Yang Best Practices

Thomas D. Nadeau, Brocade
Reinaldo Penno, Cisco

LINUX FOUNDATION
COLLABORATIVE PROJECTS

Agenda

- ODL Yang tools
- Other Useful Yang tools
- Other Communities Working on Yang Models
- ODL Yang Recommendations and caveats

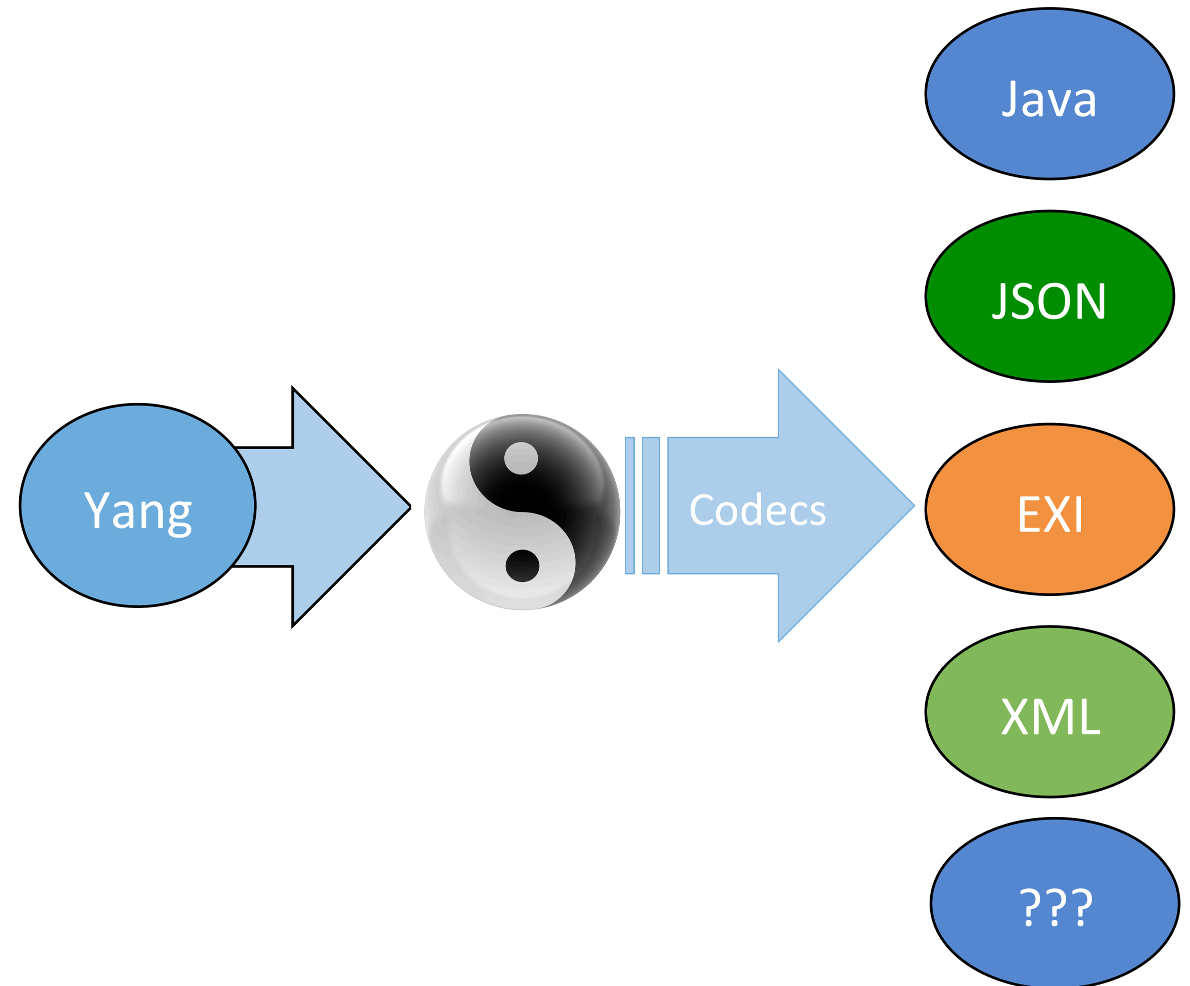
Yangtools – What is Yang?

- Yang is a modeling language
- Models semantics and data organization
- Models can be ‘augmented’
- Can model:
  - Config/Operational data as a tree
  - RPCs
  - Notifications
  - Text base
  - Simple Compact
- Standard based (RFC 6020)

OPENDAYLIGHT

What do Yangtools do?

- Generate Java code from Yang Inputs
- Provides Codecs used to convert
  - Generated Java classes to DOM
  - DOM to various formats
    - XML
    - JSON
    - Etc
- Codecs make data-driven code generation possible:
  - RESTCONF
  - Netconf
  - Other bindings (AMQP expected this summer)
  - Model translations, etc… possible



OPENDAYLIGHT

OPEN DAYLIGHT

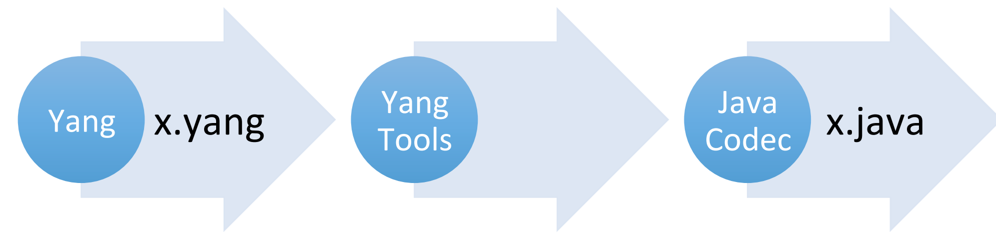
Yang	Java

typedef bridge-name {
        type string;
}
	public class BridgeName implements Serializable {
	    private final String _value;
	
	    @ConstructorProperties("value")
	    public BridgeName(String _value) {
	        …
	    }
	    public BridgeName(BridgeName source) {
	        this._value = source._value;
	    }
	
	    public String getValue() {
	        return _value;
	    }
	    …
	}

Yang	Java

`grouping bridge-attributes {`
`  leaf bridge-name {`
`    type bridge-name;`
`  }`
`  …`
`}`
	`public interface BridgeAttributes`
	`  extends DataObject {`
	`    BridgeName getBridgeName();`
	`    …`
	`}`

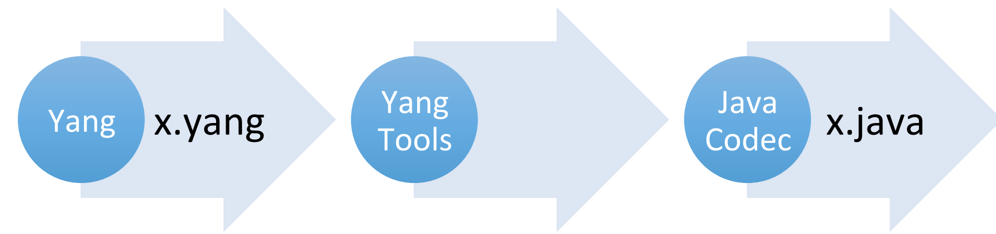
OPEN DAYLIGHT



Yang	Java

`container connection-info {`
`    uses connection-info-attributes;`
`}`
	`public interface ConnectionInfo`
	`    extends Augmentable<ConnectionInfo>,`
	`            ConnectionInfoAttributes {`
	
	`}`

OPEN DAYLIGHT



Yang	Java

Yang

```
container connection-info {
    uses connection-info-attributes;
}
```

Java

```
public class ConnectionInfoBuilder
    implements Builder <ConnectionInfo> {
      /* fields */
      public void setRemoteIp(IpAddress value)
      …
      public IpAddress getRemoteIp()
      …
      public ConnectionInfo build() {
        return new ConnectionInfoImpl(this);
      }
}
```

OPEN DAYLIGHT

Yang	Java

`list controller-entry {`
`key "target"`
`leaf target {`
`type inet:uri;`
`}`
`}`
	`public interface ControllerEntry`
	`extends Augmentable<ControllerEntry>,`
	`Identifiable<ControllerEntryKey> {`
	`Uri getTarget();`
	
	`ControllerEntryKey getKey();`
	`…`
	`}`

OPEN DAYLIGHT

Yang	Java

<pre>list controller-entry {
    key “target”
    leaf target {
        type inet:uri;
    }
}</pre>
	<pre>public class ControllerEntryBuilder
	    implements Builder <ControllerEntry> {
	        /* fields */
	        public ControllerEntryBuilder setTarget(Uri value) {
	          …
	        }
	        …
	        public Uri getTarget(Uri value) {…}
	        ControllerEntryKey getKey() {…}
	        …
	        public ControllerEntry build() {
	          return new ControllerEntryImpl(this);
	        }
	        …
	}</pre>

OPEN DAYLIGHT

Yang	Java

rpc hello-world {
  input {
    leaf name {
      type string;
    }
  }
  output {
    leaf greating {
      type string;
    }
  }
}
	public interface HelloService
	  extends RpcService {
	    Future<RpcResult<HelloWorldOutput>> helloWorld(
	          HelloWorldInput input);
	
	}

OPEN DAYLIGHT

Yang	Java

`rpc hello-world {`
`  input {`
`    leaf name {`
`      type string;`
`    }`
`  }`
`  output {`
`    leaf greating {`
`      type string;`
`    }`
`  }`
`}`
	`public interface HelloWorldInput`
	`    extends DataObject,`
	`    Augmentable<HelloWorldInput> {`
	
	`      String getName();`
	
	`}`

OPEN DAYLIGHT

Yang	Java

`rpc hello-world {`
`input {`
`leaf name {`
`type string;`
`}`
`}`
`output {`
`leaf greating {`
`type string;`
`}`
`}`
`}`
	`public class HelloWorldInputBuilder`
	`implements Builder <HelloWorldInput> {`
	`/* fields */`
	
	`public HelloWorldInputBuilder setName(String value) {`
	`this._name = value;`
	`return this;`
	`}`
	`public HelloWorldInput build() {`
	`return new HelloWorldInputImpl(this);`
	`}`
	`…`
	`}`

OPEN DAYLIGHT

Yang	Java

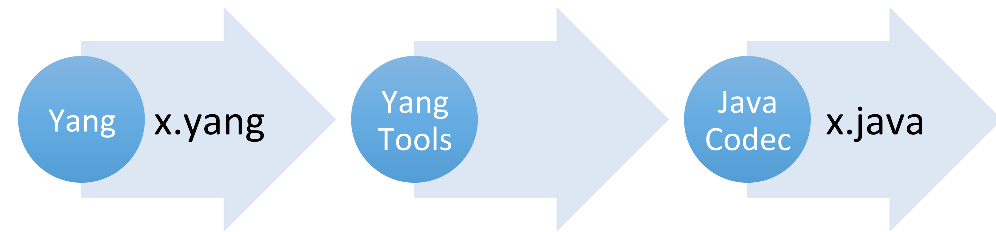
<pre>rpc hello-world {
  input {
    leaf name {
      type string;
    }
  }
  output {
    leaf greating {
      type string;
    }
  }
}</pre>
	<pre>public interface HelloWorldOutput
	    extends DataObject,
	    Augmentable<HelloWorldOutput> {
	
	    String getGreating();
	
	}</pre>

OPEN DAYLIGHT

Yang	Java

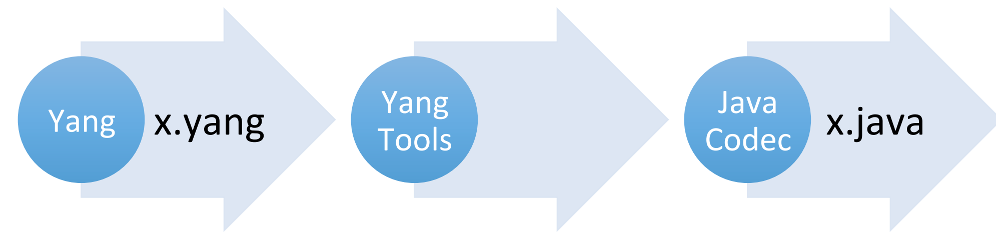
<pre>rpc hello-world {
   input {
     leaf name {
       type string;
     }
   }
   output {
     leaf greating {
       type string;
     }
   }
}</pre>
	<pre>public class HelloWorldOutputBuilder
	    implements Builder <HelloWorldOutput> {
	      /* fields */
	
	      public HelloWorldOutputBuilder setName(String value) {
	            this._name = value;
	            return this;
	      }
	      public HelloWorldOutput build() {
	            return new HelloWorldOutputImpl(this);
	      }
	      …
	}</pre>

OPEN DAYLIGHT



Yang	Java

notification random-greeting-notification
{
    leaf random-greeting {
        type string;
    }
}
	public interface RandomGreetingNotification
	    extends ChildOf<DataObject>,
	    Augmentable<RandomGreetingNotification>,
	    Notification {
	        String getRandomGreeting();
	}



Yang	Java

`notification random-greeting-notification {`
`leaf random-greeting {`
`type string;`
`}`
`}`
	`public class RandomGreetingNotificationBuilder implements Builder<RandomGreetingNotification> {`
	`public RandomGreetingNotificationBuilder setRandomGreeting(String value) {`
	`this._randomGreeting = value;`
	`return this;`
	`}`
	`public RandomGreetingNotification build() {`
	`return new RandomGreetingNotificationImpl(this);`
	`}`
	`…`
	`}`

Yang to Java benefits

- Consistent Data Transfer Objects (DTOs) everywhere
  - **Immutable**: to avoid thread contention
  - **Strongly typed:** reduce coding errors
  - **Consistent**: reduce learning curve
  - **Improvable** – generation can be improved and all DTOs get those improvements immediately system wide
  - **Automated Bindings**:
    - RestConf– xml and json
    - NETCONF
    - amqp and xmpp – on the horizon
  - **Can be generated at runtime**
    - Both client and server sides

OPENDAYLIGHT

Other Available Tools

- http://www.yangvalidator.com/
  - Can post your drafts
  - Extraction and compilation functions
  - https://github.com/cmoberg/bottle-yang-extractor-validator
- NETCONF Central
  - http://netconfcentral.org
  - Tools, tips, models, etc...

Other Communities working on Yang Models

- Github Yang Models
  - https://github.com/YangModels
  - Some vendor, experimental from IEEE, IETF, ODL, etc…
- IETF
  - NETMOD Working Group
    - https://datatracker.ietf.org/wg/netmod/documents/
  - Model Coordination Group
    - http://www.ietf.org/iesg/directorate/yang-model-coordination-group.html
  - Yang Doctors
    - https://www.ietf.org/iesg/directorate/yang-doctors.html

Other Communities Working on Yang Models

- IEEE
  - https://github.com/YangModels/yang/tree/master/experimental/ieee
- Open Config
  - http://openconfig.org
  - https://github.com/YangModels/yang/tree/master/openconfig
- MEF
  - http://www.metroethernetforum.org/PDF_Documents/technical-specifications/MEF_38.pdf
  - http://www.metroethernetforum.org/PDF_Documents/technical-specifications/MEF_39.pdf

ODL Yang Recommendations and caveats

Why use Yang models?

- Cornerstone of ODL infrastructure
- Data model driven development
- Easy to understand and debug issues across applications
- Persistence and Clustering
- Yang Models + Datastore = Anonymous Point to Multipoint messaging system
- JSON bindings, RESTconf support, RPC support

OPENDAYLIGHT

Versioning Yang models can be problematic

- Revision date included in package
- Import statements include revision. Sometimes multiple independent revisions
- Changing and aligning imports in large project can be a daunting task

```
import org.opendaylight.yang.gen.v1.urn.cisco.params.xml.ns.yang.sfc.rsp.rev140701.rendered.service.paths.RenderedServicePath;
import org.opendaylight.yang.gen.v1.urn.cisco.params.xml.ns.yang.sfc.rsp.rev140701.rendered.service.paths.rendered.service.path
import org.opendaylight.yang.gen.v1.urn.cisco.params.xml.ns.yang.sfc.sf.rev140701.service.functions.ServiceFunction;
import org.opendaylight.yang.gen.v1.urn.cisco.params.xml.ns.yang.sfc.sff.rev140701.ServiceFunctionForwarders;
```

OPENDAYLIGHT

Model Validation Limitations

- RFC6020, section 8 states several data store constraints such as
  - Any "must" constraints MUST evaluate to "true".
  - Any referential integrity constraints defined via the "path" statement MUST be satisfied.
  - Any "unique" constraints on lists MUST be satisfied.
  - The "min-elements" and "max-elements" constraints are enforced for lists and leaf-lists.
- See section 8 for complete details

Model Validation Limitations

- It is common for people new and experienced in Opendaylight to be surprised that some of these are not enforced.
- The most common pitfall is the fact that path integrity constraints are not validated. Example:
- ```
  typedef service-function-path-ref {
    type leafref {
      path "/sfc-sfp:service-function-paths/" +
        "sfc-sfp:service-function-path/sfc-sfp:name";
    }
    description
      "This type is used by data models that need to reference
       configured service functions.";
  }
  ```
  ```

Model Validations

- In Lithium we recently fixed:
  - The "min-elements" and "max-elements" constraints are enforced for lists and leaf-lists.
  - Range constraints map to the appropriate Java type

List vs. Leaf-List

- In general better to use List even if you do not really need keys.
- Easier to control ordering
- Removal, update or creation of new elements are based on keys, therefore fully specific Instance Identifiers are possible such as:

```
InstanceIdentifier<SffServicePath> sfStateIID =
        InstanceIdentifier.builder(ServiceFunctionForwardersState.class)
                .child(ServiceFunctionForwarderState.class,
serviceFunctionForwarderStateKey)
                .child(SffServicePath.class, sffServicePathKey).build();
```

- No need to loop for ordered insert or removal

OPENDAYLIGHT

ENUMs vs. Identities

- If you want a list of pre-defined values in a model that can be easily extended by third-parties identities are recommended.
- The most common example is RFC7224. Interface types are just a long list of identifies that is easy to extend.
- Example in the Service Function Chaining Project:
-
```
identity firewall {
  base "service-function-type-identity";
  description "Firewall";
}

identity dpi {
  base "service-function-type-identity";
  description "Deep Packet Inspection";
}

identity napt44 {
  base "service-function-type-identity";
  description "Network Address and Port Translation 44";
}
```

OPENDAYLIGHT

Identity Indirection Model

```
identity interface-type {
        description
          "Base identity from which specific
           interface types are derived.";
      }
 identity iana-interface-type {
        base if:interface-type;
        description
          "This identity is used as a base for
           all interface types defined in the
          'ifType definitions' registry.";
      }
```

```
identity iso88023Csmacd {
      base iana-interface-type;
      status deprecated;
      description
        "Deprecated via RFC 3635.
         Use ethernetCsmacd(6) instead.";
      reference
        "RFC 3635 - Definitions of Managed
         Objects for the Ethernet-like
         Interface Types";
    }
```

OPENDAYLIGHT

Identity Indirection Model

- Easy to create a new "namespace" such as `identity <vendor>-interface-type`
- Easy to create new interface types without touching or revving base model: `identity reinaldoInterface`
- Good for multi-vendor, multi-organization projects like Opensource

Augmentation

- One of the problems with using augmentations is that the code generated use names like ‘Augmentation1’, ‘Augmentation2’ for the classes.
- In order to give the generated classes meaningful names you can use ext:augment-identifier . Example:

```
augment "/sfc-sff:service-function-forwarders/"
augment "/sfc-sff:service-function-forwarders/"
      + "sfc-sff:service-function-forwarder/"
      + "sfc-sff:sff-data-plane-locator" {

  ext:augment-identifier "sff-ovs-locator-options-augmentation";
  uses options;
}
```

```
SffOvsLocatorOptionsAugmentation sffDataPlaneLocatorOvsOptions =
sffDataPlaneLocator.getAugmentation(SffOvsLocatorOptionsAugmentation.class);
```

No support for default statement

- **RFC6020, section 7.6.1. The leaf's default value**
- “The default value of a leaf is the value that the server uses if the leaf does not exist in the data tree.”
- Today there is no support for default statement.
- This also catches many new and experienced developers off guard
- You must supply a value for all leaves otherwise the object is not created at all. Notice that this is not the same as having a leaf with value of null or zero, the containing object is not at all created.

No validation of RESTconf operations

- This is caveat certainly makes the “top 3”
- Any changes to the datastore via RESTconf can not be validated by the application
- Meaning, you can not prevent a user from adding, deleting, updating the datastore in catastrophic ways.
- The only way to apply business logic to a RESTconf operation is through RPCs, which unfortunately are much more awkward to use.

Many other caveats, why?

- In Yang there are many different ways to accomplish the same design
- When designing and using a rich model (choices, groups, identities, paths, multiple hierarchies, amongst others) you might stumble upon an issue
- Standards always evolving, interoperability and expectations change.
- It is a self-correcting issue as more and more people use Yang and bugs get fixed
- But we could benefit from recommended design patterns since the number of different ways to accomplish the same data model can be rather large.