Adapter example

This section contains the following topics:

Step 1: Create a new adapter

In a terminal window, open a command-line terminal and run:

```
cwm-sdk create-adapter -vendor vendor1 -product product1 -feature feature1
```

Now you have a new catalog named `vendor1.product1` at your home dierctory with the following contents:

```
Makefile
adapter.properties
docs
go
proto

 ./docs:
  index.html
 ./go:
  common
  go.mod
  feature1

  ./go/common:
   errors.go
   logger.go
  ./go/feature1:

 ./proto:
  vendor1.product1.common.adapter.proto
  vendor1.product1.feature1.adapter.proto
```

Step 2: Define mock activity

The Adapter SDK has generated the `.proto` files. In the `vendor1.product1.feature1.adapter.proto` file, define the interface of the adapter:

Step 1 Open the `vendor1.product1.feature1.adapter.proto` file with a text editor or inside a terminal window. The contents are as below.

```
syntax = "proto3";

package vendor1.product1.feature1;

option go_package = "cisco.com/cwm/adapters/vendor1/product1/feature1";

import "google/protobuf/struct.proto";

service Activities {

 // CWM SDK NOTE: Activity functions are defined as RPCs here e.g.

 /* Documentation for MyActivity */
 rpc MyActivity(MyRequest) returns (MyResponse);
}

// CWM SDK NOTE: Messages here e.g.

/* Documentation for MyRequest */
message MyRequest {
 string                stringInput  = 1;
 int32                 integerInput = 2;
 bool                  booleanInput = 3;
 google.protobuf.Value anyInput     = 4; // CWM SDK NOTE: Useful for accepting a json object from
the workflow definition
}

/* Documentation for MyResponse */
message MyResponse {
 string                stringOutput  = 1;
 int32                 integerOutput = 2;
 bool                  booleanOutput = 3;
 google.protobuf.Value anyOutput     = 4; // CWM SDK NOTE: Useful for returning a json object to
the workflow definition
}
```

Step 2 To define your activity, replace the placeholder 'MyActivity' with a mock 'Hello' activity, along with the MyRequest and MyResponse placeholder names and message parameters as shown below:

```
service Activities {
 /* Documentation for Hello Activity */
 rpc Hello(MyRequest) returns (MyResponse);
}

/* Documentation for MyRequest */
message MyRequest {
 string name = 1;
}

/* Documentation for MyResponse */
message MyResponse {
  string message = 1;
 }
```

Step 3: Generate adapter source code

Step 1 Based on the `adapter.proto` file that you have edited and on the remaining `.proto` files, generate the source **go** code for the adapter and inspect the files. In the main adapter directory, run:

```
cwm-sdk update-adapter && ls
```

```
The output will look like:
```

```
 .go/
  common
  go.mod
  feature1
  main.go

  go//common:
  errors.go
  logger.go
  vendor1.product1.common.adapter.pb.go

  go//feature1:
  activities.go
  adapter.go
  vendor1.product1.feature1.adapter.pb.go
```

Step 2 **Note** The `.adapter.pb.go` files should not be edited manually.

The `.adapter.pb.go` files generated using the **Protobufs compiler** define all the messages from the `adapter.proto` files.

Step 3 The generated `activities.go` file contains stubs for all the RPCs you have defined in the `.adapter.proto` file. Open the file:

```
 package feature1

 import (
  "cisco.com/cwm/adapters/vendor1/product1/common"
  "context"
 )

 func (adp *Adapter) Hello(ctx context.Context, req *MyRequest, cfg *common.Config) (*MyResponse,
error) {

  var res *MyResponse
  var err error

  // CWM SDK NOTE: Implement your activity logic here...

  return res, err
 }
```

Step 4 Edit the file to return a message:

```
func (adp *Adapter) Hello(ctx context.Context, req *MyRequest, cfg *Config) (*MyResponse, error) {
 return &MyResponse {Message: "Hello, " + req.GetName() + "!"}, nil
}
```

Define another activity

If you wish to add another activity to the existing feature set (**go** package):

Step 1 Open and edit the `adapter.proto` file and define another activity underneath the existing one:

```
service Activities {
 rpc Hello(MyRequest) returns (MyResponse);
 rpc Fancy(MyRequest) returns (MyResponse);
}
```

Step 2 Update the activities **go** code using the SDK:

```
cwm-sdk extend-adapter -activity fancy -feature feature1
```

After you update the **fancy** activity part of the `.adapter.proto` file with a sample logic, update the adapter:

```
cwm-sdk update-adapter
```

Once the code is generated, the `activities.go` file is updated with the new 'Fancy' activity stub, while the code for the 'Hello' activity remains.

Step 4: Add another feature

If you wish to add another feature (**go** package) to the example adapter, use the `extend-adapter` command. In the main adapter directory, run:

```
cwm-sdk extend-adapter -feature feature2
```

Step 1 A new `vendor1.product1.feature2.adapter.proto` file has been added for your adapter:

```
.proto/
  vendor1.product1.common.adapter.proto
  vendor1.product1.feature2.adapter.proto
  vendor1.product1.feature1.adapter.proto
```

Step 2 To define activities for the new feature, open the `vendor1.product1.feature2.adapter.proto` file, and modify the contents accordingly:

```
syntax = "proto3";

package vendor1.product1.feature2;

option go_package = "cisco.com/cwm/adapters/vendor1/product1/feature2";

import "google/protobuf/struct.proto";

service Activities {
   /* Documentation for Goodbye Activity */
```

```
  rpc Goodbye(MyRequest) returns (MyResponse);
 }

 /* Documentation for MyRequest */
 message MyRequest {
  string name = 1;
 }

 /* Documentation for MyResponse */
 message MyResponse {
  string message = 1;
 }
```

Step 3 Generate the code for the 'feature2' package and activities.

```
cwm-sdk update-adapter -features feature2
```

```
.go/goodbyes
 activities.go
 adapter.go
 vendor1.product1.feature2.adapter.pb.go
```

Step 5: Create an installable archive

```
cwm-sdk create-installable
```

The generated `tar.gz` archive contains the all required files of the adapter and can be installed in CWM. The **go** vendor command has been executed in order to eliminate any external dependencies.