Automated Test Framework

User Manual

Dmitry Chmerev

March 26, 2015

Contents

1 ATF basics

ATF is a set of Lua classes and extensions providing a simple way to create, manage and use network connections, process Applink protocol messages and use HMI protocol — everything needed for communication with SDL. It's generally a library for black-box test development for SDL.

Test consists of several use cases — a set of data to be sent to input SDL channels and data expected to be received from SDL output.

2 Test script structure

Because ATF test runner is merely a Lua interpreter, test script is just a Lua program. The script should create a Test table, using one of base scripts, add some test cases and return this table to allow another test script to extend this test with new cases. Test cases should be implemented as functions with only argument — Test table. Test case function name must start with a capital letter. Basically a test case fills expectations table (see 3) and sends a fixed message to SDL. When SDL responds, expectations are verified, and if all expectations have been satisfied, the test case is considered passed.

The base test creates two connection to SDL — mobile connection and HMI connection, and initializes them. Then an application is registered using `RegisterAppInterface` call.

3 Expectations creating

To specify what data is expected to be received from SDL, the Expectations Table is used. Every item of this table describes single message. To simplify Expectations writing ATF provides several functions making typical expectations.

3.1 `EXPECT_RESPONSE(correlationId, arguments, ...)`

Returns expectation of mobile connection input. This function takes following arguments: the first, `correlationId`, is the correlation identifier of response expected. The latter ones, `arguments`, are optional tables of expected response arguments. If expectation has complied more than once, arguments are checked against last arguments parameter.

3.2 `EXPECT_NOTIFICATION(funcName, arguments, ...)`

Returns expectation of mobile connection notification. The function takes two arguments: `funcName`, name of notification, and optional `arguments`, which are used to verify notification data.

3.3 EXPECT_ANY

Returns expectation of any unexpected activity on mobile session.

3.4 EXPECT_ANY_SESSION_NOTIFICATION(funcName, arguments, ...)

Returns expectation of notification to any session.

3.5 EXPECT_EVENT(event, name)

Returns expectation of custom event

3.6 EXPECT_HMIRESPONSE

Returns expectation of HMI connection data. The only argument, `id`, is identifier of request sent.

3.7 EXPECT_HMINOTIFICATION

Returns expectation of an HMI notification from SDL.

4 Expectations extensions

Expectations may be extended with some extra specifications. They are intended to be used in call chain: programmer can improve expectation parameters by calling following methods one after one.
Example:

```
EXPECT_NOTIFICATION("OnHMIStatus")
  :Timeout(5000)
  :Times(AnyNumber())
  :Do(function() print("OnHMIStatus got") end)
```

4.1 Times

If you want an expectation to occur several times, add `Times` call to expectation.
Example:

```
EXPECT_NOTIFICATION("OnHMIStatus")
  :Times(2)
```

`Times()` arguments may be one of following values:

- x or Exactly(x), to specify that expectation should occur exactly x times to get passed. Particularly, `Times(0)` means that expectation will fail test if occur

- AtMost(x) — expectation should occur x or fewer times
- AtLeast(x) — expectation should occur x or more times
- Between(x, y) — expectation should occur between x and y times
- AnyNumber() — expectation may occur or not

4.2 `Do, DoOnce`

These two functions takes one argument — function that will be called when expectation complied. This callback function will be called with two arguments: complied expectation and incoming data. `Do` function registers callback forever, `DoOnce` — for the first trigger only.

4.3 `Timeout`

Default expectation timeout is 10 000 ms. To change it, call `Timeout` function with new timeout value in milliseconds.

4.4 `Pin, Unpin`

Pinned expectation is not to be removed when test case is finished. This ability is used generally for some service expectations (timeout controllers, notification handlers etc.) To make the expectation pinned, call `Pin()` extension. To unpin it, call `Unpin()`.

4.5 `ValidIf`

In some cases the standard RPC validation routine is not enough to verify expectation data. Use `ValidIf` function to register verifying callback that would be used when expectation complied. Callback should take two arguments: expectation object and data table. If data is valid, function must return `true`, otherwise it must return two values: `false` and error message to be displayed in test report.

5 Requests sending

The connectivity test provides two connections to SDL channels: `mobileSession` and `hmiConnection`.

5.1 Mobile side

To send a request from mobile application side, call `mobileSession:SendRPC(message, filename)`. This function returns `correlationId` field of request has been sent. `filename` is optional. If is not `nil`, the file contents added to RPC request binary data. For example,

```
self.mobileSession:SendRPC("Alert", { alertText1 = "Hello" })
self.mobileSession:SendRPC("PutFile",
                           {
                             syncFileName = "icon.png",
                             fileType = "GRAPHIC_PNG"
                           },
                           "icon.png")
```

To send an arbitrary message, use `Send` method. It takes the Applink message, the following fields are supported:

- version (optional, default 2) Version of message
- encryption (optional, default false) Encryption flag
- frameType (optional, default 1) Frame Type field
- serviceType (required) Sevice Type field
- frameInfo (required) Frame Info field
- rpcType (required) If serviceType is 7, type of RPC request
- rpcFunctionId (required) RPC function id
- rpcCorrelationId (required) RPC correlation id
- payload (required) JSON payload of RPC request
- binaryData (optional) binary data of message or RPC request

5.2 HMI side

`HMIConnection` provides functions similar to `MobileSession`'s.

- `SendRequest(method, params)` Sends HMI request, returns id of sent message.
- `SendNotification(method, params)` Sends notification with parameters.
- `SendResponse(id, method, result, params)` Sends response to request with specified id, method name. `result` field is a name of result code. For example, `"SUCCESS"` or `"ABORTED"`.
- `Send(text)` Sends arbitrary text through HMI connection.

To send a message from HMI side, use `hmiConnection:Send(message)`. We will add some more convenient methods later. For example,

```
self.hmiConnection:Send({
                            jsonrpc = "2.0",
                            method = "UI.OnSystemContext",
                            params =
                            {
                              appID = 100,
                              systemContext = "ALERT"
                            }
                          })
```

6 Services

To start a service use `MobileSession:StartService(n)` function. It takes service number and returns expectation awating for `StartServiceACK` message. To end service, call `MobileSession:StopService(n)`.

Example:

```
self.mobileSession:StartService(11)
  :Do(function() print("Service started") end)
```

7 Media streaming

ATF allows to start media streaming in background. Use `MobileSession:StartStreaming(service, filename, bandwidth)` to start media streaming. Optional `bandwidth` argument sets maximum bandwidth of streaming (bytes per second). Default is 30 kbps. Note that the service should be started before `StartStreaming` is called.

Example:

```
self.mobileSession:StartService(10)
  :Do(function() self.mobileSession:StartStreaming(10, "video.mpg", 30 * 102
```

To stop file streaming, call `StopStreaming(filename)`.

8 Auxiliary functions

8.1 RAISE_EVENT(event, data)

Raises custom event with optional data. If there is an expectation it will be satisfied.

8.2 RUN_AFTER

Use `RUN_AFTER` routine to postpone function execution.
Usage: `RUN_AFTER(function, timeout)`. The function will be called in `timeout` msec.