Adapter example

This section contains the following topics:

Adapter example

This tutorial is a step-by-step instruction on building an example adapter using the Workflow Adapter SDK. It gives an idea on the adapter structure and on how you provide input to define adapter activities to be consumed by a workflow worker. Before you start, you need to go through the Prerequisites section to set up your development environment.

Step 1: Create new adapter

In a terminal window, open your `cwmsdk` repository directory and run:

```
cwm-sdk create-adapter -location ~/your_repo/adapters -vendor companyX -feature featureX
-product productX
```

Now you have a directory in `adapters` named `companyX.productX` with the following contents:

```
Makefile
adapter.properties
go
proto

 ./go:
  common
  go.mod
  featureX

  ./go/common:

  ./go/featureX:

 ./proto:
  cisco.cwm.sdk.resource.proto
  companyX.productX.common.adapter.proto
  companyX.productX.featureX.adapter.proto
```

Step 2: Define mock activity

The Adapter SDK has generated the `.proto` files. In the `companyX.productX.featureX.adapter.proto` file, define the interface of the adapter:

Step 1 Open the `companyX.productX.featureX.adapter.proto` file with a text editor or inside a terminal window. The contents are as below.

```
syntax = "proto3";

package productXfeatureX;

option go_package = "www.cisco.com/cwm/adapters/companyX/productX/featureX";

service Activities {
 // NOTE: Activity functions are defined as RPCs here e.g.

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

// NOTE: Messages here e.g.

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

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

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(Request) returns (Response);
}

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

/* Documentation for Response */
message Response {
 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:

```
make generate-model && ls

 .go/
  common
  go.mod
  featureX

  go//common:
  companyX.productX.common.adapter.pb

  go//featureX:
  companyX.productX.featureX.adapter.pb

The `.adapter.pb.go` files generated using the **Protobufs compiler** define all the messages from
the `adapter.proto` files.
!!! caution
 The `.adapter.pb.go` files should not be edited manually.
```

Step 2 Now generate the **go** code for the defined activities. In the main adapter directory, run:

```
 make generate-code && ls

 .go/
  common
  go.mod
  featureX
  main.go

  go//common:
  companyX.productX.common.adapter.pb.go

  go//featureX:
  activities.go
  adapter.go
  companyX.productX.featureX.adapter.pb.go
```

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

```
 package featureX

 import (
  "context"
  "errors"
  "go.temporal.io/sdk/activity"
 )

 func (adp *Adapter) Hello(ctx context.Context, req *Request, cfg *Config) (*Response, error) {

  activity.GetLogger(ctx).Info("Activity Hello called")

  var res *Response
  var err error

  if ctx == nil {
   return nil, errors.New("Invalid context")
  }

  if req == nil {
   return nil, errors.New("Invalid request")
  }

  if cfg == nil {
   return nil, errors.New("Invalid config")
  }
```

```
 cancel := ctx.Done()
 done := make(chan any)

 go func() {

  //
  // NOTE:
  //
  // Enter activity code to set response and error here...
  //
  // Perform step 1
  //
  // ...
  //
  // activity.activity.RecordHeartbeat(ctx, "Activity completed step 1")
  //
  // Perform step 2
  //
  // ...
  //
  // activity.activity.RecordHeartbeat(ctx, "Activity completed step 2")
  //
  // ...
  //
  // All logic steps are completed
  //

  done <- nil
 }()

 //
 // NOTE
 //
 // For a long running call heartbeats can be recorded in a separate
 //
 // go func () {
 //     for {
 //         activity.RecordHeartbeat(ctx, "Activity is running")
 //         // TODO sleep for some interval
 //     }
 // } ()
 //

 for {
  select {
  case <-cancel:

   //
   // NOTE
   //
   // Execute any cleanup required for a canceled activity here...
   //

   return nil, errors.New("Activity was canceled")
  case <-done:
   return res, err
  }
 }
}
```

Step 3 Edit the file to return a message:

```
go func() {

 res = &Response {Message: "Hello, " + req.GetName() + "!"}
 err = nil

 done <- 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(Request) returns (Response);
 rpc Fancy(Request) returns (Response);
}
```

Step 2 Update the activities go code using the SDK:

```
make generate-code
```

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. Open your `cwmsdk` repository directory in a terminal and run:

```
cwm-sdk extend-adapter -feature featureY
```

Step 1 A new `companyX.productX.featureY.adapter.proto` file has been added for your adapter:

```
.proto/
  cisco.cwm.sdk.resource.proto
  companyX.productX.common.adapter.proto
  companyX.productX.featureY.adapter.proto
  companyX.productX.featureX.adapter.proto
```

Step 2 To define activities for the new feature, open the `companyX.productX.featureY.adapter.proto` file, and modify the contents accordingly

```
syntax = "proto3";

package companyXproductX;

option go_package = "www.cisco.com/cwm/adapters/companyX/productX/featureY";

service Activities {
 /* Documentation for Goodbye Activity */
 rpc Goodbye(Request) returns (Response);
}
```

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

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

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

```
make generate-model && generate-code && ls
```

```
.go/goodbyes
 activities.go
 adapter.go
 companyX.productX.featureY.adapter.pb.go
```

Step 5: Create an installable archive

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

The generated archive contains the all required files of the adapter. The **go** vendor command has been executed in order to eliminate any external dependencies.