

OpenWhisk Package Specification

Version 0.8, Working Draft 02

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 specification is licensed under [The Apache License, Version 2.0](#).

Introduction

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://raw.githubusercontent.com/openwhisk/openwhisk/master/core/controller/src/main/resources/whiskswagger.json>
- *OpenAPI Specification when defining REST APIs and parameters:*
 - <https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md>

Revision History

Version	Date	Notes
0.8.1	2016-11-03	Initial public point draft, Working Draft 01

0.8.2	2016-12-12	Working Draft 02
-------	------------	------------------

Programming Model

OpenWhisk Entities

OpenWhisk uses the following entities to describe its programming model:

Action

A stateless, relatively short-running function invoked as an event handler.

Note: Typically, an Action's run time is on the order of seconds or even milliseconds, but usually well under the defaulted maximum of 5 minutes.

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).

Note: Rule conditional logic can be described relative to the event data and the target Action input parameters.

Note: In a pub-sub system, a trigger could be viewed as a message topic.

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 allows control operations (e.g., configure, start, stop, pause, resume, etc.) and configuration of Events coming from an Event Source.

Note: The terms Feed and Trigger are often used interchangeably conversationally (since they both represent a class of events); however, we make this distinction.

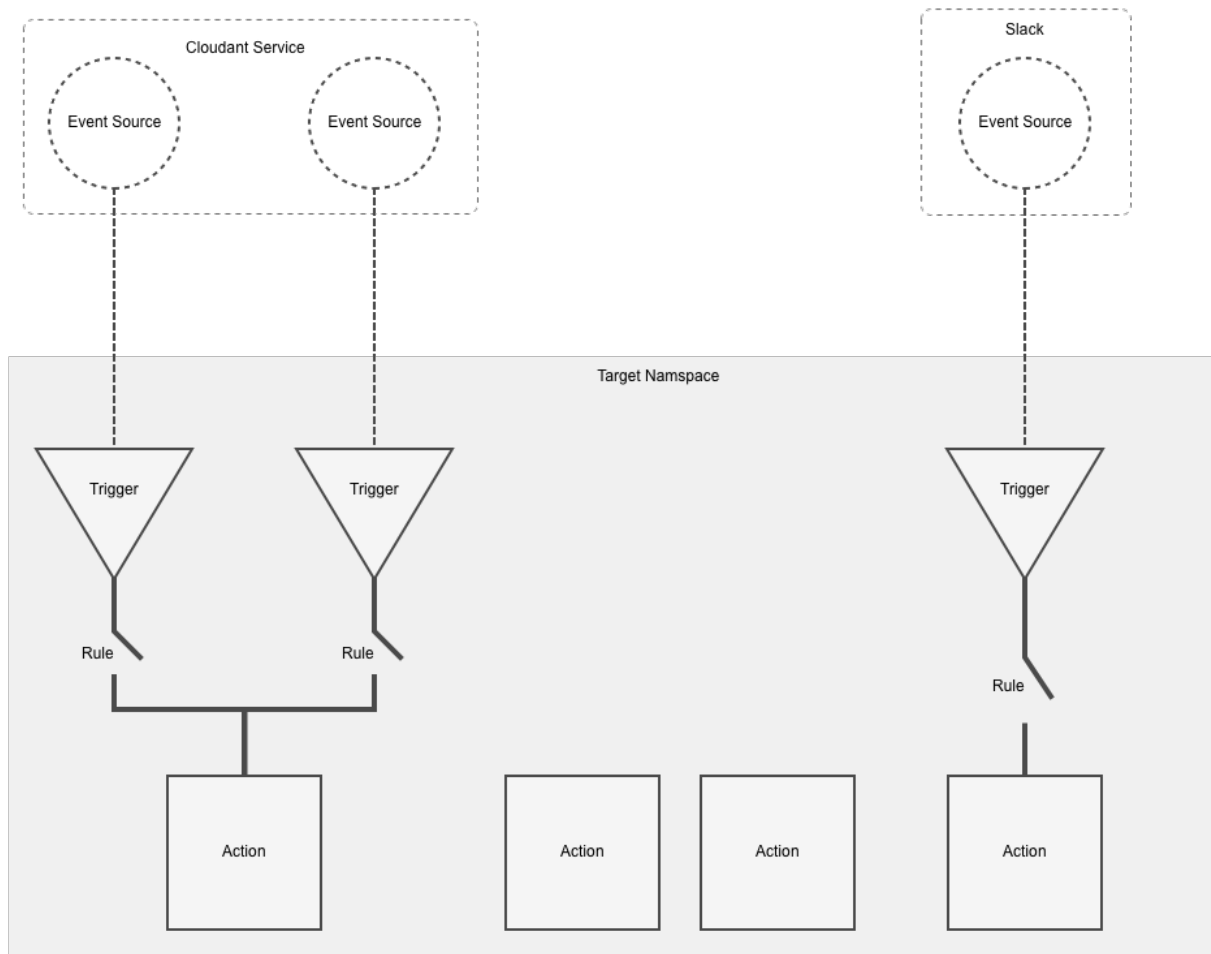
Package

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.

Note: Not all actions must belong to packages.

Note: Designed as a first-class entity within the OpenWhisk platform to be used by tooling such as catalogs (repositories), associated package managers, installers, etc.

Conceptual representation



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 Action 'A' in a declared sequence and provides it as input to the next Action 'B' in the sequence and so on until the sequence completes.

Note: This composition technique allows the reuse of existing action implementations treating them as “building blocks” for other Actions.

Namespacing

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

The fully qualified name of any entity has the format:

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

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

Note: The /whisk.system namespace is reserved for entities that are distributed with the OpenWhisk system.

Entity Names

The names of all entities, including actions, triggers, rules, packages, and namespaces, are a sequence of characters that follow the following format:

- The first character SHALL be an alphanumeric character, a digit, or an underscore.
- The subsequent characters MAY be alphanumeric, digits, spaces, or any of the following:
_, @, ., -
- The last character SHALL NOT be a space.

More precisely, a name must match the following regular expression (expressed with Java metacharacter syntax):

```
\A([\w]|[\w][\w@ .-]*[\w@.-])+\z
```

Cardinality

Trigger to Action

A single trigger can be configured to invoke multiple Actions.

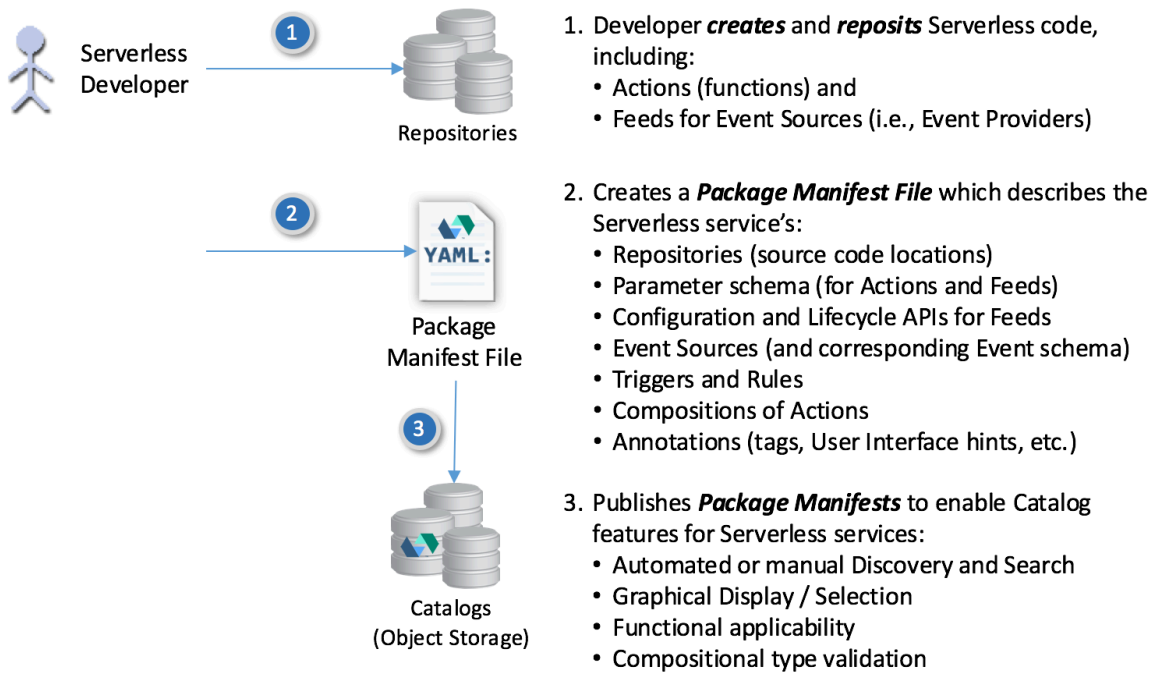
Package processing

This document defines two file artifacts that are used to deploy Packages to a target OpenWhisk platform; these include:

- **Package Manifest file:** Contains the Package definition along with any included Action, Trigger or Rule definitions that comprise the package. This file includes the schema of input and output data to each entity for validation purposes.
- **Deployment file:** Contains the values and bindings used configure a Package to a target OpenWhisk platform provider's environment and supply input parameter values for Packages, Actions and Triggers. This can include Namespace bindings, security and policy information.

Conceptual Package creation and publishing

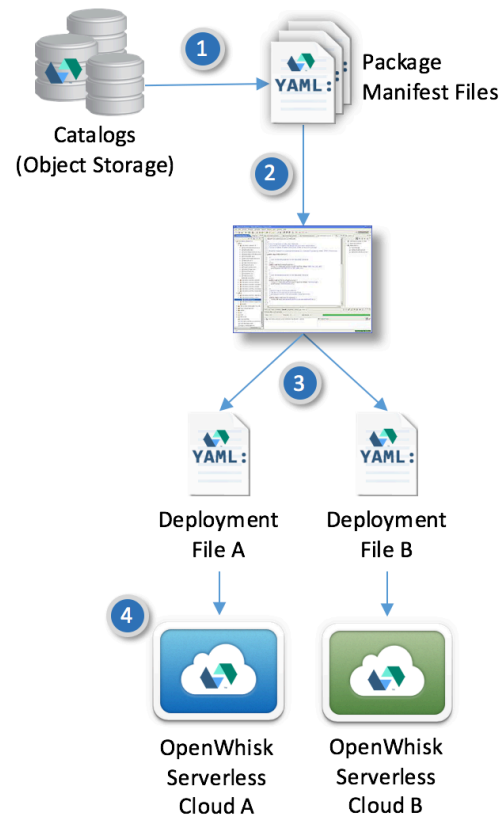
The following diagram illustrates how a developer would create OpenWhisk code artifacts and associate a Package Manifest file that describes them for deployment and reuse.



Conceptual tooling integration and deployment

The following diagram illustrates how Package manifests can be leveraged by developer tooling to 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)**.



Definitions

Activation

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

Repository

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

Project

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

Application

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

[Cloud] Service

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

Specification

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

YAML Types

Many of the types we use in this profile are *built-in* types from the [YAML 1.2 specification](#) (i.e., those identified by the “tag:yaml.org,2002” version tag).

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

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 (see YAML-TS-1.1)	ISO 8601 compatible.
null	tag:yaml.org,2002:null	Different meaning than an empty string, map, list, etc.
version	tag:maven.apache.org:version (see Maven version)	Typically found in modern tooling (i.e., “package@version” or “package:version” format).
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.

Requirements

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

OpenWhisk Types

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

Type Name	Description	Notes
scalar-unit	Convenience type for declaring common scalars that have an associated unit. For example, "10 msec.", "2 Gb", etc.)	See description below.
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.

scalar-unit types

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.

Grammar

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

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

- **scalar**: is a required scalar value (e.g., integer).
- **unit**: is a required unit value. The unit value MUST be type-compatible with the scalar value.

Requirements

- Whitespace: any number of spaces (including zero or none) SHALL be allowed between the scalar value and the unit value.
- 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.

177 *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)
TB	terabyte (1000000000000 bytes)

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

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

179 *Object type example*

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

```
inputs:
  person:
    type: object
    parameters:
```

182

183 **Entity Schema**

184 This section defines all the essential schema used to describe OpenWhisk packages within a
185 manifest.

186 *General Requirements*

- 187
- All field names in this specification SHALL be case sensitive.

188 *map schema*

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

190 *Single-line grammar*

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

191 Multi-line grammar

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

192 Examples

193 Single-line

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

194 Multi-line

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

195

196 Parameter schema

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

199 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.
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 .
properties	no	<list of Parameter schema>	N/A	The optional properties if the 'type' key has the value 'object'. Its value is a listing of Parameter schema from this specification.

200 *Grammar*

201 *Single-line*

```
<parameterName>: <any>
```

202 • Where <any> is the parameter’s ‘type’.

203 *Multi-line*

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

204 *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.

205

206 *Entity schema*

207 The entity schema contains fields that are common to all OpenWhisk entities.

208 *Fields*

Key Name	Required	Value Type	Default	Description
version	true	version	N/A	The required version for the Entity.
description	false	string256	N/A	The optional description for the Entity.
displayName	false	string16	N/A	This is the optional name that will be displayed on small form-factor devices.
annotations	false	TBD	N/A	The optional annotations for the Entity.
inputs	false	list of parameter	N/A	The optional ordered list inputs to the Entity.
outputs	false	list of parameter	N/A	The optional outputs from the Entity.

209 *Grammar*

```
version: <version>  
description: <string>  
annotations: <map of <string>>  
inputs: <list of <parameter>>
```

outputs: <list of <parameter>>

Requirements

- Version values for all entities SHALL be provided.
- Non-required fields MAY be stored as “annotations” within the OpenWhisk framework after they have been used for processing.
- Description string values SHALL be limited to 256 characters.
- DisplayName string values SHALL be limited to 16 characters.
- The lists of inputs and outputs MAY also be described as ordered lists (using the YAML ‘-’ dash convention).
- Annotations MAY be ignored by target consumers of the Manifest file as they are considered data non-essential to the deployment of management of OpenWhisk entities themselves.
 - Target consumers MAY preserve (persist) these values, but are not required to.
- For any OpenWhisk Entity, the maximum size of all Annotations SHALL be (TBD) characters.

Notes

- Input and output parameters are implemented as JSON Objects within the OpenWhisk framework.
- Several, non-normative Annotation keynames and allowed values for (principally for User Interface (UI) design) may be defined below for optional usage.

Action entity

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

Fields

Key Name	Required	Value Type	Default	Description
function	true	string	N/A	Required source location (path inclusive) of the Action code either <ul style="list-style-type: none">Relative to the Package manifest file.Relative to the specified Repository.
runtime	false	string	N/A	The required runtime name (and optional version) that the Action code requires for an execution environment. <i>Note: May be optional if tooling allowed to make assumptions about file extensions.</i>
limits	false	map of <limit keys and values>	N/A	Optional map of limit keys and their values. <i>See section “Valid limit keys” below for a listing of recognized keys and values.</i>
feed	false	boolean	false	Optional indicator that the Action supports the required parameters (and operations) to be run as a Feed Action.

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'
 - **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.

Notes

- 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.

Valid Runtime names

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

Each of these runtimes also include additional built-in packages (or libraries) that have been determined be useful for Actions surveyed and tested by the OpenWhisk platform. These packages may vary by OpenWhisk release; details on runtimes and their package conventions and default package (or libraries) here:

Runtime name	Supported versions	Default	Description
nodejs	nodejs@6, nodejs@5	nodejs@6	Node.js runtime.
java	java@8	java@8	Java language runtime.
python	python@2.7	python@2.7	Python language runtime.
swift	swift@3, swift@2	swift@3	Swift language runtime.
language:default	N/A	N/A	Permit the OpenWhisk platform to select the correct default language runtime.

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

Recognized File extensions

Although it is best practice to provide a runtime value when declaring an Action, it is not required. In those cases, that a runtime is not provided, the package tooling will attempt to derive the correct runtime based upon the the file extension for the Action's function (source code file).

265 The following file extensions are recognized and will be run on the latest version of
266 corresponding Runtime listed below:
267

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

268 *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). <i>Note: This value is not changeable via APIs at this time.</i>
userInvocationRate	integer	5000	See description	The maximum number of Action invocations allowed per user, per minute. <i>Note: This value is not changeable via APIs at this time.</i>
codeSize	scalar-unit.size	48 MB	See description	The maximum size of the Action code. <i>Note: This value is not changeable via APIs at this time.</i>
parameterSize	scalar-unit.size	1 MB	See description	The maximum size <i>Note: This value is not changeable via APIs at this time.</i>

269 *Notes*

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

273 *Grammar*

```
<actionName>:
  <Entity schema> # Common to all OpenWhisk Entities
  function: <string>
  runtime: <name>[@<[range of ]version>]
  limits:
    <list of limit key-values>
  feed: <boolean> # default: false
```

274 *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
```

275 *Trigger entity*

276 The Trigger entity schema contains the necessary information to describe the stream of events
277 that it represents. For more information see: “[Creating Triggers and Rules](#)”.

278 *Fields*

Key Name	Required	Value Type	Default	Description
feed	false	string		The optional name of the Feed associated with the Trigger.
events	false	list of Event		<div>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.</div> <div>Note: This feature is not supported at this time. This is viewed as a possible feature that may be implemented along with configurable options for handling of invalid events.</div>

279

280 *Requirements*

- 281
- 282
- 283
- 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 above.

284 *Grammar*

```
<triggerName>:
  <Entity schema> # Common to all OpenWhisk Entities
  feed: my_feed
```

285 *Rule entity*

286 The Rule entity schema contains the information necessary to associates one trigger with one
287 action, with every firing of the trigger causing the corresponding action to be invoked with the
288 trigger event as input. For more information see: “[Creating Triggers and Rules](#)”.

289 *Fields*

Key Name	Required	Value Type	Default	Description
trigger	true	string	N/A	Required name of the Trigger the Rule applies to.
action	true	string	N/A	Required name of the Action the Rule applies to.
rule	false	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.

290 *Requirements*

- 291
- The Rule name (i.e., <ruleName> MUST be less than or equal to 256 characters.
 - The Rule entity schema includes all general [Entity Schema](#) fields in addition to any fields declared above.
- 292
- 293

294 *Grammar*

```
<ruleName>:
  <Entity schema> # Common to all OpenWhisk Entities
  trigger: <string>
  action: <string>
  rule: <regex> # “true” supported at this time
```

295 *Example*

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

296 *Composition entity*

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

300 *Fields*

Key Name	Required	Value Type	Default	Description
type	false	string	sequence	The optional type of Action composition. <i>Note: currently only 'sequence' is supported.</i>
inputs	false	list of parameter	N/A	The optional list of parameters for the Action composition (e.g., Action Sequence).
sequence	false	ordered list of Action (names)	N/A	The optional expression that describes the connections between the Actions that comprise the Action sequence composition.
parameterMappings	false	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.

301 *Requirements*

- 302 • The Composition name (i.e., <compositionName> MUST be less than or equal to 256 characters.
- 303 • The Composition entity schema includes all general [Entity Schema](#) fields in addition to any fields
- 304 declared above.

305 *Grammar*

```

<compositionName>:
  <Entity schema> # Common to all OpenWhisk Entities
  type: <string>
  inputs: <list of parameters>
  sequence:
    - <order list of action names>
  parameterMappings:
    # TBD. This is a future use case.

```

306 *Example*

```

my_action_sequence:
  type: sequence
  sequence:
    - action_1
    - action_2
    - action_3

  inputs:
    simple_input_string: string
  outputs:
    annotated_output_string: string

```

Feed entity

The OpenWhisk Feed entity schema contains the information necessary to reference a configurable service that works with an existing network accessible service to produce events on its behalf thereby acting as an Event Source.

These are standalone services unto themselves which the OpenWhisk platform does not currently deploy and run from an OpenWhisk Package. At this time, the Package Manifest simply provides the information to access, configure and manage (via lifecycle operations) the Feed service that is already running elsewhere.

Fields

Key Name	Required	Value Type	Default	Description
location	false	string	N/A	The URL for the Feed service which can be used by the OpenWhisk platform or registration and configuration.
credential	false	Credential	N/A	The required 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	false	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	false	openwhisk openapi@<version>	openwhisk	The specification format for the operation definitions.
action	false	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.

Requirements

- The Feed name (i.e., <feedName> MUST be less than or equal to 256 characters.
- The Feed entity schema includes all general [Entity Schema](#) fields in addition to any fields declared above.
- If the action field is set, the corresponding Action definition and function (code) MUST be a valid Feed Action.
- The location and credential SHOULD be supplied if the Feed is not a Feed action from the Deployment File.

Grammar

```
<feedName>:
  <Entity schema> # Common to all OpenWhisk Entities
  location: <string>
  credential: <Credential>
  operations:
    <list of operations>
  action: <string>
```

326 *Example*

```
my_feed:
  description: A simple event feed
  location: https://my.company.com/services/eventHub
  credential: my_credential
  operations:
    # Note: This exmple shows the mandatory operations for Feed Actions.
    # 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
    ...
```

327 *Discussion*

328 For a description of types of Feeds and why they exist, please see
329 <https://github.com/openwhisk/openwhisk/blob/master/docs/feeds.md>.

330 *Feed Actions*

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

334 *Feed actions and Lifecycle Operations*

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

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

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

342 *Package entity*

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

344 *Fields*

Key Name	Required	Value Type	Default	Description
license	true	string	unlicensed	The required value that indicates the type of license the Package is governed by.
repositories	false	list of Repository	N/A	The optional list of external repositories that functions and other artifacts can be found by tooling.
actions	false	list of Action	N/A	Optional list of OpenWhisk Action entity definitions.
triggers	false	list of Trigger	N/A	Optional list of OpenWhisk Trigger entity definitions.
rules	false	list of Rule	N/A	Optional list of OpenWhisk Rule entity definitions.
feeds	false	List of Feed	N/A	Optional list of OpenWhisk Feed entity definitions.
compositions	false	List of Composition	N/A	Optional list of OpenWhisk Composition entity definitions.

345 *Requirements*

- 346
- The Package name MUST be less than or equal to 256 characters.
- 347
- The Package entity schema includes all general [Entity Schema](#) fields in addition to any fields declared above.
- 348
- 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'.
- 349
- Multiple (mixed) licenses MAY be described using using [NPM SPDX license syntax](#).
- 350
- A valid Package entity MUST have one or more valid Actions defined.
- 351
- 352

353 *Grammar*

```
<packageName>:
  <Entity schema> # Common to all OpenWhisk Entities
  license: <string>
  repositories: <list of repository>
  actions: <list of actions>
  triggers: <list of triggers>
  rules: <list of rules>
  feeds: <list of feeds>
  compositions: <list of compositions>
```

354 *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_awesome_action:
      <Action schema>
```

```

triggers:
  trigger_for_awesome_action:
    <Trigger schema>
rules:
  rule_for_awesome_action>
    <Rule schema>

```

Extended Schema

Repository

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

Fields

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

Requirements

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

Grammar

Single-line (no credential)

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

Multi-line

```

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

```

Example

```

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

```

Credential

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

Key Name	Required	Value Type	Default	Description
protocol	false	string	N/A	Optional protocol name used to indicate the authorization protocol to be used with the Credential's token and other values.
tokenType	true	string	password	Required token type used to indicate the type (format) of the token string within the supported types allowed by the protocol.
token	true	string	N/A	Required token used as a credential for authorization or access to a networked resource.
description	no	string256	N/A	Optional description for the Credential.
keys	false	map of string	N/A	Optional list of protocol-specific keys or assertions.

Requirements

- The Credential name (i.e., <credentialName> MUST be less than or equal to 256 characters.
- 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)

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
```

378 *Notes*

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

380 *Examples*

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

```
inputs:
  my_credential:
    type: Credential
    properties:
      user: my_username
      token: my_password
```

382 *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
      # Note: this would be base64 encoded before transmission by any impl.
      token: myusername:mypassword
```

383 *X-Auth-Token*

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

384 *OAuth bearer token*

```
inputs:
  my_credential:
    type: Credential
    properties:
      protocol: oauth2
      token_type: bearer
      # token encoded in Base64
      token: 8ao9nE2DEjr1zCsicWMpBC
```

385 *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>
```

386

387 **Package Artifacts**

388 *Package Manifest File*

389 The Package Manifest file is the primary OpenWhisk Entity used to describe an OpenWhisk
390 Package and all necessary **schema** and **file** information needed for deployment. It contains the
391 [Package entity schema](#) described above.

392 *Deployment File*

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

397 *Grammar*

```
# TBD optional tag identifying file type, ala. Bash

# collection, a description of source code
application: # TBD project, application or service "as a service"
  name: <string>
  namespace: <string>

packages:
  <package name>:
    function: <URL> # by reference (reusable packages)
    package_credential: <token> (TBD)
    namespace: <string> # deployment oriented, i.e., a targetnamespace
                      # scope package, inherited
    credential: <token> # use this credential for all actions, triggers,
etc.
    <credential fields if more than a simple, single value
token>
  inputs:
    <list of parameter name-value pairs>

  actions:
    <action name>:
```



```

    namespace: <string> # scope = this action only, override package
decl.
    credential: <token>
    inputs:
        <list of parameter name-value pairs>

    triggers:
        <trigger name>:
            namespace: # scope = this trigger only, override package decl.
            credential: <token>
            inputs:
                <list of parameter name-value pairs>

    feeds: # we call them "feeds" but these are feed actions
    ...

```

398

399 *Example*

```

packages:
  helloworld:
    namespace: /mycompany/greetings/
    credential: 1234-5678-90abcdef-0000
    inputs:
        city: Boston

    actions:
        hello:
            inputs: # input bindings
                personName: Paul
    ...

```

400

401 • *Note: A common use would be to associate a namespace (i.e., a target namespace binding) to a package*
 402 *and all included Actions and Triggers would automatically inherit that namespace (if applied at that*
 403 *level) unless otherwise provided (similar to style inheritance in CSS).*

404 **Normative References**

405

Tag	Description
RFC2119	S. Bradner, <i>Key words for use in RFCs to Indicate Requirement Levels</i> , 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
Maven-Version	The version type is defined with the Apache Maven project's policy draft: https://cwiki.apache.org/confluence/display/MAVEN/Version+number+policy
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

406 **Non-normative References**

407

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 .

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

- 439 3. Creates a **Manifest File** that specifies the set of OpenWhisk entities and their relations
440 (e.g. *manifest.yml*). It also references the LICENSE.txt file.
441 4. initializes and uploads the set of files as a new git repo.
442

443 *Notes:*

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

446 *Scenario 4: Export into Repository*

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

- 449 1. exports a defined set of entities (a whole namespace?) into a set of files that includes code
450 files, and generated manifest, LICENSE.txt and README files.
451 2. initializes and uploads the set of files as a new git repo.
452 Example: `git init ...` etc.

453 *Scenario 5: Discovery and Import from object store*

454 Discover an OpenWhisk package (manifest) co-located with data in an Object storage service.
455 This package would include a description of the Actions, Triggers, Rules and Event Sources (or
456 Feeds) necessary to interact with data it is associated with directly from the Object storage
457 repository; thus allowing anyone with access to the data an immediate way to interact and use the
458 data via the OpenWhisk Serverless platform.

459 **Schema Use Cases**

460 **“Hello world”**

461 “Hello world” is a simple use case which will accept a string parameter and return a “Hello
462 world” if the parameter is null or “Hello + <string>” if the string parameter is not null.

463 *Manifest Files*

464 *Example 1: No parameter descriptions*

```
helloworld:
  version: 1.0
  license: Apache-2.0
  actions:
    hello:
      version: 1.0
      function: src/hello/hello.js
      inputs:
        personName: string
```

```
    outputs:
      greeting_string: string
```

Defaults discussion

This specification places an emphasis on simplicity for the casual developer who may wish to hand-code a Manifest and/or Deployment File; however, it also provides a robust optional schema that can be advantaged when integrating with larger application projects using design and development tooling such as IDEs.

In the above example, please note the following intelligent defaults would be applied by a deployment tooling authored using this specification:

- The runtime (i.e., runtime: nodejs) is found from the .js extension on the Action function's filename.

Example 2: with parameter descriptions

```
helloworld:
  version: 1.0
  license: Apache-2.0
  actions:
    hello:
      version: 1.0
      function: src/hello/hello.js
      runtime: nodejs@6
      inputs:
        personName:
          type: string
          description: input person name
      outputs:
        greeting_string:
          type: string
          description: output greeting string
```

Basic Trigger and Rule

This use case will demonstrate how rules and triggers are configured in manifest and deployment file. It will use the previous helloworld action, create a rule to associate a trigger with it.

Manifest Files

```
helloRules:
  version: 1.0
  license: Apache-2.0
  actions:
    hello:
      version: 1.0
      function: src/hello/hello.js
      runtime: nodejs
      inputs:
        personName: string
      outputs:
```

```
    greeting_string: string

  triggers:
    locationUpdate:
      inputs:
        name: string
        place: string

  rules:
    myRule:
      trigger: locationUpdate
      action: hello
```

480 **Deployment Files**

```
packages:
  helloRules:
    triggers:
      locationUpdate:
        inputs:
          name: Donald
          place: Washington, D.C.
```

481

482 **Github feed**

483 This use case will install a feed to fire a trigger when there is activity in a specified GitHub
484 repository.

485 **Manifest Files**

```
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
```

486 **Deployment Files**

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

487

488 **Github feed advanced**

489 This use case use the Github feed to create a trigger. When there is any push event happened, it
490 will send a notification email.

491 **Manifest Files**

```
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
```

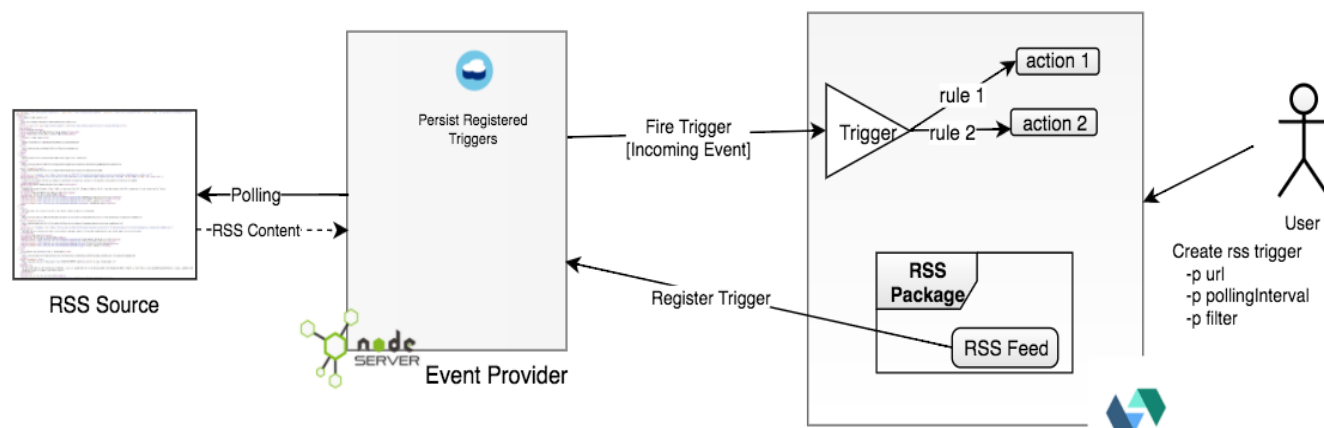
492 **Deployment Files**

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

RSS Package

The RSS package provides RSS/ATOM feeds which can receive events when a new feed item is available. It also defines a trigger to listen to a specific RSS feed. It describes the OpenWhisk package repositied here:

<https://github.com/openwhisk/openwhisk-package-rss>.



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:

```
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

```

506

507 *Deployment File*

508 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

```

509

510 Using such a deployment file, allows for more flexibility and the resulting Manifest file would
 511 not have needed any ‘value’ fields.

512 *Polygon Tracking*

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

519 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
  whenPolygonUpdate:
    trigger:
      polygonUpdate
    action:
      handlePolygonUpdate

Composition:
  handlePolygonUpdate:
```

```

sequence:
createGeoQueryFromPolygonUpdate,polygonGeoQuery,createPushParamsFromPolygo
nUpdate,superpush
  handlePolygonUpdate:
    sequence:
createGeoQueryFromPointUpdate,pointGeoQuery,createPushParamsFromPointUpdat
e,superpush

```

520 *Deployment Descriptor:*

521

```

application:
  name: PolygonTracking
  namespace: polytrack

packages:

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

  polytrack:
    credential: ABDCE
    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:

```

```

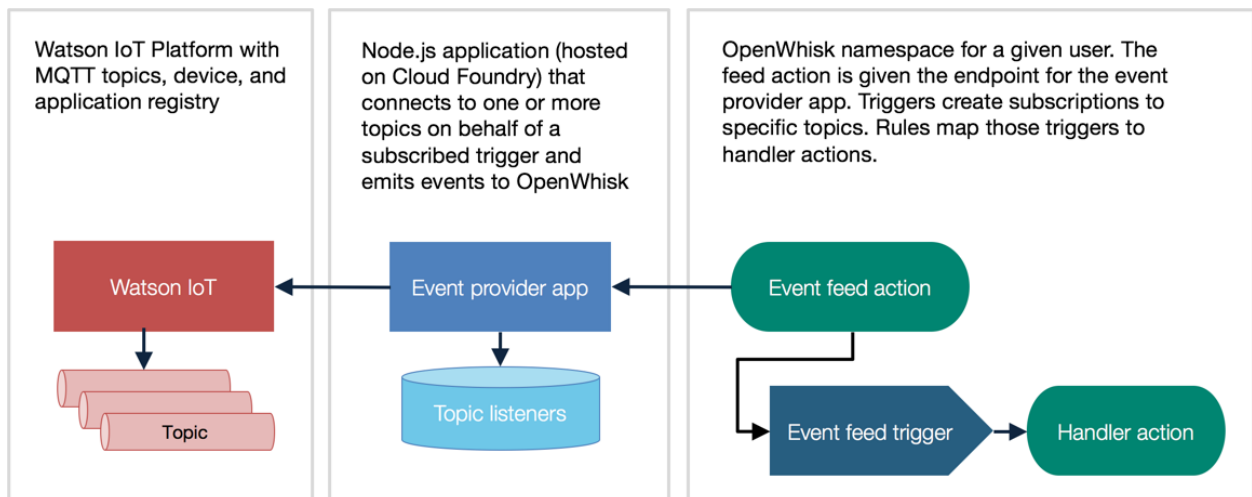
inputs:
  appId: $PUSHAPPID
  appSecret: $PUSHAPPSECRET
pointGeoQuery:
  inputs:
    designDoc: $DESIGNDOC
    indexName: $GEOIDX
    relation: $COVEREDBY
    username: $CLOUDANT_username
    password: $CLOUDANT_password
    host: $CLOUDANT_host
    dbName: $POLYDB
polygonGeoQuery:
  inputs:
    designDoc: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

```

522
523

524 MQTT Package (tailored for Watson IoT)

525 The MQTT package that integrates with Watson IoT provides message topic feeds which can
526 receive events when a message is published. It also defines a trigger to listen to a specific MQTT
527 topic. It describes the OpenWhisk package reposited here: [https://github.com/krook/openwhisk-](https://github.com/krook/openwhisk-package-mqtt-watson)
528 [package-mqtt-watson](https://github.com/krook/openwhisk-package-mqtt-watson).
529



530
531

532 *Manifest File*

533 *with inline values (no Deployment File)*

534 This example makes use of in-line “values” where the developer does not intend to use a separate
535 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
```

536

537 *Deployment File*

538 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

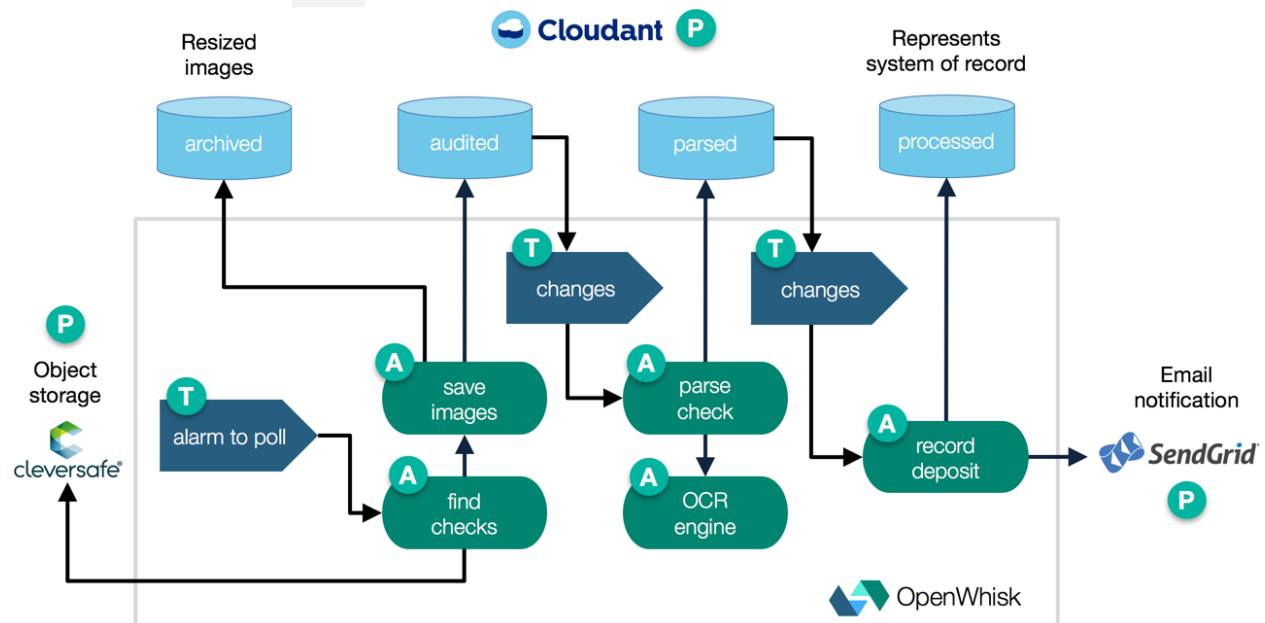
```

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

Check deposit processing with optical character recognition

This use case demonstrates an event-driven architecture that processes the deposit of checks to a bank account using optical character recognition. It relies on Cloudant and SoftLayer Object Storage. On premises, it could use CouchDB and OpenStack Swift. Other storage services could include FileNet or Cleversafe. Tesseract provides the OCR library.

This application uses a set of actions and triggers linked by rules to process images that are added to an object storage service. When new checks are detected a workflow downloads, resizes, archives, and reads the checks then it invokes an external system to handle the transaction.



```
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
      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

```

556 *Deployment Descriptor:*

557

```

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
```

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 OpenWhisk platform within the IBM Bluemix Cloud platform. They are referenced by the term "curated" since they are maintained alongside the OpenWhisk open source code to guarantee compatibility. More information on curated feeds can be found here: <https://github.com/openwhisk/openwhisk/blob/master/docs/catalog.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
```

576
577

578 **Cloudant**

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

581 *Package Manifest*

582 The “cloudant” Package Manifest would appear as follows:

TBD

583 **Public Sources**

584 The following examples are Event Sources that can provide event data to OpenWhisk. We
585 describe them here using this specification’s schema.

586 **GitHub WebHook**

587 Note: the GitHub WebHook is documented here: <https://developer.github.com/webhooks/>.

588

589 A sample description of the GitHub Event Source and its “create hook” API would appear as follows:

TBD

590

591 Other Considerations

592 Tooling interaction

593 Using package manifest directly from GitHub

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

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

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

603 Simplification of WebHook Integration

604 Using RESTify

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

610 Enablement of Debugging for DevOps

611 Isolating and debugging “bad” Actions using (local) Docker

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

614
615 Considerations include, but are not limited to:

- 616 • Isolation on separate “debug” container
- 617 • Recreates “inputs” at time of failure
- 618 • Possibly recreates message queue state
- 619 • Provides additional stacktrace output
- 620 • Provides means to enable “debug” trace output
- 621 • Connectivity to “other” debug tooling

622 Using software debugging (LLDB) frameworks

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

625

Appendices

OpenWhisk Entity Names

Entity Name Tests

Note that Java `/w` matches “word characters” (not just alphanumerics) such as `'_'` (underscore).

Against Java Regex: `\A([\w]/[\w][\w@.-]*[\w@.-]+)\z`

Test String	matches()	n [start(n), end(n)] group(n)	Notes
“a”	true	0: [0,1] a 1: [0,1] a	Shortest Alpha
“1”	true	0: [0,1] 1 1: [0,1] 1	Shortest Numeric
“_1”	true	0: [0,2] _1 1: [0,2] _1	Underscore at beginning
“aAbBcCzZ”	true	0: [0,8] aAbBcCzZ 1: [0,8] aAbBcCzZ	All Alphas
“1234567890”	true	0: [0,10] 1234567890 1: [0,10] 1234567890	All Numerics
“1 2 a Z 9 0”	true	0: [0,11] 1 2 a Z 9 0 1: [0,11] 1 2 a Z 9 0	AlphaNumerics with spaces
“09 @@ _ 4m .. -- aZ”	true	0: [0,23] 09 @@ _ 4m .. -- aZ 1: [0,23] 09 @@ _ 4m .. -- aZ	Multiples of all possible allowed characters.
“Name includes a short domain name is a@b.c”	true	0: [0,42] Name includes a short domain name is a@b.c 1: [0,42] Name includes a short domain name is a@b.c	Sentence-style name with embedded domain name
“_”	true	0: [0,1] _ 1: [0,1] _	Underscore only
“_.”	true	0: [0,2] _. 1: [0,2] _.	Underscore, dot only
“_-@”	true	0: [0,4] _.-@ 1: [0,4] _.-@	Underscore, all special characters only (no spaces)
“___”	true	0: [0,3] ___ 1: [0,3] ___	3 underscores only
a+b	false	N/A	Invalid ‘+’ character
@a	TBD		
-a	TBD		

633 *Against Java Regex: \A([\w][\w@ .-]*[\w@.-]+)\z*

Test String	matches()	Notes
TBD		

634

635 **Acknowledgements**

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

637

638 Castro, Paul

639 Guo, Ying Chun

640 Krook, Daniel

641 Linnemeier, Micah

642 Liu, David

643 Mitchell, Nick

644 Ortelt, Thomas

645 Rutkowski, Matt

646 Santana, Carlos