# OpenWhisk Package Specification

## Version 0.9, Working Draft 01, 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 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, Add. Use cases, examples |
| 0.8.3 | 2017-02-02 | Working Draft 03, Add use cases, examples, $ notation |
| 0.8.4 | 2017-04-18 | Working Draft 04,<br>Support JSON parameter type;<br>Clarify use of Parameter single-line grammar and inferred types.<br>Add support for API Gateway mappings.<br>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.<br>• Updated conceptual Manifest/Deployment File processing images. |
| 0.8.7 | 2017-08-24 | • Added explicit Application entity and grammar.<br>• Added API listing to Package entity.<br>• Cleaned up pseudo-grammar which contained various uses of credentials in places not intended.<br>• 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.<br>• Acknowledge PHP as supported runtime (kind).<br>• Added “sequences” entity as a convenient way to declare action sequences in the manifest. Updated supported runtime values. |
| 0.8.9,<br>0.8.9.1 | 2017-09-22<br>2017-09-29 | • Clarified “version” key requirements for Package (required) and Action (optional); removed from shared entity schema.<br>• Made “license” key optional for package.<br>• keyword “package” (singular) and “packages” (plural) both allowed.<br>• Adjusted use case examples to reflect these changes.<br>• Rework of schema use cases into full, step-by-step examples.<br>• 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,<br>0.9.1 | 2017-11-23,<br>2017-11-30 | • Identified new user scenarios including: clean, refresh, sync, pre/post processing<br>• Clarified “runtime” field on Action is equivalent to “kind” parameter used on the Apache OpenWhisk CLI for Actions.<br>• Added “project” key as an synonym name for “application”.” key, moving application to become deprecated. Project name made required.<br>• Support “public” (i.e., publish) key on Package.<br>• Documented support for the “raw-http” annotation under Action.<br>• Documented support for the “final” annotation under Action.<br>• Documented support for the “main” field under Action.<br>• Dollar Notation section becomes Interpolation / updates<br>    ○ Supported beyond Parameter values<br>    ○ Package names can be interpolated<br>    ○ Annotations values can be interpolated<br>    ○ Multiple replacements supported in same value<br>• Usage scenarios 6-8 added, i.e., Clean, Project Sync, Tool chain support. |

## 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.

**Comment [MR1]:** https://github.com/apache/incubator-openwhisk/blob/master/docs/triggers_rules.md

A trigger that is fired without an accompanying rule to match against has no visible effect. **Triggers cannot be created inside a package; they must be created directly under a namespace. WHY?**

You can create multiple rules that associate the same trigger with different actions. Triggers and rules cannot belong to a package. The rule may be associated with an action that belongs to a package however, for example:

### *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.*

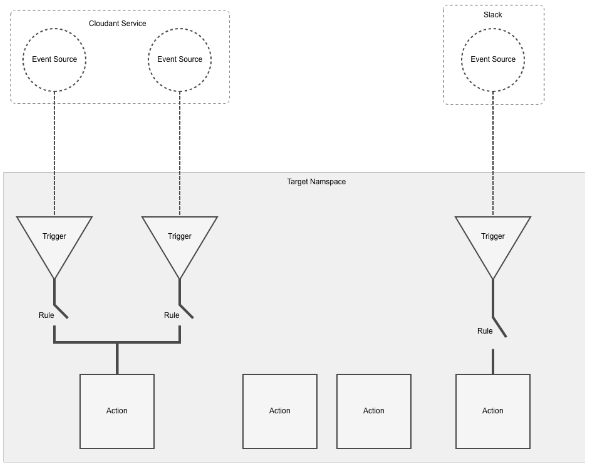
*Note: Not all actions must belong to packages, but can exist under a namespace.*

## Cardinality

### *Trigger to Action*

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

## Conceptual representation



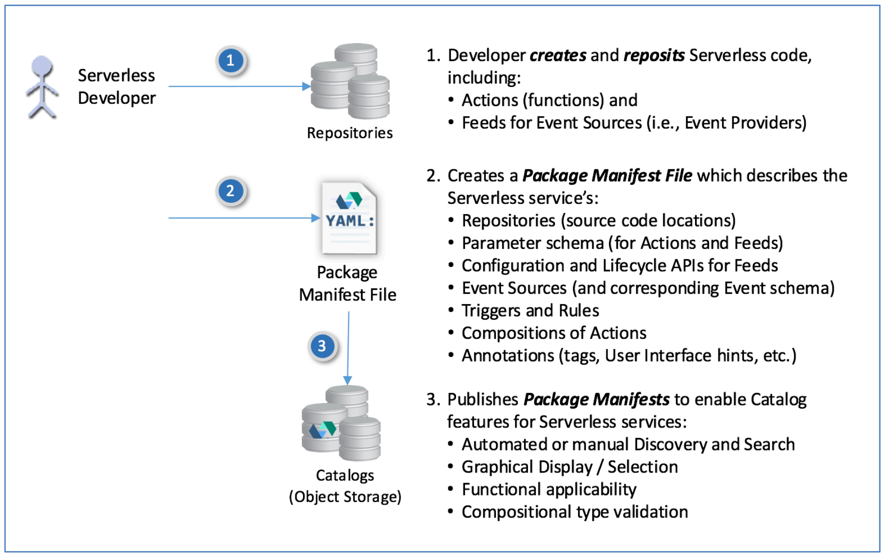
## Package processing

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

- **<u>Package Manifest file</u>**: 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.
- **<u>Deployment file</u>**: 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

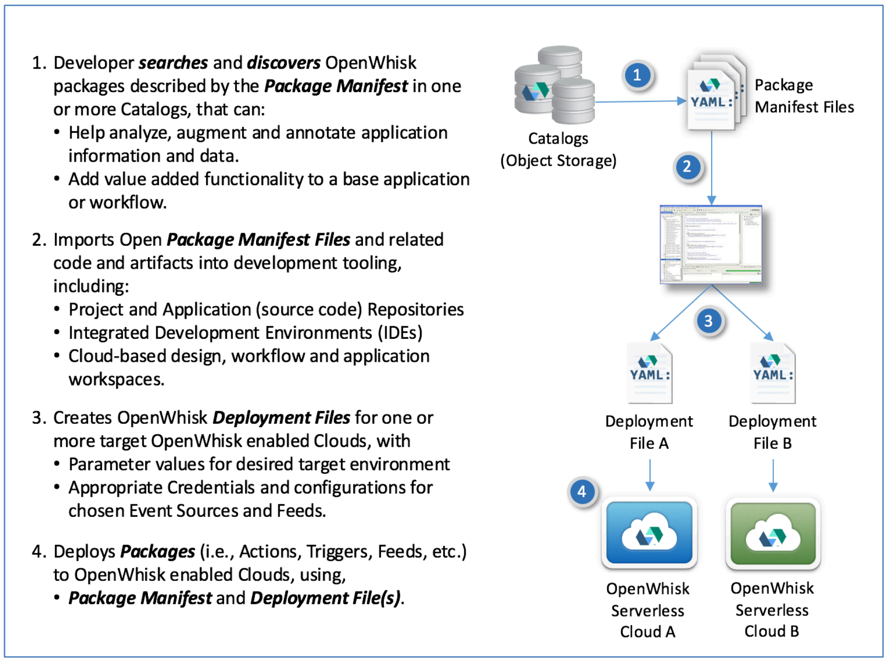
The following diagram illustrates how a developer would create OpenWhisk code artifacts and associate a Package Manifest file that describes them for deployment and reuse.



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

## 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)***.

### *Notes*

- Deployment Files are optional. Deployment can be fully accomplished with simply the Manifest File.

## 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.

### *Requirements*

- 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.
- The maximum name length of any entity name is 256 characters (i.e., ENTITY_NAME_MAX_LENGTH = 256).

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

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

## 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. |

**Comment [MR2]:** Note: Swagger also includes byte (base64 encoded characters) and binary (any sequence of octets) which we can explore later.

**Comment [MR3]:** TBD: Cloud Foundry and other platforms that have packages declare maximums for names, as well as many string values.

**Comment [MR4]:** Note: Swagger references an XML defn. for date and date time: http://xml2rfc.ietf.org/public/rfc/html/rfc3339.html#anchor14

#### *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 types, OpenWhisk supports the types listed in the table below. A complete description of each of these types is provided below.

| 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.<br>Note: found in modern tooling (i.e., "package@version" or "package:version" format).<br><br>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.<br><br>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:<br>• scalar-unit.size<br>• scalar-unit.time<br>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)..<br><br>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.

#### *Example*

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

#### *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.

*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) |

**Comment [MR5]:** TBD: we could expand and allow for any case combination and say we normalize to the unit case?

*Example*

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

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

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

**Comment [MR6]:** TBD: we could expand to allow uppercase and say we normalize to lowercase?

*Example*

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

### Object type example

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

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

**Comment [MR7]:** MUSTFIX

## Schema

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

### General Requirements

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

### map schema

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

#### Single-line grammar

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

#### Multi-line grammar

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

#### Examples

#### Single-line

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

#### Multi-line

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

### Parameter schema

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

#### 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.. |

**Comment [MR8]:** This is effectively JSON data…
We could simplify by removing this, but maps are not a formal YAML construct. We could skip describing this and simply allow JSON data.

**Comment [MR9]:** TBD: **"Dynamic Enumeration"**, have pre-conditions (certain fields have to be provided), the endpoint to provide the value (set for the enum) and post processing may be needed to allow selection in UI (perhaps extracted from a JSON field.

**Last consideration**: Post-process filtering, e.g., may want to exclude certain), excluding records from the record set.

Sometimes the results of a filter need the results of another API call. E.g., Slack… the list channels API like "list PUBLIC" then do a join against your user ID, need to create a post-processing call to fetch your user records.

**Comment [MR10]:** Swagger comparison:

```
  "parameters": [
            {
              "name": "namespace",
              "in": "path",
              "description": "The namespace",
              "required": true,
              "type": "string"
            }
          ],
```

**Comment [MR11]:** TBD: a "bind time hint" which parms does the user suggest values should provide (and not use defaults (can provide at bind time or invocation time).

Users need to be guided with choices

**Comment [MR12]:** Some actions are action specific, whisk gives option to declare parms. At the package-level (binding) for example, an access token for Slack that can be used across multiple Slack actions (at bind time).

**Slack or other example needed. TODO TODO create placeholder matt!!!!!!!!**

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| 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. 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. |

### *Requirements*

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

### *Notes*

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

### *Grammar*

#### *Single-line*

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

- Where `<YAML type>` is inferred to be a YAML type as shown in the YAML Types section above (e.g., `string`, `integer`, `float`, `boolean`, etc.).
- If you wish the parser to validate against a different schema, then the multi-line grammar MUST be used where the value would be supplied on the keyname "value" and the type (e.g., 'json') and/or schema (e.g., OpenAPI) can be supplied.

#### *Multi-line*

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

**Comment [MR11]:** TBD: a "bind time hint" which parms does the user suggest values should provide (and not use defaults (can provide at bind time or invocation time).

Users need to be guided with choices

**Comment [MR13]:** TBD: declare values

**Comment [MR14]: TODO**: Need an Enum. (of string) of supported values

**Comment [MR15]:** TBD: The discussion has been had that for people writing tooling (e.g., wskdeploy) that we should provide a set of descriptive errors (and not simply pass out the error from the target platform provider).

**Comment [MR16]:** TB interpreted as YAML parser would interpret it (inferred) in most cases it is a string.

**Comment [MR17]:** TBD: Need complex type example (i.e., for object type)

**Comment [MR18]:** TODO: link to actual grammar/schema reference.

#### 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

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

#### 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. |

#### Grammar

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

#### Requirements

- 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.
- 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 256 characters.

#### Notes

- 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.

**Comment [MR19]:** TODO: Describe (Likely above) how Namespaces can be applied from Deployment File, and also how Namespaces are inherited (by document xxx) much like CSS style sheets inherit values.

**Formatted Table**

**Comment [MR20]:** TBD: These may have to be NAMESPACED and put into annotations of the actual entity so they are stored in the CouchDB store.

**Comment [MR21]: Methodology** (for UI or other additions): Prototype as annotations, but elevate as needed. Do not re-invent the wheel (let's follow Apple or Android specs.)

**Comment [MR22]:** wsk -i package get /whisk.system/zipaction
ok: got package zipaction

```
{
  "namespace": "whisk.system",
  "name": "zipaction",
  "version": "0.0.13",
  "publish": false,
  "binding": {},
  "actions": [
     {
        "name": "cat",
        "version": "0.0.13",
        "annotations": [
           {
              "key": "exec",
              "value": "nodejs:6"
           }
        ]
     }
  ]
}
```

**Comment [MR23]:** TBD: verify

287 *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<br>• Relative to the Package manifest file.<br>• Relative to the specified Repository. |
| runtime | no | `string` | `N/A` | The required runtime name (and optional version) that the Action code requires for an execution environment.<br><br>*Note: May be optional if tooling allowed to make assumptions about file extensions.* |
| 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.<br><br>*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-export | no | `boolean` | `false` | Optionally, turns the Action into a "web action" causing it to return HTTP content without use of an API Gateway. |
| main | no | `string` | `N/A` | The optional name of the function to be aliased as a function named "main".<br><br>*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".* |
| raw-http | no | `boolean` | `false` | The optional flag to indicate if a Web Action is able to consume the raw contents within the body of an HTTP request.<br><br>Note: this option is ONLY valid if web-export is set to 'true'. |
| final | no | `boolean` | `false` | TODO |

288 *Requirements*

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

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

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

**Comment [MR24]:** TODO: ***yamlparser.go*** has the following fields that are not document:
- location (deprecated)
- Credential
- ExposedURL

**Comment [MR25]:** TBD: how do we reference "stable" version without knowing a number???

**Comment [MR26]:** Nick: 2 use cases
1)Pulling from Docker (what version/tag to use) will vary from 1 source provider to another. It's the version you want to pull from
2) On whisk side this is deployment versioning; typically test/live structure. Can look at how encoded. Tags on source vs. tags on versions.

**Comment [MR27]:** TBD: do we want ORDERED lists? Or allow optional order? Since JSON object does not preserve order? BUT other langs do, but do we care?

**Comment [MR28]:** TODO: it appears "web-export" has been reduced to "web" on CLI, should we discuss allowing an overload for this boolean field with the "web" (shortened) name?

**Comment [MR29]:** **Enabling raw HTTP handling**
Raw HTTP web actions are enabled through the `--web` flag by using a value of `raw`.
```
 wsk action create /guest/demo/hello
hello.js --web raw
```
**Disabling raw HTTP handling**
Disabling raw HTTP can be accomplished by passing a value of `false` or `no` to the `--web` flag.
```
 wsk update create /guest/demo/hello
hello.js --web false
```

**Comment [MR30]:** Note: Cloud Foundry and other platforms that have packages declare maximums for names, as well as many string values.

- 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.

**Comment [MR31]:** TBD: Please verify we want to throw an error, we could ignore for some values, use defaults or maximums. Typically deterministic processing/behavior needs to be documented.

**Comment [MR32]:** Note: SHOULD the following parms be standardized???
payload: msg.trigger_payload || {}, ... [1]

**Comment [MR33]:** Normative? Optional

### *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.

**Comment [MR34]:** https://console.stage1.ng.bluemix.net/docs/openwhisk/openwhisk_reference.html#openwhisk_syslimits

### *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-export: <boolean>
```

**Comment [MR35]:** We COULD have more than one language in same package for same package (user/provider chooses best for their Cloud).

TODO: Review xCode example

### *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
```

**Comment [MR36]:** TODO: need a Web Action example

**Comment [MR37]:** TBD: Do we wish to support inlined code here?

```
  outputs:
    awesome_output_value:
      description: Impressive output string
      type: string
  limits:
    memorySize: 512 kB
    logSize: 5 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; examples of supported runtimes as of this specification version include:

| Runtime value | OpenWhisk kind | 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 |
| `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. |

### *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). The following file extensions are recognized and will be run on the latest version of corresponding Runtime listed below:

| File extension | Runtime used | Description |
|---|---|---|
| `.js` | nodejs | Latest Node.js runtime. |
| `.java` | java | Latest Java language runtime. |
| `.py` | python | Latest Python language runtime. |

**Comment [MR38]:** TBD: provide links to

**Comment [MR39]:** We COULD publish known versions (here or an appendix); however, this has seemed to be a "moving target" recently.

As we approach OpenBeta and GA, and solidify, we SHOULD list here AND add a column for what "current" version maps to.

We could put this as an addendum or eventually link to some more dynamic document in GitHub.

**Comment [MR40]:** **TBD: Daisy: Blackbox (Docker) Actions**: Need to think about how to set the field "runtime" if it is a docker action. We should document it.

**Formatted Table**

**Comment [MR41]:** Kind supports language:default, works in many cases, but not for swift as runtime version is important.

**Comment [MR42]:** Pyc valid?

| File extension | Runtime used | Description |
|---|---|---|
| `.swift` | swift | Latest Swift language runtime. |
| .php | php | Latest PHP language runtime. |

**Comment [MR43]:** Paul: no intention to have more than .swift for now, must use "kind" i.e., runtime (deafult will be swift 3). Node defaults to 6.

### Valid Limit keys

**Comment [MR44]:** https://console.stage1.ng.bluemix.net/docs/openwhisk/openwhisk_reference.html#openwhisk_syslimits

| 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).<br><br>*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.<br><br>*Note: This value is not changeable via APIs at this time.* |
| `codeSize` | scalar-unit.size | 48 MB | *See description* | The maximum size of the Action code.<br><br>*Note: This value is not changeable via APIs at this time.* |
| `parameterSize` | scalar-unit.size | 1 MB | *See description* | The maximum size<br><br>*Note: This value is not changeable via APIs at this time.* |

**Comment [MR45]:** TBD: Why default to 10? Why not default to 0 for production and 10 for test? perhaps some global setting?

**Comment [MR46]:** TBD What does this mean with the namespace changes just made?

### Notes

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

### Web Actions

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

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

**Comment [MR47]:** Using the `--web` flag with a value of `true` or `yes` allows an action to be accessible via REST interface without the need for credentials. A web action can be invoked by using a URL that is structured as follows: `https://{APIHOST}/api/v1/web/{QUALIFIED ACTION NAME}.{EXT}`. The fully qualified name of an action consists of three parts: the namespace, the package name, and the action name.
*The fully qualified name of the action must include its package name, which is* `default` *if the action is not in a named package.*
An example is `guest/demo/hello`. The web action API path can be used with `curl` or `wget` without an API key. It can even be entered directly in your browser.

- **headers**: a JSON object where the keys are header-names and the values are string 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).

### *Trigger entity*

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

#### *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.<br><br>***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.* |

#### *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 above.

#### *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 Package (although there are discussions about changing this in the future). This means that Trigger 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`

#### *Grammar*

```
<triggerName>:
  <Entity schema>
  feed: <feed name>
  credential: <Credential>
  inputs:
    <list of parameter>
```

**Comment [MR48]:** TBD: Can we add Event Schema here????

**Comment [MR49]:** TBD: Can we at some point decribe queue backing/limits (storage), persistence, guaranteed message delivery etc.?

**Comment [MR50]:** TBD: yamlparser.go supports the following fields we do NOT yet document:
- Credential (added here hastily for v0.8.9)
- Namespace
- Source

**Comment [MR51]:** MUSTFIX: Define Event schema/grammar and reference here.

*Example*

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

## Rule entity

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

*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.<br><br>Note: In this version of the specification, only the expression "`true`" is currently supported. |

*Requirements*

- 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.

*Requirements*

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

*Grammar*

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

*Example*

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

## Feed entity

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

At this time, the Package Manifest simply provides the information to describe a Feed (service), its Action, lifecycle operations (along with their parameters) and the associated service it works with. In the future, we intend to allow more granular ability to manage Feeds directly using their operations.

### *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:<br>• A credential string.<br>• 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. |

### *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 using a Deployment File.
- Operation names in manifests MAY be lower or upper cased (e.g., "create" or "CREATE").

### *Grammar*

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

**Comment [MR52]: Curated** Feed: some event sources (like Alarms, Cloudant), those sources had a way to get an event feed. A runtime can intereact with that services API and expose to Whisk API, easy to access with CLI (without going to Cloudant.

**WebHook**:
Can just call whisk tirgger API

(not yet built) **Messaging**: integration with messaging (message hub)

**Comment [MR53]:** TBD: MUSTFIX: Is this false????

**Comment [MR54]:** TODO: Need example use cases for OpenAPI uses of operation (as well as schema).

**Comment [MR55]:** IMO, the Feed Action is a "containment" of the Action within the Feed defintion (i.e., an implementation choice that we expose).

**Comment [MR56]:** This is a "handshake" for composition. The Action says "I am a Feed Action", the Feed defintion must confirm that indeed the Action has been declared to be a Feed Action (by setting `feed: true`).

**Comment [MR57]: MUSTFIX**: need to define Operation grammar elsewhere to reference here.

**Comment [MR58]:** TODO: define operations grammar/structure

```
  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
    ...
```

**Comment [MR59]:** Need to craete a credential example

```
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
```

*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.

**Comment [MR60]:** This is the defn. that seems accurate and we want to represent in this spec. (schema), but then the discussion after this intro. Seems to treat Feed also as some vague entity. Perhaps this is confusion form working on the code to store information regarding the actual Feed service (and its parameters and operations)?

It seems that implementation of how the core interacts with Feeds is being confused after this with the actual Feed service?

**Comment [MR61]: TBD**: Should Whisk lifecycle be "best practice" (i.e., optional) or required? Should this be part of the operations or separated?

**Comment [MR62]:** Again, "Feed Action" is too confusing, these seem to simply be operations of a lifecycle

**Comment [MR63]:** Normative? Optional

**Comment [MR64]:** TODO: Show an example

## Sequence entity

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

*Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| actions | yes | `list of Action` | N/A | • The required list of two or more actions |

*Requirements*

- The comma separated list of Actions on the actions key SHALL imply the order of the sequence (from left, to right).
- There MUST be two (2) or more actions declared in the sequence.

*Notes*

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

*Grammar*

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

*Example*

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

### *API entity*

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

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

*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
```

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

*Grammar*

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

*Note*

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

*Example*

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

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

*Requirements*

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

*Notes*

- The API entity within the OpenWhisk programming model is considered outside the scope of the Package. This means that 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 api list -v`

## Package entity

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

*Fields*

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| version | yes | `version` | N/A | The required user-controlled version for the Package. |

**Comment [MR65]:** See PR https://github.com/openwhisk/openwhisk-wskdeploy/pull/243

1. Current impl. is a list (array); we need a true dep. graph
2. Dep. graph should assure:
a. No cycles
b. dependency order (if this cannot be derived, we need "-" to change grammar to ordered list to impose author provided order).
c. Version resolution; that is, if diff. packages ref. the same dependency, they must be at the same version.
d. Provide warnings for unused dependencies.
---

This is a first cut at adding dependencies to a manifest.yml file. This adds a `dependencies` key where the dependency is a GitHub repo.

```
package:
  name: opentest
  dependencies:
      hellowhisk:
          url:
https://github.com/paulcastro/hellowhisk
          version: 1.0.1
      myCloudant:
           source:
/whisk.system/cloudant
           inputs:
                  dbname: MyGreatDB
  sequences:
    mySequence:
      actions: hellowhisk/greeting,
hellowhisk/httpGet
  triggers:
    myTrigger:
  rules:
    myRule:
      trigger: myTrigger
```

This manifest references a GitHub project aliased as "hellowhisk", version 1.0.1 at the given URL. If `version` is not specified, it will pull from `master`. Dependencies that specify a `source` are interpreted as bindings, and we do a package bind. `url` specifies a GitHub dependency and is treated as an independent deployme ... [2]

**Comment [MR66]:** TBD: yamlparser.go supports the following fields we do NOT yet document here:
- Function (deprecated?)
- Namespace
- ApiHost
- Inputs

**Formatted Table**

**Comment [MR67]:** TBD: how do we reference "stable" version without knowing a number???

**Comment [MR68]:** Nick: 2 use cases
3)Pulling from Docker (what version/tag to use) will vary from 1 source provider to another. It's the version you want to pull from
4) On whisk side this is deployment versioning; typically test/live structure. Can look at how encoded. Tags on source vs. tags on versions.

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| license | no | `string` | `N/A` | The required value that indicates the type of license the Package is governed by.<br><br>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:<br>• A credential string.<br>• 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. |
| 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). |

Formatted Table

Comment [MR69]: TODO: Must have examples!!!!!

### *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.

Comment [MR70]: Note: Cloud Foundry and other platforms that have packages declare maximums for names, as well as many string values.

### *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
```

*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>
```

## Interpolation of values using Environment Variables

*Dollar Notation ($) schema for values*

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

*Syntax*

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

**Comment [MR71]:** May want to create actual, minimalist example here instead of "schema".

**Deleted:** ~~*Schema for accessing*~~

**Comment [MR72]:** authoring the $ notation, another issue with using key=value properties in depl. files is that we have structured (datatypes) as values as well to consider

Credentials is one case, and would be necessary for other "objects" to allow us to consume and produce Inputs/Outputs for use with OpenAPI-defined services

log4j would be one example of collapsing hierarchical representations, but a bit awkward as they support functions for many types (of the language). YAML/JSON has hierarchy already via indentation or braces of course (edited)

Also, the current structure of the depl. file allows packages to referenced from external repos (with desc. of credentials and the cred. keys/values as well)

flattening to something like a properties file would cause us to invent new things to add features like this later (and become kludgey like log4j)

**Deleted:** ~~a parameter~~

**Deleted:** ~~the~~

**Deleted:** ~~the~~

**Deleted:** ~~`parameter`~~

*Example*

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

*Requirements*

- Processors or tooling that encounter ($) Dollar notation and are unable to locate the value in the execution environment SHOULD resolve the value to be the default value for the type (e.g., an empty string (“”) for type ‘`string`’).
- A value binding provided on the ‘value’ key takes precedence over a value binding on the ‘default’ key.

*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). |

Formatted Table

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| 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. |
| 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.<br><br>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. |

**Formatted Table**

**Comment [MR73]:** TBD – Need to define schema

### *Requirements*

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

### *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.
```

**Comment [MR74]: MUSTFIX**: align with sequnce grammar we now support.

**Comment [MR75]:** TODO

### *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
```

**Comment [MR76]:** TBD: show single line grammar as well.

## Extended Schema

### *Dependencies*

The dependencies section allows you to declare other OpenWhisk packages that your application or project (manifest) are dependent on. A Dependency is used to declare these other packages 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. |

**Comment [MR77]:** TBD: yamlparser.go supports the following fields we do NOT yet document here:
- annotations

**Formatted Table**

#### *Requirements*

- No additional requirements.

#### *Notes*

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

#### *Grammar*

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

**Comment [MR78]:** TODO: this is not accurate to GitHub… branches and releases (ZIP files of source) are used.

#### *Example*

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

### 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 |
|---|---|---|---|---|
| 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. |

**Formatted Table**

**Comment [MR79]:** Do we want to formalize a URL beyond string?

#### 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.

**Comment [MR80]:** Note: Cloud Foundry and other platforms that have packages declare maximums for names, as well as many string values.

#### Grammar

##### Single-line (no credential)

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

**Comment [MR81]:** FYI: we could state that some "repos", such as GitHub are well-known and protocol assumed?

##### Multi-line

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

**Comment [MR82]:** FYI: we could state that tooling can prompt for Credentials when not supplied via Environment (mapping file) bindings.

#### 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 | 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. |

**Formatted Table**

**Comment [MR83]:** TBD what should our default be, if none, then we have to make this required…

**Comment [MR84]:** TODO: Support encrypted keys (as we use in testing Feeds) with Bluemix.

**Comment [MR85]:** TBD: what is default? Should there be a default?

| Key Name | Required | Value Type | Default | Description |
|---|---|---|---|---|
| token | yes | `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 | no | `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
```

### Notes

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

Formatted Table

**Comment [MR86]:** Note: removed “user” for SSH keypairs used in OpenStack

Formatted Table

**Comment [MR87]:** TBD: find standard ref. and see if tokentype value can conform to dash and not underscore.

**Comment [MR88]:** TBD: add norm. ref. https://tools.ietf.org/html/draft-ietf-ipsec-isakmp-xauth-06

**Comment [MR89]:** TBD add norm. ref. https://oauth.net/2/
And verify IF we can use Oauth 1 or 2 or both in OW.

**Comment [MR90]:** We should list values here

**Comment [MR91]:** TBD adopt general Object grammar (incl. any description) field

*Examples*

*Plain username-password (no standardized protocol)*

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

*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
```

*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
```

*OAuth bearer token*

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

*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>
```

## Package Artifacts

### 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.
Fields

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

### Project (or Application)

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 | application:
  version: <version>
```

**Comment [MR92]:** *Sample (working draft) towards basic bindings from .env file*
NOTE: env is seeming to be a "deployment" thing
.env files include
- Namespace bindings ()
- Credential binding (is this diff than parameter)?
- Parameter binding (may be diff. from "environment", i.e., a different "feel")

*with serverless, action-level parms. This seems to be overkill since these should not change for diff target clouds.*

**Comment [MR93]:** TBD: yamlparser.go supports the following fields we do NOT yet document here:
- ApiHost

**Formatted Table**

**Deleted:** no

```
  name: <string256>
  namespace: <string>
  credential: <string>
  package:
    <package definition>
```

*Grammar (plural)*

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

*Requirements*

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

**Comment [MR94]:** BUG: ERROR: we do NOT treat this as an error today.

*Notes*

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

*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
    ...
```

*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
    ...
```

*Example Notes*

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

**Formatted Table**

**Deleted:** ,

**Deleted:** Maven-Version ... [3]

## 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/ |

**Formatted Table**

## Scenarios and Use cases

### Usage Scenarios

Comment [MR95]: https://releaseblueprints.ibm.com/display/CLOUDOE/Whisk+Design+for+2016-Q4+Release#WhiskDesignfor2016-Q4Release-ManifestFile

#### *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

Comment [MR96]: Subject to name change based upon agreement

*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*

Comment [MR97]: Paul: this seems like a versioning scenario. Should we leverage this? Or create new scenario?

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.

Comment [MR98]: TBD: rules for update, e.g., overwrite, etc.

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

**Formatted:** Font:

**Formatted:** Font:

**Formatted:** Font:

**Formatted:** Font:

*Notes:*

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

### *Scenario 4: Export into Repository*

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

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

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

**Comment [MR99]:** TODO: Thomas
Default use case for Lambda; see: http://docs.aws.amazon.com/lambda/latest/dg/with-s3.html

Paul: [this is also the] cocoapods model, local repo.

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

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

### *Scenario 6: Clean*

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

### *Scenario 7: Project Sync*

#### *Sync remote project from local*

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

#### *Sync local project from remote*

The user has already deployed a project (manifest) and to a remote OpenWhisk platform. They have modified (i.e., added, updated or deleted entities) in the remotely deployed project (perhaps using the remote platforms UI or the command line interface (CLI). The remote addition, deletion or update of these affected entities should be reflected in the remote OpenWhisk platform.

#### *Clean deployed (non-shared) entities*

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

#### *Create (refresh) project from remote*

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

#### *Add entities to project from local*

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

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

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

## Guided examples

This packaging specification grammar places an emphasis on simplicity for the casual developer who may wish to hand-code a Manifest 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.

This guide will use examples to incrementally show how to use the OpenWhisk Packaging Specification to author increasingly more interesting Package Manifest and Deployment files taking full advantage of the specification's schema.

*Please note that the Apache 'wskdeploy' utility will be used to demonstrate output results.*

Formatted

### Package Examples

#### Example 1: Minimal valid Package Manifest

This use case shows a minimal valid package manifest file.

including:

- shows how to declare a Package named '`hello_world_package`'.

##### *Manifest Files*

###### *Example 1: Minimum valid Package manifest file*

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

###### *Notes*

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

### Actions Examples

#### Example 1: The "Hello world" Action

As with most language introductions, in this first example we encode a simple "hello world" action, written in JavaScript, using an OpenWhisk Package Manifest YAML file.

It shows how to:

- declare a single Action named '`hello_world`' within the '`hello_world_package`' Package.
- associate the JavaScript function's source code, stored in the file '`src/hello.js`', to the '`hello_world`' Action.

### Manifest File

*Example: “Hello world” using a NodeJS (JavaScript) action*

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

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

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

### Deploying

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

### Invoking

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

### Result

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

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

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

### Discussion

This "hello world" example represents the minimum valid Manifest file which includes only the required parts of the Package and Action descriptors.

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

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

It shows how to:

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

### *Manifest File*

*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
```

### *Deployment*

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

### *Invoking*

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

### *Result*

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

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

### *Discussion*

In this example:

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

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

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

It shows how to:

- declare input and output parameters on the action ‘`hello_world`’ using a simple, single-line grammar.
- add two input parameters, ‘`name`’ and ‘`place`’, both of type ‘string’ to the ‘`hello_world`’ action.
- add an '`integer`' parameter, '`age`', to the action.
- add a '`float`' parameter, '`height`', to the action.
- add two output parameters, '`greeting`' and '`details`', both of type '`string`', to the action.

### *Manifest File*

*Example: “Hello world” with typed input and output parameter declarations*

```
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
```

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

```
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 };
}
```

### *Deployment*

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

### *Invoking*

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

### *Result*

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 "
},
```

### *Discussion*

In this example:

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

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

This example builds on the previous '"Hello world" with typed input and output parameters' example with more robust input and output parameter declarations by using a multi-line format for declaration.

**Deleted:** "Hello world" with typed input and output parameters'

This example:

- shows how to declare input and output parameters on the action '`hello_world`' using a multi-line grammar.

### *Manifest file*

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

*Example: input and output parameters with advanced fields*

```
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
```

### *Deployment*

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

### *Invoking*

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

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"
},
```

### *Discussion*

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

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

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

### *Manifest File*

*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
```

### *Deployment*

without the Deployment file:

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

### *Invoking*

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

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

Invoking the action would result in the following response:

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

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

### *Triggering*

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

```
$ wsk trigger fire meetPerson
```

### *Result*

which results in an Activation ID:

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

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

```
$ wsk activation list

d03ee729428d4f31bd7f61d8d3ecc043 hello_world_triggerrule
3e10a54cb6914b37a8abcab53596dcc9 meetPersonRule
5ff4804336254bfba045ceaa1eeb4182 meetPerson
```

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

```
$ wsk activation get d03ee729428d4f31bd7f61d8d3ecc043
```

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"
}
```

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

### *Discussion*

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

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

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

### *Manifest File*

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

*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
```

### *Deployment File*

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

*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
```

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

### *Deploying*

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
```

### *Triggering*

Fire the '`meetPerson`' Trigger:

```
$ wsk trigger fire meetPerson
```

### *Result*

Find the activation ID for the “hello_world_triggerrule’ Action that firing the Trigger initiated and 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"
}
```

### *Discussion*

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

## Github feed

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

### *Manifest File*

```
git_webhook:
  version: 1.0
  license: Apache-2.0
  feeds:
    webhook_feed:
      version: 1.0
      function: github/webhook.js
      runtime: nodejs@6
```

**Comment [MR100]:** ERROR!!!

```
      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
```

**Comment [MR101]:** TBD: Allow "allowed_values" where list of allowed string values can be optionally Listed.

### *Deployment File*

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

## Advanced examples

### Github feed advanced

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

### *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
```

### Deployment File

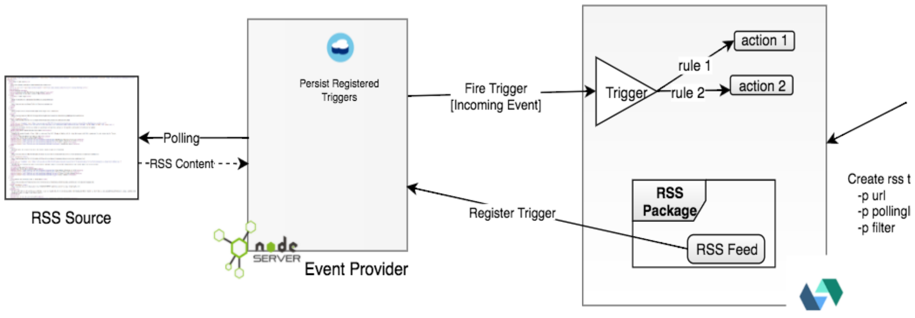
```
packages:
  git_webhook:
    feeds:
      webhook_feed:
        inputs:
          email: daisy@company.com
          title: Github Push Notification
```

## 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 reposited here: https://github.com/openwhisk/openwhisk-package-rss.

**Comment [MR102]:** Note: we may need to describe the Event Source (that is, the "edge" of the Event Provider), the event schema it sends out, and the confiduration. Matt to speak to Vincent about his investigations.

**Comment [MR103]:** 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
```

### Deployment File

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
```

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

## Polygon Tracking

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

### Manifest File:

```
application:
  name: PolygonTracking
```

**Comment [MR104]:** URL Type needed?

**Comment [MR105]:** Paul has a video we will want to post and link to

**Comment [MR106]:** (PAUL): Notes:
-The triggers bind to curated feeds in OpenWhisk. We need a way to describe this binding
- From a dev standpoint, typing in inputs and outputs seems redundant unless there is a purpose. This could be something we do if we want to enable automated mapping of outputs of Action A1 to inputs of Action A2
- This just describes bare minimum and leaves out other fields like "location" of source code. I see where specifying this is useful. We should also support some defaults based on convention, mainly for the dev who may not want to type all this out. Deployers will likely override this with their own settings but it doesn't necessarily have to be set at dev time.

```
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
```

**Comment [MR107]:** TODO: review and update test case with Daniel Krook

**Comment [MR108]:** review and update test case with Daniel Krook

```
        whenPointUpdate:
          trigger:
            polygonUpdate
          action:
            handlePolygonUpdate

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

916 ***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
```

**Comment [MR109]:** -[Paul]:
- Need a better way to describe the binding of an existing package to this current one. Not sure if this should be in a deployment or manifest files
- I added a named package, not sure if we want to support default packages. In OpenWhisk, you can have an action tied directly to a namespace.

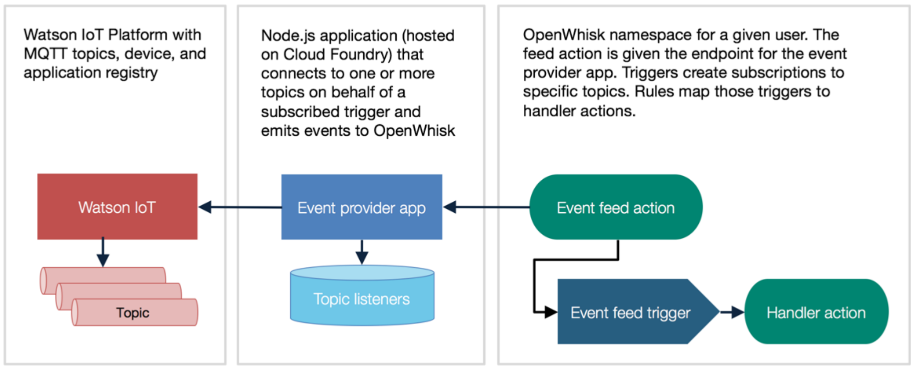
**Comment [MR110]:** TODO: We need to provide schema for this curated Feed binding.

```
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
```

**Comment [PC111]:** Specifying the parameter bindings in this deployment descriptor makes sense to me. Needs to be clear how names are used as keys to map back to manifest file and perhaps defaults, e.g. if we support a default such that there is a "hello.js" file in the directory and a tool sees the file but nothing in the manfiest describes it. Then perhaps filename w/o extension"hello" becomes the key and you can refer to it in the deployment descriptor

## 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.
Watson IoT
Event provider app
Event feed action
Topic
Topic listeners
Event feed trigger
Handler action

### *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
```

### *Deployment File*

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

```
packages:
```

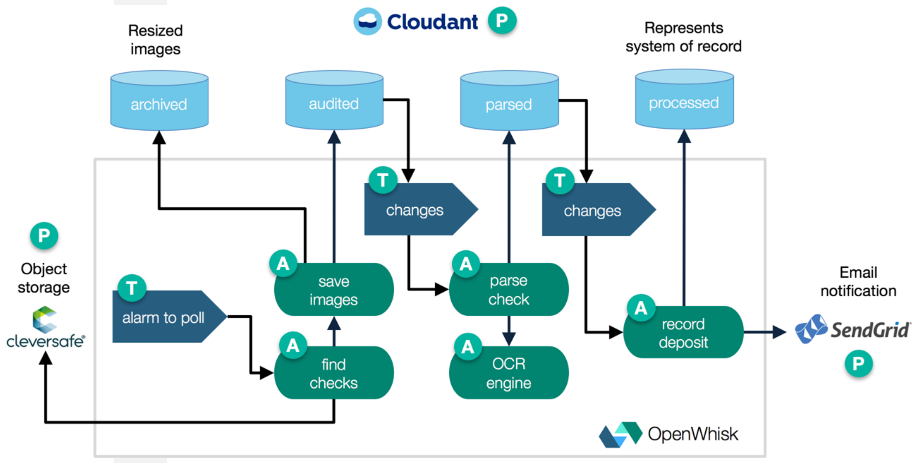
```
mqtt_watson:
  inputs:
    provider_endpoint: http://localhost:8080/mqtt-watson

  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.



948 ***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
          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
```

**Comment [DK112]:** You can see how these triggers, actions, and rules are created in https://github.com/krook/openchecks/blob/master/deploy.sh

**Comment [DK113]:** This sample is not in a package itself, it's at the root at the namespace. There was a bug that made this required (can't recall at the moment)

**Comment [DK114]:** How to add?

```
--feed /whisk.system/alarms/alarm
--feed /$CURRENT_NAMESPACE/checks-db/changes
--feed /$CURRENT_NAMESPACE/checks-db/changes
```

**Comment [DK115]:** These could be package variables. These actions are not in a package (there was a reason for this, can't recall at the moment, may have been a limitation for a previous demo and not this one…)

Most of these params are set at creation time, but the Docker action "parse-check-with-ocr" gets them at invocation time.

**Comment [DK116]:** These all return whisk.async (i.e., no result) on success but do return JSON on failure. The Docker action is the only one with a return value because it's called within a waterfall/promise.

```
        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
```

949 ***Deployment File:***

```
application:
  name: OpenChecks
  namespace: openchecks

  packages:

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

    openchecks:
    credential: ABDCF
    inputs:
      XXX=YYY
```

**Comment [DK117]:** openchecks is not actually in its own package. If it were, there'd be a lot of params here not duplicated below.

```
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:
```

**Comment [MR118]:** TBD: this has been called into question. This SHOULD be treated as one string and allow the Action to process as it sees fit.

```
        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 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:
```

**Comment [MR119]:** See https://github.com/openwhisk/openwhisk-alarms-trigger/blob/master/installCatalog.sh

**Comment [MR120]:** TBD, as this has not been officialy, open sourced?

**Comment [MR121]:** Implies that the following parameters are supported:
- **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.

**Comment [MR122]:** If you know your authorization key and namespace, you can configure the CLI to use them. Otherwise you will need to provide one or both for most CLI operations.

```
wsk property set [--apihost
<openwhisk_baseurl>] --auth
<username:password> --namespace
<namespace>
```

**Comment [MR123]:** TBD: define object?

**Comment [MR124]:** TBD:why are these NOT standardized?

**Comment [MR125]:** **NOTE: MUSTFIX:** This replaces a kludge on the package API where (since feeds are not recognized as top-level entities) an annotation is used to create a "feed" action and setup a trigger/rule automatically, between them.

In this case:

```
    annotations
      parameters: '[ {"name":"cron",
"required":true}, {"name":"trigger_payload",
"required":false} ]'
```

would be passed on the "package update' CLI API. Here we choose to actually define the feed action.

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

### Cloudant

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

#### *Package Manifest*

The “cloudant” Package Manifest would appear as follows:

```
TBD
```

## Public Sources

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

### GitHub WebHook

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

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

```
TBD
```

**Comment [MR126]:** See https://github.com/openwhisk/openwhisk-cloudant-trigger/blob/master/installCatalog.sh

**Comment [MR127]:** echo "Usage: ./installCatalog.sh <authkey> <apihost> <cloudanttriggerhost> <cloudanttriggerport>"
AUTH="$1"
APIHOST="$2"
CLOUDANT_TRIGGER_HOST="$3"
CLOUDANT_TRIGGER_PORT="$4"

CLOUDANT_PROVIDER_ENDPOINT=$CLOUDANT_TRIGGER_HOST':'$CLOUDANT_TRIGGER_PORT
echo 'cloudant trigger package endpoint:' $CLOUDANT_PROVIDER_ENDPOINT

PACKAGE_HOME="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

export WSK_CONFIG_FILE= # override local property file to avoid namespace clashes

echo Installing Cloudant package.

$WSK_CLI -i --apihost "$APIHOST" package update --auth "$AUTH"  --shared yes cloudant \
   -a description "Cloudant database service" \
   -a parameters '[ {"name":"bluemixServiceName", "required":false, "bindTime":true}, {"name":"username", "required":true, "bindTime":true, "description": "Your Cloudant username"}, {"name":"password", "required":true, "type":"password", "bindTime":true, "description": "Your Cloudant password"}, {"name":"host", "required":true, "bindTime":true, "description": "This is usually your username.cloudant.com"}, {"name":"dbname", "required":false, "description": "The name of your Cloudant database"}, {"name":"includeDoc", "required":false, "type": "boolean", "description": "Should the return value include the full documents, or only ... [4]

**Comment [MR128]:**

```
{
  "name": "web",
  "active": true,
  "events": [
    "push",
    "pull_request"
  ],
  "config": {
    "url": "http://example.com/webhook",
    "content_type": "json"
  }
}

# Response
Status: 201 Created
Location:
https://api.github.com/orgs/octocat/hook
s/1
```
... [5]

# 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

**Comment [MR129]:** LLDB, stacktrace, new console,trace(), etc.
http://lldb.llvm.org/

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

The following error types are supported by this specification:

| Name | Error Type | Notes |
|---|---|---|
| CommandError | ERROR_COMMAND_FAILED | Only used in wskdeploy.go, RunCommand(),<br>Which in turn is called by:<br>• Deploy<br>• DeployWithCredentials<br>• DeployProjectPathOnly<br>• DeployManifestPathOnly<br>• Undeploy<br>• UndeplyWithCredentials<br>• UndeployProjectPathOnly<br>• UndeployManifestPathOnly<br><br>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:<br>• Auth key<br>• API Host<br>• Namespace |
| ParameterTypeMismatchError | ERROR_YAML_PARAMETER_TYPE_MISMATCH | |

## Acknowledgements

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

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

**Page 17: [1] Commented** **Matt Rutkowski** **11/3/16 5:43:00 PM**

Note: SHOULD the following parms be standardized???

payload: msg.trigger_payload || {},

maxTriggers: msg.maxTriggers || 1000

**Page 25: [2] Commented** **Matt Rutkowski** **4/19/17 3:06:00 PM**

See PR https://github.com/openwhisk/openwhisk-wskdeploy/pull/243

1. Current impl. is a list (array); we need a true dep. graph

2. Dep. graph should assure:

a. No cycles

b. dependency order (if this cannot be derived, we need "-" to change grammar to ordered list to impose author provided order).

c. Version resolution; that is, if diff. packages ref. the same dependency, they must be at the same version.

d. Provide warnings for unused dependencies.

---

This is a first cut at adding dependencies to a manifest.yml file. This adds a `dependencies` key where the dependency is a GitHub repo.

```
package:
  name: opentest
  dependencies:
      hellowhisk:
          url: https://github.com/paulcastro/hellowhisk
          version: 1.0.1
      myCloudant:
           source: /whisk.system/cloudant
           inputs:
                  dbname: MyGreatDB
  sequences:
    mySequence:
      actions: hellowhisk/greeting, hellowhisk/httpGet
  triggers:
    myTrigger:
  rules:
    myRule:
      trigger: myTrigger
```

This manifest references a GitHub project aliased as "hellowhisk", version 1.0.1 at the given URL. If `version` is not specified, it will pull from `master`.

Dependencies that specify a `source` are interpreted as bindings, and we do a package bind. `url` specifies a GitHub dependency and is treated as an independent deployment. For example, the root package `opentest` refers to entities in the `hellowhisk` package using the package name format. The following happens on deploy

1. wskdeploy downloads and unpacks a zip file of the gihub repo in `$ProjectPath/Packages/<dependencyname>`
2. Deploys dependencies first
3. Deploys root package

This PR does not include dependency graph management so use at your own risk.

There is a use case in tests/usescases/deptest that illustrates a project that has no local source code, just a manifest that combines the entities in a dependency.

**Page 37: [3] Deleted** **Matt Rutkowski** **10/26/17 1:47:00 PM**

| | |
|---|---|
| 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 |

**Page 64: [4] Commented** **Matt Rutkowski** **11/3/16 5:01:00 PM**

```
echo "Usage: ./installCatalog.sh <authkey> <apihost> <cloudanttriggerhost> <cloudanttriggerport>"
AUTH="$1"
APIHOST="$2"
CLOUDANT_TRIGGER_HOST="$3"
CLOUDANT_TRIGGER_PORT="$4"

CLOUDANT_PROVIDER_ENDPOINT=$CLOUDANT_TRIGGER_HOST':'$CLOUDANT_TRIGGER_PORT
echo 'cloudant trigger package endpoint:' $CLOUDANT_PROVIDER_ENDPOINT

PACKAGE_HOME="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

export WSK_CONFIG_FILE= # override local property file to avoid namespace clashes

echo Installing Cloudant package.

$WSK_CLI -i --apihost "$APIHOST" package update --auth "$AUTH"  --shared yes cloudant \
   -a description "Cloudant database service" \
   -a parameters '[ {"name":"bluemixServiceName", "required":false, "bindTime":true}, {"name":"username", "required":true, "bindTime":true, "description": "Your Cloudant username"}, {"name":"password", "required":true, "type":"password", "bindTime":true, "description": "Your Cloudant password"}, {"name":"host", "required":true, "bindTime":true, "description": "This is usually your username.cloudant.com"}, {"name":"dbname", "required":false, "description": "The name of your Cloudant database"}, {"name":"includeDoc", "required":false, "type": "boolean", "description": "Should the return value include the full documents, or only the document ID?"}, {"name":"overwrite", "required":false, "type": "boolean"} ]' \
   -p package_endpoint "$CLOUDANT_PROVIDER_ENDPOINT" \
   -p bluemixServiceName 'cloudantNoSQLDB' \
   -p host '' \
   -p username '' \
   -p password '' \
   -p dbname ''

# Cloudant feed action

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/changes
"$PACKAGE_HOME/actions/changes.js" \
   -t 90000 \
   -a feed true \
   -a description 'Database change feed' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"includeDoc", "required":false} ]'

# Cloudant account actions

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/create-database \
   "$PACKAGE_HOME/actions/account-actions/create-database.js" \
   -a description 'Create Cloudant database' \
   -a parameters '[ {"name":"dbname", "required":true} ]'

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/read-database \
   "$PACKAGE_HOME/actions/account-actions/read-database.js" \
   -a description 'Read Cloudant database' \
   -a parameters '[ {"name":"dbname", "required":true} ]'

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/delete-database \
   "$PACKAGE_HOME/actions/account-actions/delete-database.js" \
   -a description 'Delete Cloudant database' \
   -a parameters '[ {"name":"dbname", "required":true} ]'

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/list-all-databases \
   "$PACKAGE_HOME/actions/account-actions/list-all-databases.js" \
   -a description 'List all Cloudant databases'

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/read-updates-feed \
   "$PACKAGE_HOME/actions/account-actions/read-updates-feed.js" \
   -a description 'Read updates feed from Cloudant account (non-continuous)' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"params", "required":false} ]'

# Cloudant database actions

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/create-document \
   "$PACKAGE_HOME/actions/database-actions/create-document.js" \
   -a description 'Create document in database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"doc", "required":true, "description": "The JSON
document to insert"}, {"name":"params", "required":false} ]' \

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/read-document \
   "$PACKAGE_HOME/actions/database-actions/read-document.js" \
   -a description 'Read document from database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true, "description": "The Cloudant
document id to fetch"}, {"name":"params", "required":false}]' \
   -p docid ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/update-document \
   "$PACKAGE_HOME/actions/database-actions/update-document.js" \
   -a description 'Update document in database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"doc", "required":true}, {"name":"params",
"required":false} ]' \
   -p doc '{}'

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/delete-document \
   "$PACKAGE_HOME/actions/database-actions/delete-document.js" \
   -a description 'Delete document from database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true, "description": "The Cloudant
document id to delete"},  {"name":"docrev", "required":true, "description": "The document revision number"} ]' \
   -p docid '' \
   -p docrev ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/list-documents \
   "$PACKAGE_HOME/actions/database-actions/list-documents.js" \
   -a description 'List all docs from database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"params", "required":false} ]'

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/list-design-documents \
   "$PACKAGE_HOME/actions/database-actions/list-design-documents.js" \
   -a description 'List design documents from database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"includedocs", "required":false} ]' \

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/create-query-index \
   "$PACKAGE_HOME/actions/database-actions/create-query-index.js" \
   -a description 'Create a Cloudant Query index into database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"index", "required":true} ]' \
   -p index ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/list-query-indexes \
   "$PACKAGE_HOME/actions/database-actions/list-query-indexes.js" \
   -a description 'List Cloudant Query indexes from database' \
   -a parameters '[ {"name":"dbname", "required":true} ]' \

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/exec-query-find \
   "$PACKAGE_HOME/actions/database-actions/exec-query-find.js" \
   -a description 'Execute query against Cloudant Query index' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"query", "required":true} ]' \
   -p query ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/exec-query-search \
   "$PACKAGE_HOME/actions/database-actions/exec-query-search.js" \
   -a description 'Execute query against Cloudant search' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true}, {"name":"indexname",
"required":true}, {"name":"search", "required":true} ]' \
   -p docid '' \
   -p indexname '' \
   -p search ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/exec-query-view \
   "$PACKAGE_HOME/actions/database-actions/exec-query-view.js" \
   -a description 'Call view in design document from database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true}, {"name":"view",
"required":true}, {"name":"params", "required":false} ]' \
   -p docid '' \
   -p viewname ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/manage-bulk-documents \
   "$PACKAGE_HOME/actions/database-actions/manage-bulk-documents.js" \
   -a description 'Create, Update, and Delete documents in bulk' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docs", "required":true}, {"name":"params",
"required":false} ]' \
   -p docs '{}'

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/delete-view \
   "$PACKAGE_HOME/actions/database-actions/delete-view.js" \
   -a description 'Delete view from design document' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true}, {"name":"viewname",
"required":true}, {"name":"params", "required":false} ]' \
   -p docid '' \
   -p viewname ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/delete-query-index \
   "$PACKAGE_HOME/actions/database-actions/delete-query-index.js" \
   -a description 'Delete index from design document' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true}, {"name":"indexname",
"required":true}, {"name":"params", "required":false} ]' \
   -p docid '' \
   -p indexname ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/read-changes-feed \
   "$PACKAGE_HOME/actions/database-actions/read-changes-feed.js" \
   -a description 'Read Cloudant database changes feed (non-continuous)' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"params", "required":false} ]'

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/create-attachment \
   "$PACKAGE_HOME/actions/database-actions/create-update-attachment.js" \
   -a description 'Create document attachment in database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true}, {"name":"docrev",
"required":true}, {"name":"attachment", "required":true}, {"name":"attachmentname", "required":true},
{"name":"contenttype", "required":true}, {"name":"params", "required":false} ]' \
   -p docid '' \
   -p docrev '' \
   -p attachment '{}' \
   -p attachmentname '' \
   -p contenttype ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/read-attachment \
   "$PACKAGE_HOME/actions/database-actions/read-attachment.js" \
   -a description 'Read document attachment from database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true}, {"name":"attachmentname",
"required":true}, {"name":"params", "required":false} ]' \
   -p docid '' \
   -p attachmentname ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/update-attachment \
   "$PACKAGE_HOME/actions/database-actions/create-update-attachment.js" \
   -a description 'Update document attachment in database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true}, {"name":"docrev",
"required":true}, {"name":"attachment", "required":true}, {"name":"attachmentname", "required":true},
{"name":"contenttype", "required":true}, {"name":"params", "required":false} ]' \
   -p docid '' \
   -p docrev '' \
   -p attachment '{}' \
   -p attachmentname '' \
   -p contenttype ''

$WSK_CLI -i --apihost "$APIHOST" action update --auth "$AUTH" --shared yes cloudant/delete-attachment \
   "$PACKAGE_HOME/actions/database-actions/delete-attachment.js" \
   -a description 'Delete document attachment from database' \
   -a parameters '[ {"name":"dbname", "required":true}, {"name":"docid", "required":true}, {"name":"docrev",
"required":true}, {"name":"attachmentname", "required":true}, {"name":"params", "required":false} ]' \
   -p docid '' \
   -p docrev '' \
-p attachmentname ''
```

**Page 64: [5] Commented** **Matt Rutkowski** **11/3/16 5:02:00 PM**

```
{
  "name": "web",
  "active": true,
  "events": [
    "push",
    "pull_request"
  ],
  "config": {
    "url": "http://example.com/webhook",
    "content_type": "json"
  }
}
```

```
# Response
Status: 201 Created
Location: https://api.github.com/orgs/octocat/hooks/1
{
  "id": 1,
  "url": "https://api.github.com/orgs/octocat/hooks/1",
  "ping_url": "https://api.github.com/orgs/octocat/hooks/1/pings",
  "name": "web",
  "events": [
    "push",
    "pull_request"
  ],
  "active": true,
  "config": {
    "url": "http://example.com",
    "content_type": "json"
  },
  "updated_at": "2011-09-06T20:39:23Z",
  "created_at": "2011-09-06T17:26:27Z"
}
```