *Feeds*, along with a robust set of samples that demonstrate how to integrate OpenWhisk with various external service providers (e.g., GitHub, Slack, etc.) along with several platform and run-time Software Development Kits (SDKs).

The code for the Actions, along with any support services implementing *Feeds*, are packaged according to this specification to be compatible with the OpenWhisk catalog and its tooling. It also serves as a means for architects and developers to model OpenWhisk package Actions as part of full, event-driven services and applications providing the necessary information for artifact and data type validation along with package management operations.

Compatibility

This specification is intended to be compatible with the following specifications:

- *OpenWhisk API which is defined as an OpenAPI document:*
    - https://github.com/apache/openwhisk/blob/master/core/controller/src/main/resources/apiv1swagger.json
    -
- *OpenAPI Specification when defining REST APIs and parameters:*
    - https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md

37 Revision History

Version	Date	Notes
0.8.1	2016-11-03	• Initial public point draft
0.8.2	2016-12-12	• Begin adding use cases, examples to all sections
0.8.3	2017-02-02	• Add more use cases, examples • Add description of $ (dollar) notation
0.8.4	2017-04-18	• Support JSON parameter type; • Clarify use of Parameter single-line grammar and inferred types. • Add support for API Gateway mappings. • Add support for Web Actions
0.8.5	2017-04-21	• Add support for "dependencies", that is allow automatic deployment of other OpenWhisk packages (from GitHub) that the current package declares as a dependency.
0.8.6	2017-07-25	• Clarified requirements for $ dollar notation. • Updated conceptual Manifest/Deployment File processing images.
0.8.7	2017-08-24	• Added explicit Application entity and grammar. • Added API listing to Package entity. • Cleaned up pseudo-grammar which contained various uses of credentials in places not intended. • Fixed Polygon Tracking example (indentation incorrect).
0.8.8	2017-08-29	• Created a simplified API entity (i.e., "api") grammar that allows multiple sets of named APIs for the same basepath. • Acknowledge PHP as supported runtime (kind). • Added "sequences" entity as a convenient way to declare action sequences in the manifest. Updated supported runtime values.
0.8.9, 0.8.9.1	2017-09-22 2017-09-29	• Clarified "version" key requirements for Package (required) and Action (optional); removed from shared entity schema. • Made "license" key optional for package. • keyword "package" (singular) and "packages" (plural) both allowed. • Adjusted use case examples to reflect these changes. • Rework of schema use cases into full, step-by-step examples. • Spellcheck, fixed bugs, update examples to match web-based version.
0.8.9.1	2017-10-06	• Added grammar and example for concatenating string values on input parameters using environment variables.
0.9.0, 0.9.1	2017-11-23, 2017-11-30	• Identified new user scenarios including: clean, refresh, sync, pre/post processing • Clarified "runtime" field on Action is equivalent to "kind" parameter used on the Apache OpenWhisk CLI for Actions. • Added "project" key as an synonym name for "application." key, moving application to become deprecated. Project name made required. • Support "public" (i.e., publish) key on Package. • Documented support for the "raw-http" annotation under Action. • Documented support for the "final" annotation under Action. • Documented support for the "main" field under Action. • Dollar Notation section becomes Interpolation / updates: o Supported beyond Parameter values o Package names can be interpolated o Annotations values can be interpolated o Multiple replacements supported in same value • Usage scenarios 6-8 added, i.e., Clean, Project Sync, Tool chain support.
0.9.2	2018-02-05	• Fixed and clarified the allowed values for "web-export" on Action. • Clarified use of "final" on Action.

		• Added support (planned) for "web-custom-options" and "require-whisk-auth." flags on Actions (annotations) • Deprecate 'application' and 'package' completely (no longer supported).
0.9.2.2	2018-04-04	• Allow "web" key as an overload for "web-export" key for to indicate Web Actions. • Added Web Sequences, specify a sequence is a Web Action. • Added support for Conductor Actions, to align with OpenWhisk CLI support. • Added "docker" and "native" binary support under Action. • Added in-line "code" support under Action. • Support $$, double-dollar notation for string literals on parameter values. • Added support for "default" package (allowing all entities to be assigned directly under the user's default namespace), that is not requiring a package name to be created.
0.9.3	2018-06-01	• Added comprehensive list of schema keys and values that support interpolation (from environment). • Made "code" a deprecated field for Action. • Fix web	web-export grammar under Action schema. • Promote Web Action to its own top-level section. • Correct "plain" Credential properties.

38

Table of Contents

Programming model

OpenWhisk Entities

OpenWhisk uses the following entities to describe its programming model:

Action

A stateless, relatively short-running function (*on the order of seconds or even milliseconds*) invoked as an event handler.

Trigger

The name for a class of events. Triggers represent the events (and their data) themselves without any concept of how they were generated.

Rule

A mapping from a Trigger to an Action which may contain simple conditional logic. OpenWhisk evaluates incoming events (that belong to a Trigger) and invokes the assigned Action (event handler).

Event Source

An Event Source is the descriptor (edge) for an Event Producer (or provider). It describes the Event Format(s) produced, as well as any configuration and subscription capabilities.

Feed

A Feed is an optional service that represents and controls the stream which all belong to a Trigger. A feed provides operations called **feed actions** which handle creating, deleting, pausing, and resuming the stream of events. The feed action typically interacts with external services which produce the events

Package

A named, shared collection of Actions and Feeds. The goal of this specification is to describe OpenWhisk packages and their component entities and resources to enable an open-ecosystem.

Packages are designed to be first-class entities within the OpenWhisk platform to be used by tooling such as catalogs (repositories), associated package managers, installers, etc.

Note: Not all actions must belong to packages, but can exist under a namespace.

Composition

Action Sequence

An Action that is a sequenced composition of 2 or more existing Actions. The Action Sequence can be viewed as a named pipe where OpenWhisk can automatically take the output of a first

136 Action 'A' in a declared sequence and provides it as input to the next Action 'B' in the sequence
137 and so on until the sequence completes.
138
139 *Note: This composition technique allows the reuse of existing action implementations treating*
140 *them as "building blocks" for other Actions.*

141 ## Cardinality

142 ### *Trigger to Action*

143 With the appropriate set of Rules, it's possible for a single Trigger (event) to invoke multiple
144 Actions, or for an Action to be invoked as a response to events from multiple Triggers.

145 ## Conceptual representation

146
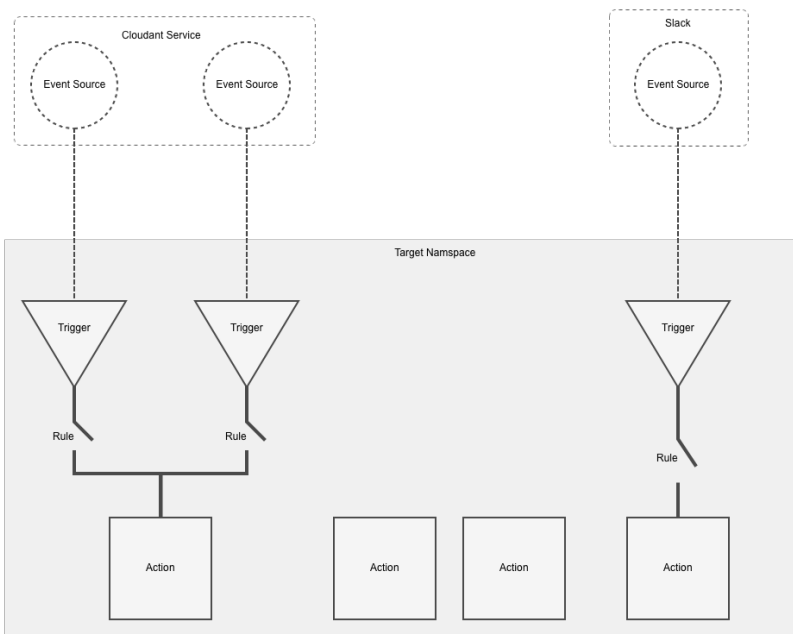
147 ## Package processing

148 This document defines two file artifacts that are used to deploy Packages to a target OpenWhisk
149 platform; these include:
150 • **Package Manifest file**: Contains the Package definition along with any included Action, Trigger or
151 Rule definitions that comprise the package. This file includes the schema of input and output data to
152 each entity for validation purposes.
153 • **Deployment file** (optional): Contains the values and bindings used configure a Package to a target
154 OpenWhisk platform provider's environment and supply input parameter values for Packages,
155 Actions and Triggers. This can include Namespace bindings, security and policy information.

156 ### *Notes*

157 • *Deployment files are optional*. Deployment can be fully accomplished with simply a Manifest File.

158 ## Conceptual Package creation and publishing

159 The following diagram illustrates how a developer would create OpenWhisk code artifacts and
160 associate a Package Manifest file that describes them for deployment and reuse.

1. Developer **creates** and **reposits** Serverless code, including:
   - Actions (functions) and
   - Feeds for Event Sources (i.e., Event Providers)

2. Creates a **Package Manifest File** which describes the Serverless service's:
   - Repositories (source code locations)
   - Parameter schema (for Actions and Feeds)
   - Configuration and Lifecycle APIs for Feeds
   - Event Sources (and corresponding Event schema)
   - Triggers and Rules
   - Compositions of Actions
   - Annotations (tags, User Interface hints, etc.)

3. Publishes **Package Manifests** to enable Catalog features for Serverless services:
   - Automated or manual Discovery and Search
   - Graphical Display / Selection
   - Functional applicability
   - Compositional type validation

161

162 ## Conceptual tooling integration and deployment

163 The following diagram illustrates how Package manifests can be leveraged by developer tooling to
164 integrate OpenWhisk Serverless functions.

1. Developer *searches* and *discovers* OpenWhisk packages described by the *Package Manifest* in one or more Catalogs, that can:
   - Help analyze, augment and annotate application information and data.
   - Add value added functionality to a base application or workflow.

2. Imports Open *Package Manifest Files* and related code and artifacts into development tooling, including:
   - Project and Application (source code) Repositories
   - Integrated Development Environments (IDEs)
   - Cloud-based design, workflow and application workspaces.

3. Creates OpenWhisk *Deployment Files* for one or more target OpenWhisk enabled Clouds, with
   - Parameter values for desired target environment
   - Appropriate Credentials and configurations for chosen Event Sources and Feeds.

4. Deploys *Packages* (i.e., Actions, Triggers, Feeds, etc.) to OpenWhisk enabled Clouds, using,
   - *Package Manifest* and *Deployment File(s)*.

165
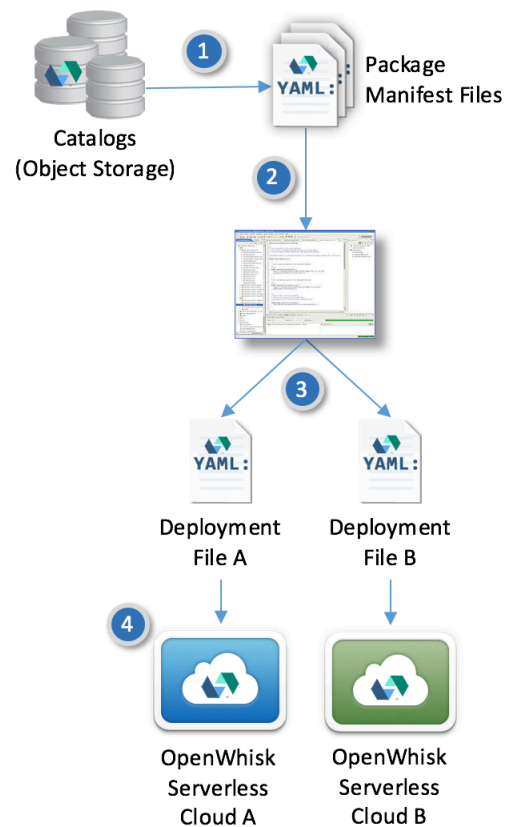
Namespacing

167 Every OpenWhisk entity (i.e., Actions, Feeds, Triggers), including packages, belongs in a
168 *namespace*.

169 The fully qualified name of any entity has the format:

```
/<namespaceName>[/<packageName>]/<entityName>
```

170
171 The namespace is typically provided at bind-time by the user deploying the package to their
172 chosen OpenWhisk platform provider.

Requirements

174 - The "/whisk.system" namespace is reserved for entities that are distributed with the OpenWhisk
175 system.

Entity Names

177 The names of all entities, including actions, triggers, rules, packages, and namespaces, are a
178 sequence of characters that follow the following format:
179 - The first character SHALL be an alphanumeric character, a digit, or an underscore.
180 - The subsequent characters MAY be alphanumeric, digits, spaces, or any of the following:

181   _, @, ., -
182   • The last character SHALL NOT be a space.
183   • The maximum name length of any entity name is 256 characters (i.e., ENTITY_NAME_MAX_LENGTH =
184   256).

185 Valid entity names are described with the following regular expression (Java metacharacter
186 syntax):

```
"\A([\w]|[\w][\w@ .-]{0,${ENTITY_NAME_MAX_LENGTH - 2}}[\w@.-])\z"
```

Definitions

Activation

189 An invocation or "run" of an action results in an activation record that is identified by a unique
190 activation ID. The term Activation is short-hand for the creation of this record and its
191 information.

Repository

193 A location that provides storage for sets of files, as well as the history of changes made to those
194 files.

Project

196 A description of a software application which enables management of its design,
197 implementation, source control, monitoring and testing.

Application

199 A computer program designed to perform a group of coordinated functions, tasks, or activities
200 to achieve some result or user benefit.

[Cloud] Service

202 Any resource, including a functional task, that is provided over the Internet. This includes
203 delivery models such as *Platform as a Service* (PaaS), *Infrastructure as a* Service (IaaS), as well
204 as *Serverless*.

Specification

206 This specification utilizes the YAML language, a superset of JSON, which supports key features
207 for packaging descriptors and configuration information such as built-in data types, complex data
208 types, anchors (relational information), files, comments and can embed other data formats such
209 as JSON and XML easily.

YAML Types

211 Many of the types we use in this profile are *built-in* types from the YAML 1.2 specification (i.e.,
212 those identified by the "tag:yaml.org,2002" version tag).
213

214 The following table declares the valid YAML type URIs and aliases that SHALL be used when
215 defining parameters or properties within an OpenWhisk package manifest:
216

Type Name	Type URI	Notes
string	tag:yaml.org,2002:str (default)	Default type if no type provided
integer	tag:yaml.org,2002:int	Signed. Includes large integers (i.e., long type)
float	tag:yaml.org,2002:float	Signed. Includes large floating point values (i.e., double type)
boolean	tag:yaml.org,2002:bool	This specification uses lowercase 'true' and lowercase 'false'
timestamp	tag:yaml.org,2002:timestamp	ISO 8601 compatible timestamp. See YAML-TS-1.1
null	tag:yaml.org,2002:null	Different meaning than an empty string, map, list, etc.

217 *Requirements*

218 • The 'string' type SHALL be the default type when not specified on a parameter or property
219   declaration.
220 • All 'boolean' values SHALL be lowercased (i.e., 'true' or 'false').

OpenWhisk Types

222 In addition to the YAML built-in types, OpenWhisk supports the types listed in the table below.
223 A complete description of each of these types is provided below.
224

Type Name	Description	Notes
version	string comprised of a version number of the format <MAJOR>.<MINOR>.<PATCH>[-<BUILD> or keywords acknowledged in this specification.	Aligns with Maven format principles, but is a simplification of Maven spec. considerations. Note: found in modern tooling (i.e., "package@version" or "package:version" format). Note: the keyword "latest" is also used as a valid version in this specification.
string256	long length strings (e.g., descriptions)	A string type limited to 256 characters.
string64	medium length strings (e.g., abstracts, hover text)	A string type limited to 64 characters.
string16	short length strings (e.g., small form-factor list displays)	A string type limited to 16 characters.
json	The parameter value represents a JavaScript Object Notation (JSON) data object.	The deploy tool will validate the corresponding parameter value against JSON schema. Note: The implied schema for JSON the JSON Schema (see http://json-schema.org/).
scalar-unit	Convenience type for declaring common scalars that have an associated unit. For example, "10 msec.", "2 Gb", etc.)	Currently, the following scalar-unit subtypes are supported: • scalar-unit.size • scalar-unit.time See description below for details.

`schema`	The parameter itself is an OpenAPI Specification v2.0 **Schema Object** (in YAML format) with self-defining schema.	The schema declaration follows the OpenAPI v2.0 specification for Schema Objects (YAML format). Specifically, see https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md#schemaObject
`object`	The parameter itself is an object with the associated defined Parameters (schemas).	Parameters of this type would include a declaration of its constituting Parameter schema.

225

scalar-unit types

227
228 Scalar-unit types can be used to define scalar values along with a unit from the list of recognized units (a subset of GNU units) provided below.

229 *Grammar*

```
<scalar> <unit>
```

230 In the above grammar, the pseudo values that appear in angle brackets have the following meaning:

231 • `scalar`: is a <u>required</u> scalar value (e.g., integer).
232 • `unit`: is a <u>required</u> unit value. The unit value MUST be type-compatible with the scalar value.

233 *Example*

```
inputs:
  max_storage_size:
    type: scalar-unit.size
    default: 10 GB
  archive_period:
    type: scalar-unit.time
    default: 30 d
```

234 *Requirements*

235
236 • Whitespace: any number of spaces (including zero or none) SHALL be allowed between the scalar value and the unit value.
237
238 • It SHALL be considered an error if either the scalar or unit portion is missing on a property or attribute declaration derived from any scalar-unit type.
239
240 • The case (lower, upper) of the `Unit`'s value is considered significant and SHALL be preserved. For example, the Unit value "kB" is not the same as "KB" or "kb".

241 *Recognized units for sizes (i.e., scalar-unit.size)*

Unit	Description
B	byte
kB	kilobyte (1000 bytes)
MB	megabyte (1000000 bytes)
GB	gigabyte (1000000000 bytes)

Unit	Description
TB	terabyte (1000000000000 bytes)

242 *Example*

```
inputs:
  memory_size:
    type: scalar-unit.size
    value: 256 MB
```

243 *Recognized units for times (i.e., scalar-unit.time)*

Unit	Description
d	days
h	hours
m	minutes
s	seconds
ms	milliseconds
us	microseconds

244 *Example*

```
inputs:
  max_execution_time:
    type: scalar-unit.time
    value: 600 s
```

245 ## Object type example

246 The Object type allows for complex objects to be declared as parameters with an optional
247 validateable schema.

```
inputs:
  person:
    type: object
    parameters:
      <Parameter schema>
```

248 # Schema

249 This section defines all the essential schema used to describe OpenWhisk packages and
250 programming model entities within a manifest.

251 ## General Requirements

252 • All key and field names in this specification SHALL be case sensitive.

253 ## map schema

254 The Map schema is used to define maps of key values within OpenWhisk entities.

255   *Single-line grammar*

```
{ <key_1>: <value_1>, ..., <key_n>: <value_n> }
```

256   *Multi-line grammar*

```
# Where 'key_n' is a type <string> and 'value_n' is type <any>.
<key_1>: <value_1
...
<key_n>: <value_n>
```

257   *Examples*

258   *Single-line*

```
alert_levels: { "high": "red", "medium": "yellow", "low": green }
```

259   *Multi-line*

```
alert_levels:
  "high": "red"
  "medium": "yellow"
  "low": green
```

260

261   ***Parameter schema***

262   The Parameter schema is used to define input and/or output data to be used by OpenWhisk
263   entities for the purposes of validation.

264   *Fields*

Key Name	Required	Value Type	Default	Description
type	no	`<any>`	`string`	Optional valid type name or the parameter's value for validation purposes. By default, the type is `string`.
description	no	`string256`	N/A	Optional description of the Parameter.
value	no	`<any>`	N/A	The optional user supplied value for the parameter. Note: this is not the default value, but an explicit declaration which allows simple usage of the Manifest file without a Deployment file.
required	no	`boolean`	`true`	Optional indicator to declare the parameter as required (i.e., `true`) or optional (i.e., `false`).
default	no	`<any>`	N/A	Optional default value for the optional parameters. This value **MUST** be type compatible with the value declared on the parameter's `type` field.
status	no	`string`	`supported`	Optional status of the parameter (e.g., `deprecated`, `experimental`). By default a parameter is without a declared status is considered supported.

Key Name	Required	Value Type	Default	Description
schema	no	`<schema>`	N/A	The optional schema if the '`type`' key has the value '`schema`'. The value would include a **Schema Object** (in YAML) as defined by the OpenAPI Specification v2.0. This object is based upon the JSON Schema Specification.
properties	no	`<list of parameter>`	N/A	The optional properties if the '`type`' key has the value '`object`'. Its value is a listing of Parameter schema from this specification.

265 *Requirements*

266 • The "schema" key's value MUST be compatible with the value provided on both the "type" and "value"
267 keys; otherwise, it is considered an error.

268 *Notes*

269 • The "type" key acknowledges some popular schema (e.g., JSON) to use when validating the value of
270 the parameter. In the future additional (schema) types may be added for convenience.

271 *Grammar*

272 *Single-line*

```
<parameterName>: <YAML type> | scalar-unit | json
```

273 • Where `<YAML type>` is inferred to be a YAML type as shown in the YAML Types section
274 above (e.g., `string`, `integer`, `float`, `boolean`, etc.).

275 • If you wish the parser to validate against a different schema, then the multi-line grammar
276 MUST be used where the value would be supplied on the keyname "value" and the type (e.g.,
277 'json') and/or schema (e.g., OpenAPI) can be supplied.

278 *Multi-line*

```
<parameterName>:
  type: <any>
  description: <string>
  required: <boolean>
  default: <any>
  status: <string>
  schema: <OpenAPI Schema Object>
```

279 *Status values*

Status Value	Description
`supported` (default)	Indicates the parameter is supported. This is the implied default status value for all parameters.
`experimental`	Indicates the parameter MAY be removed or changed in future versions.
`deprecated`	Indicates the parameter is no longer supported in the current version and MAY be ignored.

Shared Entity Schema

281 The Entity Schema contains fields that are common (shared) to all OpenWhisk entities (e.g.,
282 Actions, Triggers, Rules, etc.).

283 *Fields*

Key Name	Required	Value Type	Default	Description
description	no	string256	N/A	The optional description for the Entity.
displayName	no	string16	N/A	This is the optional name that will be displayed on small form-factor devices.
annotations	no	map of <string>	N/A	The optional annotations for the Entity.

284 *Grammar*

```
description: <string256>
displayName: <string16>
annotations: <map of <string>>
```

285 *Requirements*

286 • Non-required fields MAY be stored as "annotations" within the OpenWhisk framework after they
287   have been used for processing.
288 • "description" string values SHALL be limited to 256 characters.
289 • "displayName" string values SHALL be limited to 16 characters.
290 • "annotations" MAY be ignored by target consumers of the Manifest file as they are considered data
291   non-essential to the deployment of management of OpenWhisk entities themselves.
292   • Target consumers MAY preserve (persist) these values, but are not required to.
293 • For any OpenWhisk Entity, the maximum size of all "annotations" (values) SHALL be 256 characters.

294 *Notes*

295 • Several, non-normative "annotation" keynames and allowed values (principally for User Interface
296   (UI) design and tooling information) may be defined in this specification for optional usage.

Action entity

298 The Action entity schema contains the necessary information to deploy an OpenWhisk function
299 and define its deployment configurations, inputs and outputs.

300 *Fields*

Key Name	Required	Value Type	Default	Description
version	no	version	N/A	The optional user-controlled version for the Action.
function	yes	string	N/A	Required source location (path inclusive) of the Action code either • Relative to the Package manifest file. • Relative to the specified Repository.

Key Name	Required	Value Type	Default	Description
code	no	`string`	`N/A`	This optional field is now replaced by the "`function`" field.
runtime	maybe	`string`	`N/A`	The required runtime name (and optional version) that the Action code requires for an execution environment. *Note May be optional if tooling is allowed to make assumptions about file extensions or infer from functional code..*
inputs	no	`list` of parameter	`N/A`	The optional ordered list inputs to the Action.
outputs	no	`list` of parameter	`N/A`	The optional outputs from the Action.
limits	no	`map` of limit keys and values	`N/A`	Optional map of limit keys and their values. *See section "Valid limit keys" below for a listing of recognized keys and values.*
feed	no	`boolean`	`false`	Optional indicator that the Action supports the required parameters (and operations) to be run as a Feed Action.
web | web-export	no	`<boolean> | yes | no | raw`	`false`	The optional flag (annotation) that makes the action accessible to REST calls *without* authentication. *For details on all types of Web Actions, see: https://github.com/apache/incubator-openwhisk/blob/master/docs/webactions.md*
raw-http	no	`boolean`	`false`	The optional flag (annotation) to indicate if a Web Action is able to consume the raw contents within the body of an HTTP request. **Note**: this option is ONLY valid if "`web`" or "`web-export`" field is set to '`true`'.
docker	no	`string`	`N/A`	The optional key that references a Docker image (e.g., openwhisk/skeleton).
native	no	`boolean`	`false`	The optional key (flag) that indicates the Action is should use the Docker skeleton image for OpenWhisk (i.e., short-form for docker: openwhisk/skeleton).
final	no	`boolean`	`false`	The optional flag (annotation) which makes all of the action parameters that are already defined immutable. **Note**: this option is ONLY valid if "`web`" or "`web-export`" is set to '`true`'.
web-custom-options	no	`boolean`	`false`	The optional flag (annotation) enables a web action to respond to OPTIONS requests with customized headers, otherwise a default CORS response applies.

Key Name	Required	Value Type	Default	Description
require-whisk-auth	no	`boolean`	`false`	The optional flag (annotation) protects the web action so that it is only accessible to an authenticated subject.
main	no	`string`	`N/A`	The optional name of the function to be aliased as a function named "main". ***Note***: *by convention, Action functions are required to be called "main"; this field allows existing functions not named "main" to be aliased and accessed as if they were named "main".*

Requirements

• The Action name (i.e., `<actionName>` MUST be less than or equal to 256 characters.

• The Action entity schema includes all general Entity Schema fields in addition to any fields declared
above.

• Supplying a `runtime` name without a version indicates that OpenWhisk SHOULD use the most
current version.

• Supplying a runtime *major version* without a *minor version* (et al.) indicates OpenWhisk SHOULD use
the most current *minor version*.

• Unrecognized `limit` keys (and their values) SHALL be ignored.

• Invalid values for known `limit` keys SHALL result in an error.

• If the Feed is a Feed Action (i.e., the `feed` key's value is set to `true`), it MUST support the following
parameters:

• **lifecycleEvent**: one of 'CREATE', 'DELETE', 'PAUSE', or 'UNPAUSE'
○ These operation names MAY be supplied in lowercase (i.e., 'create', 'delete', 'pause', etc.).

• **triggerName**: the fully-qualified name of the trigger which contains events produced from this
feed.

• **authKey**: the Basic auth. credentials of the OpenWhisk user who owns the trigger.

• The keyname 'kind' is currently supported as a synonym for the key named '`runtime`'; in the future
it MAY be deprecated.

• When a `code` is specified, `runtime` SHALL be a required field.

Notes

• Input and output parameters are implemented as JSON Objects within the OpenWhisk framework.

• The maximum code size for an Action currently must be less than 48 MB.

• The maximum payload size for an Action (i.e., POST content length or size) currently must be less
than 1 MB.

• The maximum parameter size for an Action currently must be less than 1 MB.

• if no value for runtime is supplied, the value '`language:default`' will be assumed.

Grammar

```
# Note: the optional [.<type>] grammar is used for naming Web Actions.
<actionName>[.<type>]:
  <Entity schema>
  version: <version>
```

```
  function: <string>
  runtime: <name>[@<[range of ]version>]
  inputs:
    <list of parameter>
  outputs:
    <list of parameter>
  limits:
    <list of limit key-values>
  feed: <boolean>
  web | web-export: <boolean> | yes | no | raw
```

329   *Example*

```
my_awesome_action:
  version: 1.0
  description: An awesome action written for node.js
  function: src/js/action.js
  runtime: nodejs@>0.12<6.0
  inputs:
    not_awesome_input_value:
      description: Some input string that is boring
      type: string
  outputs:
    awesome_output_value:
      description: Impressive output string
      type: string
  limits:
    memorySize: 512 kB
    logSize: 5 MB
```

330   *Valid Runtime names*

331   The following runtime values are currently supported by the OpenWhisk platform.

332

333   Each of these runtimes also include additional built-in packages (or libraries) that have been
334   determined be useful for Actions surveyed and tested by the OpenWhisk platform.

335

336   These packages may vary by OpenWhisk release; examples of supported runtimes as of this
337   specification version include:

338

Runtime value	OpenWhisk kind	Docker image name	Description
nodejs	nodejs	nodejsaction:latest	Latest NodeJS runtime
nodejs@6	nodejs:6	nodejs6action:latest	Latest NodeJS 6 runtime
java, java@8	java	java8action:latest	Latest Java language runtime
python, python@2	python:2	python2action:latest	Latest Python 2 language runtime
python@3	python:3	python3action:latest	Latest Python 3 language runtime
swift, swift@2	swift	swiftaction:latest	Latest Swift 2 language runtime

Runtime value	OpenWhisk kind	Docker image name	Description
swift@3	swift:3	swift3action:latest	Latest Swift 3 language runtime
swift@3.1.1	swift:3.1.1	action-swift-v3.1.1:latest	Latest Swift 3.1.1 language runtime
php	php:7.1	action-php-v7.1:latest	Latest PHP language runtime
language:default	N/A	N/A	Permit the OpenWhisk platform to select the correct default language runtime.

339 *Recognized File extensions*

340 Although it is best practice to provide a runtime value when declaring an Action, it is not
341 required. In those cases, that a runtime is not provided, the package tooling will attempt to derive
342 the correct runtime based upon the the file extension for the Action's function (source code file).
343 The following file extensions are recognized and will be run on the latest version of
344 corresponding Runtime listed below:
345

File extension	Runtime used	Description
.js	nodejs	Latest Node.js runtime.
.java	java	Latest Java language runtime.
.py	python	Latest Python language runtime.
.swift	swift	Latest Swift language runtime.
.php	php	Latest PHP language runtime.

346 *Valid Limit keys*

Limit Keyname	Allowed values	Default value	Valid Range	Description
timeout	scalar-unit.time	60000 ms	[100 ms, 300000 ms]	The per-invocation Action timeout. Default unit is assumed to be milliseconds (ms).
memorySize	scalar-unit.size	256 MB	[128 MB, 512 MB]	The per-Action memory. Default unit is assumed to be in megabytes (MB).
logSize	scalar-unit.size	10 MB	[0 MB, 10 MB]	The action log size. Default unit is assumed to be in megabytes (MB).
concurrentActivations	integer	1000	*See description*	The maximum number of concurrent Action activations allowed (per-namespace). *Note: This value is not changeable via APIs at this time.*
userInvocationRate	integer	5000	*See description*	The maximum number of Action invocations allowed per user, per minute. *Note: This value is not changeable via APIs at this time.*

Limit Keyname	Allowed values	Default value	Valid Range	Description
codeSize	scalar-unit.size	48 MB	*See description*	The maximum size of the Action code. *Note: This value is not changeable via APIs at this time.*
parameterSize	scalar-unit.size	1 MB	*See description*	The maximum size *Note: This value is not changeable via APIs at this time.*

347   *Notes*

348   The default values and ranges for limit configurations reflect the defaults for the OpenWhisk
349   platform (open source code). These values may be changed over time to reflect the open source
350   community consensus.

351   *Web (Web-export)*

352   The following values are recognized values for the 'web' (or 'web-export') key:

Value	Alias	Description
true	yes	A value of `true` (or `yes`) enables an Action to be accessible via REST interface without the need for credentials.
false	no	A value of `false` (or `no`) disables an Action as an accessible REST.
raw	N/A	Implies a value of `true` and indicates that the Web Action is able to consume "raw" contents within the body of an HTTP request. Specifically, this means that the `__ow_body` content is encoded in Base64 when the request `Content-Type` is binary.

353   *Notes*

354   • The default `Content-Type` for an HTTP response is `application/json`, and the body can be any
355   allowed JSON value. If your action produces JSON output, then the HTTP header "`Content-Type`" can
356   be omitted.
357   • The alias "web" is also support for the full annotation key of "web-export". If a manifest has an Action
358   that contains both a "web" and "web-export" key, the values supplied for the 'web' key takes
359   precedence.
360   • <boolean> is either 'true' | 'false' string values where 'false' is equivalent to 'no' and 'true' is
361   equivalent to 'yes'.
362   • If "code" and "function" are both specified, then it is considered an error.

363

364   **Web Actions**

365   OpenWhisk can turn any Action into a "web action" causing it to return HTTP content without
366   use of an API Gateway. Simply supply a supported "type" extension to indicate which content
367   type is to be returned and identified in the HTTP header (e.g., `.json`, `.html`, `.text` or `.http`).

368 Return values from the Action's function are used to construct the HTTP response. The
369 following response parameters are supported in the response object.

- `headers`: a JSON object where the keys are header-names and the values are string
371 values for those headers (default is no headers).
- `code`: a valid HTTP status code (default is 200 OK).
- `body`: a string which is either plain text or a base64 encoded string (for binary data).

374 ## Trigger entity

375 The Trigger entity schema contains the necessary information to describe the stream of events
376 that it represents. For more information, see the document "Creating Triggers and Rules".

377 ### Fields

Key Name	Required	Value Type	Default	Description
feed	no	`string`	N/A	The optional name of the Feed associated with the Trigger.
credential	no	`Credential`	N/A	The optional credential used to access the feed service.
inputs	no	`list` of `parameter`	N/A	The optional ordered list inputs to the feed.
events	no	`list of Event`	N/A	The optional list of valid Event schema the trigger supports. OpenWhisk would validate incoming Event data for conformance against any Event schema declared under this key. *Note: This feature is <u>not supported at this time</u>. This is viewed as a possible feature that may be implemented along with configurable options for handling of invalid events.*

378 ### Requirements

- The Trigger name (i.e., `<triggerName>` MUST be less than or equal to 256 characters.
- The Trigger entity schema includes all general Entity Schema fields in addition to any fields declared
381 above.

382 ### Notes

- The '`events`' key name is not supported at this time.
- The Trigger entity within the OpenWhisk programming model is considered outside the scope of the
385 Package (although there are discussions about changing this in the future). This means that Trigger
386 and API information will not be returned when using the OpenWhisk Package API:
    - `wsk package list <package name>`
- However, it may be obtained using the Trigger API:
    - `wsk trigger list -v`

390 ### Grammar

```
<triggerName>:
  <Entity schema>
  feed: <feed name>
```

```
      credential: <Credential>
      inputs:
        <list of parameter>
```

391   *Example*

```
 triggers:
   everyhour:
     feed: /whisk.system/alarms/alarm
```

392   **Rule entity**

393   The Rule entity schema contains the information necessary to associates one trigger with one
394   action, with every firing of the trigger causing the corresponding action to be invoked with the
395   trigger event as input. For more information see the document "Creating Triggers and Rules".

396   *Fields*

Key Name	Required	Value Type	Default	Description
trigger	yes	`string`	N/A	Required name of the Trigger the Rule applies to.
action	yes	`string`	N/A	Required name of the Action the Rule applies to.
rule	no	`regex`	true	The optional regular expression that determines if the Action is fired. Note: In this version of the specification, only the expression "`true`" is currently supported.

397   *Requirements*

398   • The Rule name (i.e., `<ruleName>`) MUST be less than or equal to 256 characters.
399   • The Rule entity schema includes all general Entity Schema fields in addition to any fields declared
400   above.

401   *Requirements*

402   • OpenWhisk only supports a value of '`true`' for the '`rule`' key's value at this time.

403   *Grammar*

```
 <ruleName>:
   description: <string>
   trigger: <string>
   action: <string>
   rule: <regex>
```

404   *Example*

```
 my_rule:
   description: Enable events for my Action
   trigger: my_trigger
   action: my_action
```

405 *Feed entity*

406 The OpenWhisk Feed entity schema contains the information necessary to describe a
407 configurable service (that may work with an existing network accessible service) to produce
408 events on its behalf thereby acting as an Event Source.
409
410 At this time, the Package Manifest simply provides the information to describe a Feed (service),
411 its Action, lifecycle operations (along with their parameters) and the associated service it works
412 with. In the future, we intend to allow more granular ability to manage Feeds directly using their
413 operations.

414 *Fields*

Key Name	Required	Value Type	Default	Description
location	no	string	N/A	The URL for the Feed service which can be used by the OpenWhisk platform for registration and configuration.
credential	no	string	N/A	Contains either: • A credential string. • The optional name of a credential (e.g., token) that must be used to access the Feed service. Note: this would be defined elsewhere, perhaps as an input parameter to the Package.
operations	no	list of operations	N/A	The list of operations (i.e., APIs) the Feed supports on the URL provided described, by default, using the OpenAPI (f.k.a. "Swagger") specification schema.
operation_type	no	openwhisk | openapi@<version>	openwhisk	The specification format for the operation definitions.
action	no	string	N/A	The optional name of the Action if this is a Feed Action, that is, the Feed service implementation is an OpenWhisk Action.

415 *Requirements*

416 • The Feed name (i.e., `<feedName>` MUST be less than or equal to 256 characters.
417 • The Feed entity schema includes all general Entity Schema fields in addition to any fields declared
418   above.
419 • If the `action` field is set, the corresponding Action definition and function (code) MUST be a valid
420   Feed Action.
421 • The location and credential SHOULD be supplied if the Feed is not a Feed action using a Deployment
422   File.
423 • Operation names in manifests MAY be lower or upper cased (e.g., "create" or "CREATE").

424 *Grammar*

```
<feedName>:
  location: <string>
  credential: <string>
  operations:
```

```
    <list of operations>
  action: <string>
```

Example

The following example shows the mandatory operations for Feed Actions.

```
my_feed:
  description: A simple event feed
  location: https://my.company.com/services/eventHub
  # Reference to a credential defined elsewhere in manifest
  credential: my_credential
  operations:
    # Note: operation names in manifests MAY be lower or upper cased.
    create | CREATE:
      inputs:
        <parameters>
    delete | DELETE:
      inputs:
        <parameters>
    pause | PAUSE:
      inputs:
        <parameters>
    unpause | UNPAUSE:
      inputs:
        <parameters>
    # Additional, optional operations
    ...
```

Discussion

For a description of types of Feeds and why they exist, please see:

• https://github.com/apache/incubator-openwhisk/blob/master/docs/feeds.md.

Feed Actions

OpenWhisk supports an open API, where any user can expose an event producer service as a
feed in a **package**. This section describes architectural and implementation options for providing
your own feed.

Feed actions and Lifecycle Operations

The *feed action* is a normal OpenWhisk *action*, but it should accept the following parameters:

• **lifecycleEvent**: one of 'CREATE', 'DELETE', 'PAUSE', or 'UNPAUSE'
• **triggerName**: the fully-qualified name of the trigger which contains events produced from this feed.
• **authKey**: the Basic auth. credentials of the OpenWhisk user who owns the trigger just mentioned

The feed action can also accept any other parameters it needs to manage the feed. For example,
the Cloudant changes feed action expects to receive parameters including *'dbname'*, *'username'*,
etc.

444 *Sequence entity*

445 Actions can be composed into sequences to, in effect, form a new Action. The Sequence entity
446 allows for a simple, convenient way to describe them in the Package Manifest.

447 *Fields*

Key Name	Required	Value Type	Default	Description
actions	yes	`list of Action`	N/A	• The required list of two or more actions
web	no	`<boolean> | yes | no | raw`	`false`	The optional flag (annotation) that makes the action accessible to REST calls *without* authentication. • For details on all Action annotations, see: https://github.com/apache/incubator-openwhisk/blob/master/docs/annotations.md

448 *Web*

449 The following values are recognized values for the 'web' key:

Value	Alias	Description
true	yes	A value of `true` (or `yes`) enables an Action to be accessible via REST interface without the need for credentials.
false	no	A value of `false` (or `no`) disables an Action as an accessible REST.
raw	N/A	Implies a value of `true` and indicates that the Web Action is able to consume "raw" contents within the body of an HTTP request. Specifically, this means that the `__ow_body` content is encoded in Base64 when the request `Content-Type` is binary.

450 *Requirements*

451 • The comma separated list of Actions on the actions key SHALL imply the order of the sequence (from
452 left, to right).
453 • There MUST be two (2) or more actions declared in the sequence.
454 • Annotations MAY be ignored by target consumers of the Manifest file as they are considered data
455 non-essential to the deployment of management of OpenWhisk entities themselves.
456 • <boolean> is either 'true' | 'false' string values where 'false' is equivalent to 'no' and 'true' is
457 equivalent to 'yes'.

458 *Notes*

459 • The sequences key exists for convenience; however, it is just one possible instance of a composition
460 of Actions. The composition entity is provided for not only describing sequences, but also for other
461 (future) compositions and additional information needed to compose them. For example, the
462 composition entity allows for more complex mappings of input and output parameters between
463 Actions.

464 *Grammar*

```
sequences:
  <sequence name>:
```

```
    <Entity schema>
    actions: <ordered list of action names>
  ...
```

465 *Example*

```
sequences:
  newbot:
    actions: oauth/login, newbot-setup, newbot-greeting
```

466 **API entity**

467 This entity allows manifests to link Actions to be made available as HTTP-based API endpoints
468 as supported by the API Gateway service of OpenWhisk.

469 This entity declaration is intended to provide grammar for the experimental API *(see*
470 *https://github.com/apache/incubator-openwhisk/blob/master/docs/apigateway.md* and shown
471 using a "book club" example:

472 *CLI Example*

```
$ wsk api create -n "Book Club" /club /books get getBooks
$ wsk api create /club /books post postBooks
$ wsk api create /club /books put putBooks
$ wsk api create /club /books delete deleteBooks
```

473 the above would translate to the following grammars in the pkg. spec. to a new-top level
474 keyname "apis" in the manifest:

475 *Grammar*

```
apis:
  <API name>:                    # descriptive name
    description: <string>        # optional, description
    <basepath>:                  # shared basepath
      <path>:
        <action name>: get | post | put | delete
        ...
      ...
```

476 *Note*

477 • There can be more than one set of named <path> actions under the same <basepath>.

478 *Example*

479 A somewhat simplified grammar is also supported that allows single-line definition of Actions
480 (names) along with their HTTP verbs.
481

```
apis:
  book-club:
```

```
club:
  books:
    getBooks: get
    postBooks: post
    putBooks: put
    deleteBooks: delete
  members:
    listMembers: get
```

482 *Requirements*

483   • The API entity's name (i.e., `<API Name>`) MUST be less than or equal to 256 characters.

484 *Notes*

485   • The API entity within the OpenWhisk programming model is considered outside the scope of the
486   Package. This means that API information will not be returned when using the OpenWhisk Package
487   API:
488   • `wsk package list <package name>`
489   • However, it may be obtained using the Trigger API:
490   • `wsk api list -v`

491 **Package entity**

492 The Package entity schema is used to define an OpenWhisk package within a manifest.

493 *Fields*

Key Name	Required	Value Type	Default	Description
version	yes	`version`	N/A	The required user-controlled version for the Package.
license	no	`string`	N/A	The required value that indicates the type of license the Package is governed by. The value is required to be a valid Linux-SPDX value. See https://spdx.org/licenses/.
credential	no	`string`	N/A	The optional Credential used for all entities within the Package. The value is either: Contains either: • A credential string. • The optional name of a credential (e.g., token) that is defined elsewhere.
dependencies	no	list of Dependency	N/A	The optional list of external OpenWhisk packages the manifest needs deployed before it can be deployed.
repositories	no	list of `Repository`	N/A	The optional list of external repositories that contain functions and other artifacts that can be found by tooling.
actions	no	list of `Action`	N/A	Optional list of OpenWhisk Action entity definitions.
sequences	no	list of Sequence	N/A	Optional list of OpenWhisk Sequence entity definitions.

Key Name	Required	Value Type	Default	Description
triggers	no	list of `Trigger`	N/A	Optional list of OpenWhisk Trigger entity definitions.
rules	no	list of `Rule`	N/A	Optional list of OpenWhisk Rule entity definitions.
feeds	no	list of `Feed`	N/A	Optional list of OpenWhisk Feed entity definitions.
apis	no	list of `API`	N/A	Optional list of API entity definitions.
compositions *(Not yet supported)*	no	list of `Composition`	N/A	Optional list of OpenWhisk Composition entity definitions.
public	no	boolean	false	Optional indicator to deploy the package as a "public" package (requiring no access credentials).

Requirements

- The Package `name` MUST be less than or equal to 256 characters.
- The Package entity schema includes all general Entity Schema fields in addition to any fields declared above.
- A valid Package `license` value MUST be one of the Linux SPDX license values; for example: `Apache-2.0` or `GPL-2.0+`, or the value '`unlicensed`'.
- Multiple (mixed) licenses MAY be described using using NPM SPDX license syntax.
- A valid Package entity MUST have one or more valid Actions defined.
- The key "default" is a reserved Package name that indicates that the entities (e.g., Actions, Rules, etc.) should be registered under the user's default namespace.

Notes

- Currently, the '`version`' value is not stored in Apache OpenWhisk, but there are plans to support it in the future.
- Currently, the '`license`' value is not stored in Apache OpenWhisk, but there are plans to support it in the future.
- The Trigger and API entities within the OpenWhisk programming model are considered outside the scope of the Package. This means that Trigger and API information will not be returned when using the OpenWhisk Package API:
  - `wsk package list <package name>`
- However, their information may be retrieved using respectively:
  - `wsk trigger list -v`
  - `wsk api list -v`

Grammar

```
<packageName>:
    <Entity schema>
    version: <version>
    license: <string>
    repositories: <list of Repository>
    actions: <list of Action>
```

```
    sequences: <list of Sequence>
    triggers: <list of Trigger>
    rules: <list of Rule>
    feeds: <list of Feed>
    apis: <list of API>
    compositions: <list of Composition> # Not yet supported
```

517   *Example*

```
my_whisk_package:
  description: A complete package for my awesome action to be deployed
  version: 1.2.0
  license: Apache-2.0
  actions:
    my_awsome_action:
      <Action schema>
  triggers:
    trigger_for_awesome_action:
      <Trigger schema>
  rules:
    rule_for_awesome_action>
      <Rule schema>
```

518   **Interpolation of values using Environment Variables**

519   *Dollar Notation ($) schema for values*

520   In a Manifest or Deployment file, certain values may be set from the local execution
521   environment by using dollar ($) notation to denote names of local environment variables which
522   supply value, or portions of values, to be inserted at execution time.

523   *Syntax*

```
<some_key>: $<local_environment_variable_name>
```

524   *Example*

```
...
  inputs:
    userName: $DEFAULT_USERNAME
```

525   *Requirements*

526   • Processors or tooling that encounter ($) Dollar notation and are unable to locate the value in the
527   execution environment SHOULD resolve the value to be the default value for the type (e.g., an empty
528   string ("") for type 'string').
529   • A value binding provided on the 'value' key takes precedence over a value binding on the 'default'
530   key.
531   • Multiple interpolations SHOULD be supported on parameter values.
532   • String literals MAY be declared using double-dollar notation ($$).
533

Notes

- Processors or tooling that encounter ($) Dollar notation for values should attempt to locate the corresponding named variables set into the local execution environment (e.g., where the tool was invoked) and assign its value to the named input parameter for the OpenWhisk entity.

- This specification does not currently consider using this notation for other than simple data types (i.e., we support this mechanism for values such as strings, integers, floats, etc.) at this time.

Using environment variables in a string concatenation

If you wish to use the value of an environment variable as part of a string parameter's value, wskdeploy supports a modified Dollar notation in conjunction with curly brackets to indicate a string concatenation.

Example

```
...
  inputs:
    company_email: ${MY_EMAIL_SHORTNAME}.middleearth.travel
```

Where

- if the value "MY_EMAIL_SHORTNAME" was set in the execution environment of wskdeploy to "frodo", the parameter 'company_email' would be set (bound) to "frodo.middleearth.travel".

Composition entity *(Not yet supported)*

The Composition entity schema contains information to declare compositions of OpenWhisk Actions. Currently, this includes Action Sequences where Actions can be composed of two or more existing Actions.

Fields

Key Name	Required	Value Type	Default	Description
type	no	string	sequence	The optional type of Action composition. *Note: currently only 'sequence' is supported.*
inputs	no	list of parameter	N/A	The optional list of parameters for the Action composition (e.g., Action Sequence).
outputs	no	list of parameter	N/A	The optional outputs from the Entity.
sequence	no	ordered list of Action (names)	N/A	The optional expression that describes the connections between the Actions that comprise the Action sequence composition.

Key Name	Required	Value Type	Default	Description
parameterMappings	no	TBD	N/A	The optional expression that describes the mappings of parameter (names and values) between the outputs of one Action to the inputs of another Action. Note: Currently, mappings are not supported and JSON objects are passed between each Action in a sequence. At this time, it is assumed that the Actions in a sequence are designed to work together with no output to input mappings being performed by the OpenWhisk platform.

Requirements

555   •   The Composition name (i.e., `<compositionName>` MUST be less than or equal to 256 characters.

556   •   The Composition entity schema includes all general Entity Schema fields in addition to any fields
557   declared above.

558   *Grammar*

```
<compositionName>:
  <Entity schema> # Common to all OpenWhisk Entities
  type: <string>
  inputs:
    <list of parameter>
  outputs:
    <list of parameter>
  sequence:
    actions: <ordered list of action names>
  parameterMappings:
    # TBD. This is a future use case.
```

559   *Example: multi-line sequence*

```
my_action_sequence:
  type: sequence
  sequence:
    actions: action_1, action_2, action_3
  inputs:
    simple_input_string: string
  outputs:
    annotated_output_string: string
```

560   **Extended Schema**

561   *Dependencies*

562   The dependencies section allows you to declare other OpenWhisk packages that your application
563   or project (manifest) are dependent on. A Dependency is used to declare these other packages
564   which deployment tools can use to automate installation of these pre-requisites.

Fields

Key Name	Required	Value Type	Default	Description
location	yes	`string`	N/A	The required location of the dependent package.
version	yes	`version`	N/A	The required version of the dependent package.
inputs	no	`list of parameter`	N/A	The optional Inputs to the dependent package.

566   *Requirements*

567   • No additional requirements.

568   *Notes*

569   • The <package_name> is a local alias for the actual package name as described in the referenced
570   package. The referenced package would have its own Manifest file that would include its actual
571   Package name (and the one that would be used by the wskdeploy tool to replace the local alias).
572   • The 'version' parameter is currently used to specify a branch in GitHub and defaults to "master", this
573   behavior may change in upcoming releases of the specification.
574   • The experimental key name 'name' is only valid when the deprecated 'package' keyword has been
575   used instead of the favored key 'packages'. If it is used within the 'packages' structure, it will cause a
576   warning and be ignored as it is redundant to the <packageName>.

577   *Grammar*

```
<package name>:
  <Entity schema>
  location: <GitHub URL> |
  version: 1.0.1
  inputs:
    <list of parameter>
```

578   *Example*

```
dependencies:
  status_update:
    location: github.com/myrepo/statusupdate
    version: 1.0
  database pkg:
    location: /whisk.system/couchdb
    inputs:
      dbname: MyAppsDB
```

579

580   **Repository**

581   A repository defines a named external repository which contains (Action) code or other artifacts
582   package processors can access during deployment.

Fields

Key Name	Required	Value Type	Default	Description
description	no	`string256`	N/A	Optional description for the Repository.
url	yes	`string`	N/A	Required URL for the Repository.
credential	no	`Credential`	N/A	Optional name of a Credential defined in the Package that can be used to access the Repository.

584

585 *Requirements*

586 • The Repository name (i.e., `<repositoryName>` MUST be less than or equal to 256 characters.
587 • Description string values SHALL be limited to 256 characters.

588 *Grammar*

589 *Single-line (no credential)*

```
<repositoryName>: <repository_address>
```

590 *Multi-line*

```
<repositoryName>:
  description: <string256>
  url: <string>
  credential: <Credential>
```

591 *Example*

```
my_code_repo:
  description: My project's code repository in GitHub
  url: https://github.com/openwhisk/openwhisk-package-rss
```

592

593 **Credential**

594 A Credential is used to define credentials used to access network accessible resources. Fields

Key Name	Required	Value Type	Default	Description
protocol	no	`string`	N/A	Optional protocol name used to indicate the authorization protocol to be used with the Credential's token and other values.
tokenType	yes	`string`	password	Required token type used to indicate the type (format) of the token string within the supported types allowed by the protocol.
token	yes	`string`	N/A	Required token used as a credential for authorization or access to a networked resource. *Note: This string would be base64 encoded where necessary by protocol.*
description	no	`string256`	N/A	Optional description for the Credential.

Key Name	Required	Value Type	Default	Description
keys	no	`map` of `string`	N/A	Optional list of protocol-specific keys or assertions.

595

Requirements

597   • The Credential name (i.e., `<credentialName>` MUST be less than or equal to 256 characters.
598   • Description string values SHALL be limited to 256 characters.

Valid protocol values

Protocol Value	Valid Token Type Values	Description
`plain`	N/A	Basic (plain text) username-password (no standard).
`http`	basic_auth	HTTP Basic Authentication Protocol.
`xauth`	X-Auth-Token	HTTP Extended Authentication Protocol (base-64 encoded Tokens).
`oauth`	bearer	Oauth 2.0 Protocol
`ssh`	identifier	SSH Keypair protocol (e.g., as used in OpenStack)

600

Grammar

```
Credential:
  type: Object
  properties:
    protocol:
      type: string
      required: false
    tokenType:
      type: string
      default: password
    token:
      type: string
    keys:
      type: map
      required: false
      entry_schema:
        type: string
    user:
      type: string
      required: false
```

602   *Notes*

603   • The use of transparent user names (IDs) or passwords are not considered best practice.

604 *Examples*

605 *Plain username-password (no standardized protocol)*

```
inputs:
  my_credential:
    type: Credential
      properties:
        protocol: http
        token_type: plain
        token: myusername:mypassword
```

606 *HTTP Basic access authentication*

```
inputs:
  my_credential:
    type: Credential
    description: Basic auth. where <username>:<password> are a single string
    properties:
      protocol: http
      token_type: basic_auth
      token: myusername:mypassword
```

607 *X-Auth-Token*

```
inputs:
  my_credential:
    type: Credential
    description: X-Auth-Token, encoded in Base64
    properties:
      protocol: xauth
      token_type: X-Auth-Token
      token: 604bbe45ac7143a79e14f3158df67091
```

608 *OAuth bearer token*

```
inputs:
  my_credential:
    type: Credential
    properties:
      protocol: oauth2
      token_type: bearer
      token: 8ao9nE2DEjr1zCsicWMpBC
```

609 *SSH Keypair*

```
inputs:
  my_ssh_keypair:
    type: Credential
    properties:
      protocol: ssh
      token_type: identifier
      # token is a reference (ID) to an existing keypair (already installed)
      token: <keypair_id>
```

610

Project Artifacts

The manifest and Deployment files are comprised of the following entities:

Package Manifest File

The Package Manifest file is the primary OpenWhisk Entity used to describe an OpenWhisk Package and all necessary **schema** and **file** information needed for deployment. It contains the Package entity schema described above.

Deployment File

The Deployment file is used in conjunction with a corresponding Package Manifest file to provide configuration information (e.g., input parameters, authorization credentials, etc.) needed to deploy, configure and run an OpenWhisk Package for a target Cloud environment.

Project

An optional, top-level key that describes a set of related Packages that together comprise a higher-order project (or application) that incorporates one or more packages with external services.

Fields

Key Name	Required	Value Type	Default	Description
version	no	version	N/A	The optional user-controlled version for the Application.
name	yes	string256	N/A	The optional name of the application. Note: This key is only valid in the singular 'package' grammar.
namespace	no	string	N/A	The optional namespace for the application (and default namespace for its packages where not specified).
credential	no	string	N/A	The optional credential for the application (and default credential for its packages where not specified).
package	maybe	package (singular)	N/A	The required package definition when the key name 'packages' (plural) is not present.
packages	maybe	list of package (plural)	N/A	The required list of one or more package definitions when the key name 'package' (singular) is not present.

Grammar (singular)

```
project:
  version: <version>
  name: <string256>
  namespace: <string>
  credential: <string>
```

```
package:
  <package definition>
```

Grammar (plural)

```
project:
  version: <version>
  name: <string256>
  namespace: <string>
  credential: <string>
  packages:
    <list of package definitions>
```

629 *Requirements*

630 • The keys under the `project` (or `application`) key (e.g., `name`, `namespace`, `credential` and
631 `packages`) are only used in a manifest or deployment file if the optional application key is used.
632 • Either the key name '`package`' (singular) or the key name '`packages`' (plural) MUST be provided but
633 not both.
634   o If the '`package`' key name is provided, its value must be a valid package definition.
635   o If the '`packages`' key name is provided, its value must be one or more valid package
636   definitions.

637 *Notes*

638 • Currently, the OpenWhisk platform does not recognize the Project (or Application) entity as part of
639 the programming model; it exists as a higher order grouping concept only in this specification.
640 Therefore, there is no data stored within OpenWhisk for the Application entity.
641 • The keyword '`package`' and its singular grammar for declaring packages MAY be deprecated in
642 future versions of the specification.
643 • The keyword '`application`' MAY be deprecated in future versions of the specification.

644 *Example using the "project" keyword*

```
project:
  name: greetings
  namespace: /mycompany/greetings/
  credential: 1234-5678-90abcdef-0000
  packages:
    helloworld:
      inputs:
        city: Boston
      actions:
        hello:
          inputs: # input bindings
            personName: Paul
    ...
```

645 *Example using the synonymous "application" keyword*

```
application:
```

```
name: greetings
namespace: /mycompany/greetings/
credential: 1234-5678-90abcdef-0000
packages:
  helloworld:
    inputs:
      city: Boston
    actions:
      hello:
        inputs: # input bindings
          personName: Paul
  ...
```

646   *Example Notes*

647   • A common use would be to associate a namespace (i.e., a target namespace binding) or credential to
648   an application and all included packages automatically inherit that namespace (if applied at that
649   level) unless otherwise provided (similar to style inheritance in CSS).
650   • The project (or application) name would be treated as metadata, perhaps stored in the annotations
651   for the contained entities.

Normative References

Tag	Description
RFC2119	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels* http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
YAML-1.2	YAML, Version 1.2, 3rd Edition, Patched at 2009-10-01, Oren Ben-Kiki, Clark Evans, Ingy döt Net http://www.yaml.org/spec/1.2/spec.html
YAML-TS-1.1	Timestamp Language-Independent Type for YAML Version 1.1, Working Draft 2005-01-18, http://yaml.org/type/timestamp.html
SemVer	A simple set of rules and requirements that dictate how version numbers are assigned and incremented http://semver.org/
OpenAPI-2.0	The OpenAPI (f.k.a. "Swagger") specification for defining REST APIs as JSON. https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md
Linux-SPDX	Linux Foundation, SPDX License list https://spdx.org/licenses/
NPM-SPDX-Syntax	Node Package Manager (NPM) SPDX License Expression Syntax https://www.npmjs.com/package/spdx

Non-normative References

Tag	Description
OpenWhisk-API	OpenWhisk REST API which is defined as an OpenAPI document. https://raw.githubusercontent.com/openwhisk/openwhisk/master/core/controller/src/main/resources/whiskswagger.json
GNU-units	Size-type units are based upon a subset of those defined by GNU at http://www.gnu.org/software/parted/manual/html_node/unit.html
RFC 6838	Mime Type definitions in compliance with RFC 6838.
RFC 7231	HTTP 1.1. status codes are described in compliance with RFC 7231.
IANA-Status-Codes	HTTP Status codes as defined in the IANA Status Code Registry.
JSON Schema Specification	The built-in parameter type "json" references this specification. http://json-schema.org/

Scenarios and Use cases

Usage Scenarios

User background

The following assumptions about the users referenced in the usage scenarios:

- Experienced developer; knows Java, Node, SQL, REST principles and basic DevOps processes; uses IDEs to develop code locally.
- Limited exposure to Serverless, but interested in trying new technologies that might improve productivity.

Scenario 1: Clone and Create

Deploy an OpenWhisk app (project, set of entities, package, ...) discovered on github. The developer...

1. discovers an interesting git repo containing an OpenWhisk app (project, set of entities, package, ...)
2. clones the repo to local disk.
3. He pushes (deploys) it into one of his OpenWhisk namespaces
4. He checks out the app's behavior using OpenWhisk CLI or OpenWhisk UI

Notes

- while this scenario allows to use the manifest file as a "black box" the manifest format can influence the user experience of a developer trying to read it and understand what it does

Scenario 2: Pushing Updates with versioning

Change a cloned repo that he previously pushed into one of his namespaces. The developer...

1. changes the local repo by editing code and adding and changing entity specifications using local tools (editors, IDEs, ...).
2. bumps version number for package.
3. pushes his updates into the namespace so that the existing entities are changed accordingly.

Scenario 3: Start New Repo with Manifest

Start a new OpenWhisk app (project, set of entities) from scratch. The developer...

1. code files for the actions (e.g. *action1.js, action2.js, action3.js)*
2. creates a LICENSE.*txt file*

688   3. Creates a **Manifest File** that specifies the set of OpenWhisk entities and their relations
689   (e.g. *manifest.yml)*.   It also references the LICENSE.txt file.
690   4. initializes and uploads the set of files as a new git repo.
691

692   *Notes:*

693   • Creating the initial manifest file should be supported by providing an empty template
694   with syntax examples and other helpful comments

Scenario 4: Export into Repository

696   Share an existing OpenWhisk app (project, set of entities) with others
697   so that they can deploy and change it for their purposes. The developer…

698   1. exports a defined set of entities (a whole namespace?) into a set of files that includes code
699   files, and generated manifest, LICENSE.txt and README files.
700   2. initializes and uploads the set of files as a new git repo.
701   Example: `git init` … etc.

Scenario 5: Discovery and Import from object store

703   Discover an OpenWhisk package (manifest) co-located with data in an Object storage service.

704   This package would include a description of the Actions, Triggers, Rules and Event Sources (or
705   Feeds) necessary to interact with data it is associated with directly from the Object storage
706   repository; thus allowing anyone with access to the data an immediate way to interact and use the
707   data via the OpenWhisk Serverless platform.

Scenario 6: Clean

709   The user has deployed entities in a namespace. He/she wants to delete all entities, regardless of
710   how they were deployed (wsk, wskdeploy, etc..), in order to start from a clean slate.

Scenario 7: Project Sync

Sync remote project from local

713   The user has already started a project (manifest) and deployed it. They have modified the
714   manifest by adding, removing or updating existing entities and wants to re-deploy the project.
715   The local addition, deletion or update of these affected entities should be reflected in the remote
716   OpenWhisk platform.

Sync local project from remote

718   The user has already deployed a project (manifest) and to a remote OpenWhisk platform. They
719   have modified (i.e., added, updated or deleted entities) in the remotely deployed project (perhaps
720   using the remote platforms UI or the command line interface (CLI). The remote addition,

721   deletion or update of these affected entities should be reflected in the remote OpenWhisk
722   platform.
723

Clean deployed (non-shared) entities

725   The user has already started a project (manifest) and deployed it in a shared namespace. They
726   want to clean the deployed entities from a given project, while leaving the entities belonging to
727   the other projects untouched.

Create (refresh) project from remote

729   The user has deployed entities in a namespace in an ad hoc manner (e.g. by using a UI or the wsk
730   command line interface or CLI). They want to create a local project (manifest) from the entities
731   already deployed. A tool/command should help him/her in accomplishing this task.

Add entities to project from local

733   The user has already started a project (manifest) and are locally modifying files to add and/or
734   remove OpenWhisk entities (e.g., actions). They want to include these files into the deployment
735   manifest. A tool/command could help him/her to do this automatically.

Scenario 8: Tool Chain Support (pre-processor / post-processor) "plugins"

737   Support tool chain pipelines for pre/post processing deploy/undeploy commands. Also need to
738   consider Inputs/Outputs (parameters) these "tools" may need for configuration.

Guided examples

740 This packaging specification grammar places an emphasis on simplicity for the casual developer
741 who may wish to hand-code a Manifest File; however, it also provides a robust optional schema
742 that can be advantaged when integrating with larger application projects using design and
743 development tooling such as IDEs.
744
745 This guide will use examples to incrementally show how to use the OpenWhisk Packaging
746 Specification to author increasingly more interesting Package Manifest and Deployment files
747 taking full advantage of the specification's schema.
748
749 *Please note that the Apache 'wskdeploy' utility will be used to demonstrate output results.*

Package Examples

Example 1: Minimal valid Package Manifest

752 This use case shows a minimal valid package manifest file.
753
754 including:
755   • shows how to declare a Package named 'hello_world_package'.

Manifest Files

757 *Example 1: Minimum valid Package manifest file*

```
package:
  name: hello_world_package
  version: 1.0
  license: Apache-2.0
```

Notes

759   • Currently, the 'version' and 'license' key values are not stored in Apache OpenWhisk, but there are
760   plans to support it in the future.

Actions Examples

Example 1: The "Hello world" Action

763 As with most language introductions, in this first example we encode a simple "hello world"
764 action, written in JavaScript, using an OpenWhisk Package Manifest YAML file.
765
766 It shows how to:
767   • declare a single Action named 'hello_world' within the 'hello_world_package' Package.
768   • associate the JavaScript function's source code, stored in the file 'src/hello.js', to the
769   'hello_world' Action.

770 *Manifest File*

771 *Example: "Hello world" using a NodeJS (JavaScript) action*

```yaml
package:
  name: hello_world_package
  version: 1.0
  license: Apache-2.0
  actions:
    hello_world:
      function: src/hello.js
```

772

773 where "`hello.js`", within the package-relative subdirectory named 'src', contains the following
774 JavaScript code:

```javascript
function main(params) {
    msg = "Hello, " + params.name + " from " + params.place;
    return { greeting:  msg };
}
```

775 *Deploying*

```
$ ./wskdeploy -m docs/examples/manifest_hello_world.yaml
```

776 *Invoking*

```
$ wsk action invoke hello_world_package/hello_world --blocking
```

777 *Result*

778 The invocation should return an 'ok' with a response that includes this result:

```
"result": {
    "greeting": "Hello, undefined from undefined"
},
```

779 The output parameter 'greeting' contains "*undefined*" values for the '`name`' and '`place`' input
780 parameters as they were not provided in the manifest.

781 *Discussion*

782 This "hello world" example represents the minimum valid Manifest file which includes only the
783 required parts of the Package and Action descriptors.
784
785 In the above example,

- The Package and its Action were deployed to the user's default namespace using the 'package' name.
  - `/<default namespace>/hello_world_package/hello_world`
- The NodeJS default runtime (i.e., `runtime: nodejs`) was automatically selected based upon the '.js'
  extension on the Action function's source file '`hello.js`'.

Example 2: Adding fixed Input values to an Action

790

791 This example builds upon the previous "hello world" example and shows how fixed values can
792 be supplied to the input parameters of an Action.
793
794 It shows how to:

795 • declare input parameters on the action 'hello_world' using a single-line grammar.
796 • add 'name' and 'place' as input parameters with the fixed values "Sam" and "the Shire" respectively.

797 *Manifest File*

798 *Example: "Hello world" with fixed input values for 'name' and 'place'*

```
package:
  name: hello_world_package
  version: 1.0
  license: Apache-2.0
  actions:
    hello_world_fixed_parms:
      function: src/hello.js
      inputs:
        name: Sam
        place: the Shire
```

799 *Deployment*

```
$ ./wskdeploy -m docs/examples/manifest_hello_world_fixed_parms.yaml
```

800 *Invoking*

```
$ wsk action invoke hello_world_package/hello_world_fixed_parms --blocking
```

801 *Result*

802 The invocation should return an 'ok' with a response that includes this result:

```
"result": {
  "greeting": "Hello, Sam from the Shire"
},
```

803 *Discussion*

804 In this example:

805 • The value for the 'name' input parameter would be set to "Sam".
806 • The value for the 'place' input parameter would be set to "the Shire".
807 • The wskdeploy utility would infer that both 'name' and 'place' input parameters to be of type
808 'string'.

Example 3: "Hello world" with typed input and output parameters

809

810 This example shows the "Hello world" example with typed input and output Parameters.

811

812　It shows how to:

813　　• declare input and output parameters on the action 'hello_world' using a simple, single-line
814　　　grammar.
815　　• add two input parameters, 'name' and 'place', both of type 'string' to the 'hello_world' action.
816　　• add an 'integer' parameter, 'age', to the action.
817　　• add a 'float' parameter, 'height', to the action.
818　　• add two output parameters, 'greeting' and 'details', both of type 'string', to the action.

819　*Manifest File*

820　*Example: "Hello world" with typed input and output parameter declarations*

```yaml
package:
  name: hello_world_package
  ... # Package keys omitted for brevity
  actions:
    hello_world_typed_parms:
      function: src/hello_plus.js
      inputs:
        name: string
        place: string
        children: integer
        height: float
      outputs:
        greeting: string
        details: string
```

821　where the function 'hello_plus.js', within the package-relative subdirectory named 'src', is
822　updated to use the new parameters:

```javascript
function main(params) {
    msg = "Hello, " + params.name + " from " + params.place;
    family = "You have " + params.children + " children ";
    stats = "and are " + params.height + " m. tall.";
    return { greeting:  msg, details: family + stats };
}
```

823　*Deployment*

```
$ ./wskdeploy -m docs/examples/manifest_hello_world_typed_parms.yaml
```

824　*Invoking*

```
$ wsk action invoke hello_world_package/hello_world_typed_parms --blocking
```

825　*Result*

826　The invocation should return an 'ok' with a response that includes this result:

```
"result": {
```

```
    "details": "You have 0 children and are 0 m. tall.",
    "greeting": "Hello,  from "
},
```

827 *Discussion*

828 In this example:

829 • The default value for the `'string'` type is the empty string (i.e., `""`); it was assigned to the `'name'` and
830 `'place'` input parameters.
831 • The default value for the `'integer'` type is zero (0); it was assigned to the `'age'` input parameter.
832 • The default value for the `'float'` type is zero (0.0f); it was assigned to the `'height'` input parameter.

833 ## Example 4: "Hello world" with advanced parameters

834 This example builds on the previous '"Hello world" with typed input and output parameters'
835 example with more robust input and output parameter declarations by using a multi-line format
836 for declaration.
837
838 This example:
839 • shows how to declare input and output parameters on the action 'hello_world' using a multi-line
840 grammar.

841 *Manifest file*

842 If we want to do more than declare the type (i.e., 'string', 'integer', 'float', etc.) of the input
843 parameter, we can use specifications the multi-line grammar for Parameters.

844 *Example: input and output parameters with advanced fields*

```yaml
package:
  name: hello_world_package
  ... # Package keys omitted for brevity
  actions:
    hello_world_advanced_parms:
      function: src/hello.js
      inputs:
        name:
          type: string
          description: name of person
          default: unknown person
        place:
          type: string
          description: location of person
          value: the Shire
        children:
          type: integer
          description: Number of children
          default: 0
        height:
          type: float
          description: height in meters
          default: 0.0
```

```
        outputs:
          greeting:
            type: string
            description: greeting string
          details:
            type: string
            description: detailed information about the person
```

845 *Deployment*

```
$ ./wskdeploy -m docs/examples/manifest_hello_world_advanced_parms.yaml
```

846 *Invoking*

```
$ wsk action invoke hello_world_package/hello_world_advanced_parms --
blocking
```

847 Invoking the action would result in the following response:

```
"result":
  "details": "You have 0 children and are 0 m. tall.",
  "greeting": "Hello, unknown person from the Shire"
},
```

848 *Discussion*

849 • Describing the input and output parameter types, descriptions, defaults and other data:
850   o enables tooling to validate values users may input and prompt for missing values using the
851   descriptions provided.
852   o allows verification that outputs of an Action are compatible with the expected inputs of another
853   Action so that they can be composed in a sequence.
854 • The 'name' input parameter was assigned the 'default' key's value "unknown person".
855 • The 'place' input parameter was assigned the 'value' key's value "the Shire".

Example 5: Adding a Trigger and Rule to "hello world"

857 This example will demonstrate how to define a Trigger that is compatible with the basic
858 'hello_world' Action and associate it using a Rule.

859 *Manifest File*

860 *Example: "Hello world" Action with a compatible Trigger and Rule*

```
package:
  name: hello_world_package
  ... # Package keys omitted for brevity
  actions:
    hello_world_triggerrule:
      function: src/hello_plus.js
      inputs:
        name: string
        place: string
```

```
          children: integer
          height: float
        outputs:
          greeting: string
          details: string

    triggers:
      meetPerson:
        inputs:
          name: Sam
          place: the Shire
          children: 13
          height: 1.2

    rules:
      myPersonRule:
        trigger: meetPerson
        action: hello_world_triggerrule
```

861 *Deployment*

862 without the Deployment file:

```
$ wskdeploy -m docs/examples/manifest_hello_world_triggerrule.yaml
```

863 *Invoking*

864 First, let's try *"invoking"* the 'hello_world_triggerrule' Action directly without the Trigger.

```
$ wsk action invoke hello_world_package/hello_world_triggerrule --blocking
```

865 Invoking the action would result in the following response:

```
"result": {
  "details": "You have 0 children and are 0 m. tall.",
  "greeting": "Hello,  from "
},
```

866 As you can see, the results verify that the default values (i.e., empty strings and zeros) for the input
867 parameters on the 'hello_world_triggerrule' Action were used to compose the 'greeting' and
868 'details' output parameters. This result is expected since we did not bind any values or provide
869 any defaults when we defined the 'hello_world_triggerrule' Action in the manifest file.

870 *Triggering*

871 Instead of invoking the Action, here try *"firing"* the 'meetPerson' Trigger:

```
$ wsk trigger fire meetPerson
```

872 *Result*

873 which results in an Activation ID:

```
ok: triggered /_/meetPerson with id a8e9246777a7499b85c4790280318404
```

874 The 'meetPerson' Trigger is associated with 'hello_world_triggerrule' Action the via the
875 'meetPersonRule' Rule. We can verify that firing the Trigger indeed cause the Rule to be activated
876 which in turn causes the Action to be invoked:

```
$ wsk activation list

d03ee729428d4f31bd7f61d8d3ecc043 hello_world_triggerrule
3e10a54cb6914b37a8abcab53596dcc9 meetPersonRule
5ff4804336254bfba045ceaa1eeb4182 meetPerson
```

877 we can then use the 'hello_world_triggerrule' Action's Activation ID to see the result:

```
$ wsk activation get d03ee729428d4f31bd7f61d8d3ecc043
```

878 to view the actual results from the action:

```
"result": {
    "details": "You have 13 children and are 1.2 m. tall.",
    "greeting": "Hello, Sam from the Shire"
}
```

879 which verifies that the parameters bindings of the values (i.e., "Sam" (name), "the Shire" (place),
880 '13' (age) and '1.2' (height)) on the Trigger were passed to the Action's corresponding input
881 parameters correctly.

882 *Discussion*

883 • Firing the 'meetPerson' Trigger correctly causes a series of non-blocking "*activations*" of the associated
884   'meetPersonRule' Rule and subsequently the 'hello_world_triggerrule' Action.
885 • The Trigger's parameter bindings were correctly passed to the corresponding input parameters on the
886   'hello_world_triggerrule' Action when "firing" the Trigger.

887 ## Example 6: Using a Deployment file to bind Trigger parameters

888 This example builds on the previous Trigger-Rule example and will demonstrate how to use a
889 Deployment File to bind values for a Trigger's input parameters when applied against a
890 compatible Manifest File

891 *Manifest File*

892 Let's use a variant of the Manifest file from the previous example; however, we will leave the
893 parameters on the 'meetPerson' Trigger unbound and having only Type declarations for
894 each.

895 *Example: "Hello world" Action, Trigger and Rule with no Parameter bindings*

```
package:
  name: hello_world_package
  ... # Package keys omitted for brevity
  actions:
    hello_world_triggerrule:
      function: src/hello_plus.js
      runtime: nodejs
      inputs:
```

```
        name: string
        place: string
        children: integer
        height: float
      outputs:
        greeting: string
        details: string

  triggers:
    meetPerson:
      inputs:
        name: string
        place: string
        children: integer
        height: float

  rules:
    meetPersonRule:
      trigger: meetPerson
      action: hello_world_triggerrule
```

896   **Deployment File**

897   Let's create a Deployment file that is designed to be applied to the Manifest file (above) which will
898   contain the parameter bindings (i.e., the values) for the `'meetPerson'` Trigger.

899   *Example: Deployment file that binds parameters to the 'meetPerson' Trigger*

```
application:
  package:
    hello_world_package:
      triggers:
        meetPerson:
          inputs:
            name: Elrond
            place: Rivendell
            children: 3
            height: 1.88
```

900
901   As you can see, the package name `'hello_world_package'` and the trigger name `'meetPerson'` both
902   match the names in the corresponding Manifest file.
903

904 *Deploying*

905 Provide the Manifest file and the Deployment file to the wskdeploy utility:

```
$ wskdeploy -m docs/examples/manifest_hello_world_triggerrule_unbound.yaml
-d docs/examples/deployment_hello_world_triggerrule_bindings.yaml
```

906 *Triggering*

907 Fire the 'meetPerson' Trigger:

```
$ wsk trigger fire meetPerson
```

908 *Result*

909 Find the activation ID for the "hello_world_triggerrule' Action that firing the Trigger initiated and
910 get the results from the activation record:

```
$ wsk activation list

3a7c92468b4e4170bc92468b4eb170f1 hello_world_triggerrule
afb2c02bb686484cb2c02bb686084cab meetPersonRule
9dc9324c601a4ebf89324c601a1ebf4b meetPerson

$ wsk activation get 3a7c92468b4e4170bc92468b4eb170f1

"result": {
   "details": "You have 3 children and are 1.88 m. tall.",
   "greeting": "Hello, Elrond from Rivendell"
}
```

911 *Discussion*

912 • The 'hello_world_triggerrule' Action and the 'meetPerson' Trigger in the Manifest file both had
913   input parameter declarations that had no values assigned to them (only Types).
914 • The matching 'meetPerson' Trigger in the Deployment file had values bound its parameters.
915 • The wskdeploy utility applied the parameter values (after checking for Type compatibility) from the
916   Deployment file to the matching (by name) parameters within the Manifest file.

917 ## Github feed

918 This example will install a feed to fire a trigger when there is activity in a specified GitHub
919 repository.

920 *Manifest File*

```
git_webhook:
  version: 1.0
  license: Apache-2.0
  feeds:
    webhook_feed:
      version: 1.0
      function: github/webhook.js
```

```
        runtime: nodejs@6
        inputs:
          username:
            type: string
            description: github username
          repository:
            type: string
            description: url of github repository
          accessToken:
            type: string
            description: GitHub personal access token
          events:
            type: string
            description: the github event type

    triggers:
      webhook_trigger:
        action: webhook_feed
```

921 *Deployment File*

```
packages:
  git_webhook:
    triggers:
      webhook_trigger:
        inputs:
          username: daisy
          repository: https://github.com/openwhisk/wsktool.git
          accessToken:
          events:push
```

922

Advanced examples

923

Github feed advanced

924

925 This use case uses the Github feed to create a trigger. When there is any push event, it will send a
926 notification email.

927 *Manifest File*

```
git_webhook:
  version: 1.0
  license: Apache-2.0
  action:
    emailNotifier:
      version: 1.0
      function: src/sendemail.js
      runtime: nodejs
      inputs:
        email: string
        title: string
  rules:
```

```
    githubNotifier:
      trigger: webhook_trigger
      action: emailNotifier
```

928   *Deployment File*

```
packages:
  git_webhook:
    feeds:
      webhook_feed:
        inputs:
          email: daisy@company.com
          title: Github Push Notification
```

929

RSS Package

931   The RSS package provides RSS/ATOM feeds which can receive events when a new feed item is
932   available. It also defines a trigger to listen to a specific RSS feed. It describes the OpenWhisk
933   package reposited here:
934   https://github.com/openwhisk/openwhisk-package-rss.
935
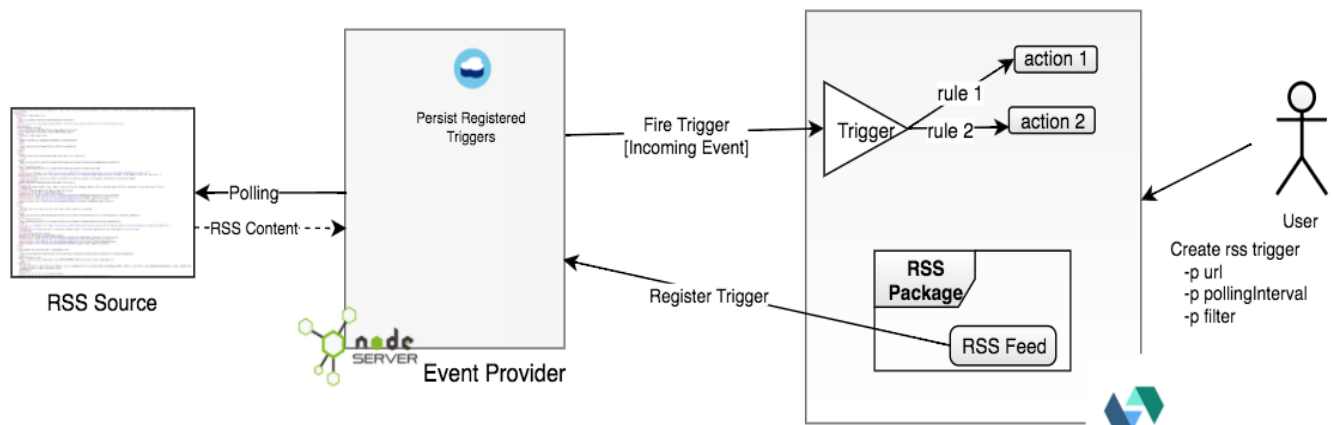
936

937   *Manifest File*

938   *with inline values (no Deployment File)*

939   This example makes use of in-line "values" where the developer does not intend to use a separate
940   Deployment file:

```
rss:
  version: 1.0
  license: Apache-2
  description: RSS Feed package
  inputs:
    provider_endpoint:
      value: http://localhost:8080/rss
      type: string
```

```
         description: Feed provider endpoint

    feeds:
      rss_feed:
        version: 1.0
        function: feeds/feed.js
        runtime: nodejs@6
        inputs:
          url:
            type: string
            description: url to RSS feed
            value: http://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml
          pollingInterval:
            type: string
            description: Interval at which polling is performed
            value: 2h
          filter:
            type: string
            description: Comma separated list of keywords to filter on

    triggers:
      rss_trigger:
        action: rss_feed
```

941

Deployment File

943 Alternatively, a Deployment File could have provided the same values (bindings) in this way:

```
packages:
  rss:
    inputs:
      provider_endpoint: http://localhost:8080/rss

    feeds:
      rss_feed:
        inputs:
          url: http://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml
          pollingInterval: 2h
```

944
945 Using such a deployment file, allows for more flexibility and the resulting Manifest file would
946 not have needed any '`value`' fields.

Polygon Tracking

948 This use case describes a microservice composition using Cloudant and a Push Notification
949 service to enable location tracking for a mobile application. The composition uses Cloudant to
950 store polygons that describe regions of interests, and the latest known location of a mobile user.
951 When either the polygon set or location set gets updated, we use the Cloudant Geo capabilities to
952 quickly determine if the new item satisfies a geo query like "is covered by" or "is contained in".
953 If so, a push notification is sent to the user.

954   *Manifest File:*

```
application:
  name: PolygonTracking
  namespace: polytrack

  packages:
    polytrack:

      triggers:
        pointUpdate:
          <feed>

        polygonUpdate:
          <feed>

      actions:
        superpush:
          inputs:
            appId: string
            appSecret: string

        pointGeoQuery:
          inputs:
            username: string
            password: string
            host: string
            dbName: string
            ddoc: string
            iName: string
            relation: string
          outputs:
            cloudantResp: json

        createPushParamsFromPointUpdate:
          <mapper>

        polygonGeoQuery:
          inputs:
            username: string
            password: string
            host: string
            dbName: string
            ddoc: string
            iName: string
            relation: string
          outputs:
            cloudantResp: json

        createPushParamsFromPolygonUpdate:
          <mapper>

      Rules:
        whenPointUpdate:
          trigger:
```

```
        pointUpdate
      action:
        handlePointUpdate
    whenPointUpdate:
      trigger:
        polygonUpdate
      action:
        handlePolygonUpdate


  Composition:
    handlePolygonUpdate:
      sequence:
createGeoQueryFromPolygonUpdate,polygonGeoQuery,createPushParamsFromPolygo
nUpdate,superpush
```

955 *Deployment File:*

```
application:
  name: PolygonTracking
  namespace: polytrack

  packages:

    myCloudant:
      <bind to Cloudant at whisk.system/Cloudant>

    polytrack:
      credential: ABDCF
      inputs:
        PUSHAPPID=12345
        PUSHAPPSECRET=987654
        COVEREDBY='covered_by'
        COVERS='covers'
        DESIGNDOC='geodd'
        GEOIDX='geoidx'
        CLOUDANT_username=myname
        CLOUDANT_password=mypassword
        CLOUDANT_host=myhost.cloudant.com
        POLYDB=weatherpolygons
        USERLOCDB=userlocation

      triggers:
        pointUpdate:
          <feed>
          inputs:
            dbname: $USERLOCALDB
            includeDoc: true
        polygonUpdate:
          <feed>
```

```
        inputs:
           dbname: $USERLOCDB
           includeDoc: true

    actions:
      superpush:
        inputs:
           appId: $PUSHAPPID
           appSecret: $PUSHAPPSECRET
      pointGeoQuery:
        inputs:
           designDoc: $DESIGNDOC
           indexName: $GEOIDX
           relation: $COVEREDBY
           username: $CLOUDANT_username
           password: $CLOUDANT_password
           host: $CLOUDANT_host
           dbName: $POLYDB
      polygonGeoQuery:
        inputs:
           designDoc: $DESIGNDOC
           indexName: $GEOIDX
           relation: $COVERS
           username: $CLOUDANT_username
           password: $CLOUDANT_password
           host: $CLOUDANT_host
           dbName: $POLYDB
```
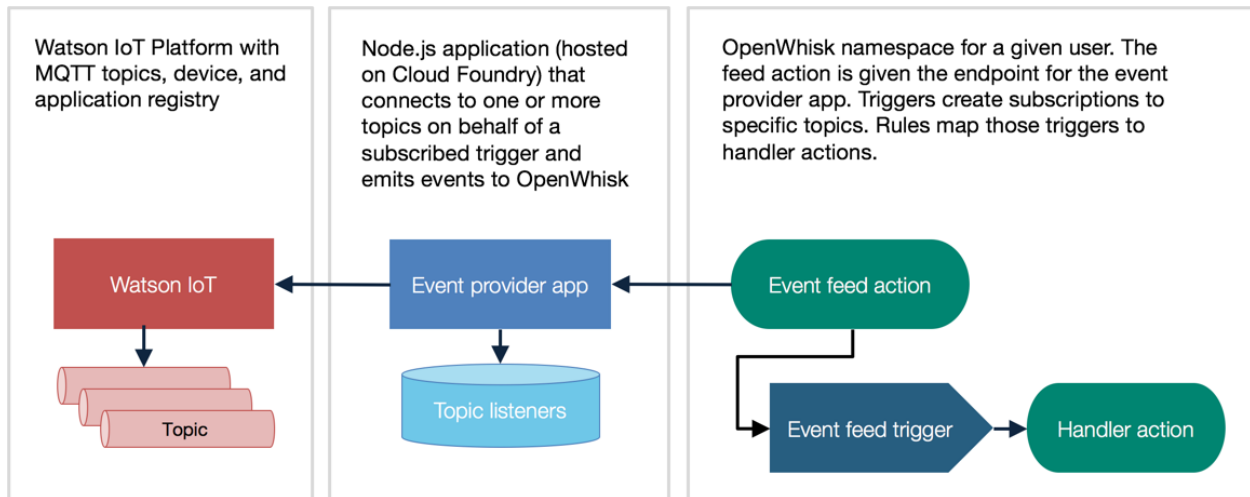
956

MQTT Package (tailored for Watson IoT)

The MQTT package that integrates with Watson IoT provides message topic feeds which can
receive events when a message is published. It also defines a trigger to listen to a specific MQTT
topic It describes the OpenWhisk package reposited here: https://github.com/krook/openwhisk-
package-mqtt-watson.

| Watson IoT Platform with MQTT topics, device, and application registry | Node.js application (hosted on Cloud Foundry) that connects to one or more topics on behalf of a subscribed trigger and emits events to OpenWhisk | OpenWhisk namespace for a given user. The feed action is given the endpoint for the event provider app. Triggers create subscriptions to specific topics. Rules map those triggers to handler actions. |

963
964

Manifest File

with inline values (no Deployment File)

This example makes use of in-line "values" where the developer does not intend to use a separate Deployment file:

```
mqtt_watson:
  version: 1.0
  license: Apache-2
  description: MQTT Feed package for Watson IoT
  inputs:
    provider_endpoint:
      value: http://localhost:8080/mqtt-watson
      type: string
      description: Feed provider endpoint

  feeds:
    mqtt_watson_feed:
      version: 1.0
      function: feeds/feed-action.js
      runtime: nodejs@6
      inputs:
        url:
          type: string
          description: URL to Watson IoT MQTT feed
          value: ssl://a-123xyz.messaging.internetofthings.ibmcloud.com:8883
        topic:
          type: string
          description: Topic subscription
          value: iot-2/type/+/id/+/evt/+/fmt/json
        apiKey:
          type: string
          description: Watson IoT API key
          value: a-123xyz
        apiToken:
          type: string
```

```
        description: Watson IoT API token
        value: +-derpbog
      client:
        type: string
        description: Application client id
        value: a:12e45g:mqttapp


  triggers:
    mqtt_watson_trigger:
      action: mqtt_watson_feed
```

969

Deployment File

971 Alternatively, a Deployment File could have provided the same values (bindings) in this way:

```
packages:
  mqtt_watson:
    inputs:
      provider_endpoint: http://localhost:8080/mqtt-watson

    feeds:
      mqtt_watson_feed:
        inputs:
          url: ssl://a-123xyz.messaging.internetofthings.ibmcloud.com:8883
          topic: iot-2/type/+/id/+/evt/+/fmt/json
          apiKey: a-123xyz
          apiToken: +-derpbog
          client: a:12e45g:mqttapp
```

972
973 Using such a deployment file, allows for more flexibility and the resulting Manifest file would
974 not have needed any '`value`' fields.
975

Check deposit processing with optical character recognition

977 This use case demonstrates an event-driven architecture that processes the deposit of checks to a
978 bank account using optical character recognition. It relies on Cloudant and SoftLayer Object
979 Storage. On premises, it could use CouchDB and OpenStack Swift. Other storage services could
980 include FileNet or Cleversafe. Tesseract provides the OCR library.
981
982 This application uses a set of actions and triggers linked by rules to process images that are
983 added to an object storage service. When new checks are detected a workflow downloads,
984 resizes, archives, and reads the checks then it invokes an external system to handle the
985 transaction.
986

987

988   *Manifest File:*

```
application:
  name: OpenChecks
  namespace: openchecks

  packages:
    openchecks:

    triggers:
      poll-for-incoming-checks:
        inputs:
          cron: string
          maxTriggers: integer

      check-ready-to-scan:
        inputs:
          dbname: string
          includDocs: boolean

      check-ready-for-deposit:
        inputs:
          dbname: string
          includDocs: boolean

    actions:
      find-new-checks:
        inputs:
          CLOUDANT_USER: string
          CLOUDANT_PASS: string
          SWIFT_USER_ID: string
          SWIFT_PASSWORD: string
          SWIFT_PROJECT_ID: string
```

```yaml
        SWIFT_REGION_NAME: string
        SWIFT_INCOMING_CONTAINER_NAME: string
        CURRENT_NAMESPACE: string

  save-check-images:
    inputs:
      CLOUDANT_USER: string
      CLOUDANT_PASS: string
      CLOUDANT_ARCHIVED_DATABASE: string
      CLOUDANT_AUDITED_DATABASE: string
      SWIFT_USER_ID: string
      SWIFT_PASSWORD: string
      SWIFT_PROJECT_ID: string
      SWIFT_REGION_NAME: string
      SWIFT_INCOMING_CONTAINER_NAME: string

  parse-check-data:
    inputs:
      CLOUDANT_USER: string
      CLOUDANT_PASS: string
      CLOUDANT_AUDITED_DATABASE: string
      CLOUDANT_PARSED_DATABASE: string
      CURRENT_NAMESPACE: string

  record-check-deposit:
    inputs:
      CLOUDANT_USER: string
      CLOUDANT_PASS: string
      CLOUDANT_PARSED_DATABASE: string
      CLOUDANT_PROCESSED_DATABASE: string
      CURRENT_NAMESPACE: string
      SENDGRID_API_KEY: string
      SENDGRID_FROM_ADDRESS: string

  parse-check-with-ocr:
    inputs:
      CLOUDANT_USER: string
      CLOUDANT_PASS: string
      CLOUDANT_AUDITED_DATABASE: string
      id: string
    outputs:
      result: JSON

rules:
  fetch-checks:
    trigger:
      poll-for-incoming-checks
    action:
      find-new-checks
  scan-checks:
    trigger:
      check-ready-to-scan
    action:
      parse-check-data
```

```
      deposit-checks:
        trigger:
          check-ready-for-deposit
        action:
          record-check-deposit
```

989 *Deployment File:*

```
application:
  name: OpenChecks
  namespace: openchecks

  packages:

    myCloudant:
      <bind to Cloudant at whisk.system/Cloudant>

    openchecks:
    credential: ABDCF
    inputs:
      XXX=YYY

    triggers:
      poll-for-incoming-checks:
        <feed>
        inputs:
          cron: */20 * * * * *
          maxTriggers: 90
```

```
    check-ready-to-scan:
      <feed>
      inputs:
        dbname: audit
        includeDoc: true
    check-ready-for-deposit:
      <feed>
      inputs:
        dbname: parsed
        includeDoc: true

  actions:
    find-new-checks:
      inputs:
        CLOUDANT_USER: 123abc
        CLOUDANT_PASS: 123abc
        SWIFT_USER_ID: 123abc
        SWIFT_PASSWORD: 123abc
        SWIFT_PROJECT_ID: 123abc
        SWIFT_REGION_NAME: northeast
        SWIFT_INCOMING_CONTAINER_NAME: incoming
        CURRENT_NAMESPACE: user_dev
    save-check-images:
      inputs:
        CLOUDANT_USER: 123abc
        CLOUDANT_PASS: 123abc
        CLOUDANT_ARCHIVED_DATABASE: archived
        CLOUDANT_AUDITED_DATABASE: audited
        SWIFT_USER_ID: 123abc
        SWIFT_PASSWORD: 123abc
        SWIFT_PROJECT_ID: 123abc
        SWIFT_REGION_NAME: northeast
        SWIFT_INCOMING_CONTAINER_NAME: container_name
    parse-check-data:
      inputs:
        CLOUDANT_USER: 123abc
        CLOUDANT_PASS: 123abc
        CLOUDANT_AUDITED_DATABASE: audited
        CLOUDANT_PARSED_DATABASE: parsed
        CURRENT_NAMESPACE: user_dev
    record-check-deposit:
      inputs:
        CLOUDANT_USER: 123abc
        CLOUDANT_PASS: 123abc
        CLOUDANT_PARSED_DATABASE: parsed
        CLOUDANT_PROCESSED_DATABASE: processed
        CURRENT_NAMESPACE: user_dev
        SENDGRID_API_KEY: 123abc
        SENDGRID_FROM_ADDRESS: user@example.org
```

```
parse-check-with-ocr:
  inputs:
    CLOUDANT_USER: 123abc
    CLOUDANT_PASS: 123abc
    CLOUDANT_AUDITED_DATABASE: audited
    id: 123abc
```

990

Event Sources

OpenWhisk is designed to work with any Event Source, either directly via published APIs from the Event Source's service or indirectly through Feed services that act as an Event Source on behalf of a service. This section documents some of these Event Sources and/or Feeds using this specification's schema.

Curated Feeds

The following Feeds are supported by the Apache OpenWhisk platform. They are considered "curated" since they are maintained alongside the Apache OpenWhisk open source code to guarantee compatibility. More information on curated feeds can be found here: https://github.com/apache/incubator-openwhisk/blob/master/docs/feeds.md.

Alarms

The `/whisk.system/alarms` package can be used to fire a trigger at a specified frequency. This is useful for setting up recurring jobs or tasks, such as invoking a system backup action every hour.

Package Manifest

The "alarms" Package Manifest would appear as follows:

```
# shared system package providing the alarms feed action
alarms:
  version: 1.0
  license: Apache-2
  description: Alarms and periodic utility

  actions:
    alarm:
      function: action/alarm.js
      description: Fire trigger when alarm occurs
      feed: true
      inputs:
        package_endpoint:
          type: string
          description: The alarm provider endpoint with port
        cron:
          type: string
          description: UNIX crontab syntax for firing trigger in
 Coordinated Universal Time (UTC).
          required: true
        trigger_payload:
          type: object
          description: The payload to pass to the Trigger, varies
          required: false
        maxTriggers:
          type: integer
          default: 1000
          required: false
```

```
feeds:
  location: TBD
  credential: TBD
  operations:
    CREATE:
      TBD
    DELETE:
      TBD
  action: alarm
```

1008
1009

Cloudant

1011 The `/whisk.system/cloudant` package enables you to work with a Cloudant database. It
1012 includes the following actions and feeds.

Package Manifest

1014 The "cloudant" Package Manifest would appear as follows:

TBD

Public Sources

1016 The following examples are Event Sources that can provide event data to OpenWhisk. We
1017 describe them here using this specification's schema.

GitHub WebHook

1019 Note: the GitHub WebHook is documented here: https://developer.github.com/webhooks/.
1020

1021 A sample description of the GitHub Event Source and its "create hook" API would appear as follows:

TBD

1022

Other Considerations

Tooling interaction

Using package manifest directly from GitHub

GitHub is an acknowledged as a popular repository for open source projects which may include OpenWhisk Packages along with code for Actions and Feeds. It is easily envisioned that the Package Manifest will commonly reference GitHub as a source for these artifacts; this specification will consider Github as being covered by the general Catalog use case.

Using package manifest in archive (e.g., ZIP) file

Compressed packaging, including popular ZIP tools, is a common occurrence for popular distribution of code which we envision will work well with OpenWhisk Packages; however, at this time, there is no formal description of its use or interaction. We leave this for future consideration.

Simplification of WebHook Integration

Using RESTify

One possible instance of a lightweight framework to build REST APIs in Nodejs to export WebHook functionality. See https://www.npmjs.com/package/restify RESTify (over Express) provides help in the areas of versioning, error handling (retry, abort) and content-negotiation. It also provides built in DTrace probes that identify application performance problems.

Enablement of Debugging for DevOps

Isolating and debugging "bad" Actions using (local) Docker

Simulate Inputs at time of an Action failure/error condition, isolate it and run it in a "debug" mode.

Considerations include, but are not limited to:

- Isolation on separate "debug" container
- Recreates "inputs" at time of failure
- Possibly recreates message queue state
- Provides additional stacktrace output
- Provides means to enable "debug" trace output
- Connectivity to "other" debug tooling

Using software debugging (LLDB) frameworks

This is a topic for future use cases and integrations. Specifically, working with LLDB frameworks will be considered. See http://lldb.llvm.org/.

Named Errors

1058

1059

1060 The following error types are supported by this specification:

1061

Name	Error Type	Notes
CommandError	ERROR_COMMAND_FAILED	Only used in wskdeploy.go, RunCommand(), Which in turn is called by: • Deploy • DeployWithCredentials • DeployProjectPathOnly • DeployManifestPathOnly • Undeploy • UndeplyWithCredentials • UndeployProjectPathOnly • UndeployManifestPathOnly which are all called directly by various integration tests (i.e., sec/tests/integration
ErrorManifestFileNotFound	ERROR_MANIFEST_FILE_NOT_FOUND	Unable to locate the Manifest file at location provided.
YAMLFileReadError	ERROR_YAML_FILE_READ_ERROR	Unable to read the general YAML file (but file found at path provided).
YAMLFormatError	ROR_YAML_FORMAT_ERROR	YAML parser detected an error.
YAMLParserError	ERROR_YAML_PARSER_ERROR	The YAML Parser detected an error with more detailed line information.
WhiskClientError	ERROR_WHISK_CLIENT_ERROR	Error detected using the OpenWhisk Client (CLI)
WhiskClientInvalidConfigError	ERROR_WHISK_CLIENT_INVALID_CONFIG	One or more configuration values is missing or invalid: • Auth key • API Host • Namespace
ParameterTypeMismatchError	ERROR_YAML_PARAMETER_TYPE_MISMATCH	

1062

Acknowledgements

Thanks to the following individuals who have contributed to the contents:

Castro, Paul
Desai, Priti
Guo, Ying Chun
Hou, Vincent
Kravchenko, Pavel
Krook, Daniel
Linnemeier, Micah
Liu, David
Mitchell, Nick
Ortelt, Thomas
Rutkowski, Matt
Santana, Carlos
Vennam, Belinda
Villard, Lionel