

# Chordant oneM2M AE Certification Contribution to ATIS OS-IoT

oneM2M Compliant Upper Tester and Reference AT  
Command Test Interface

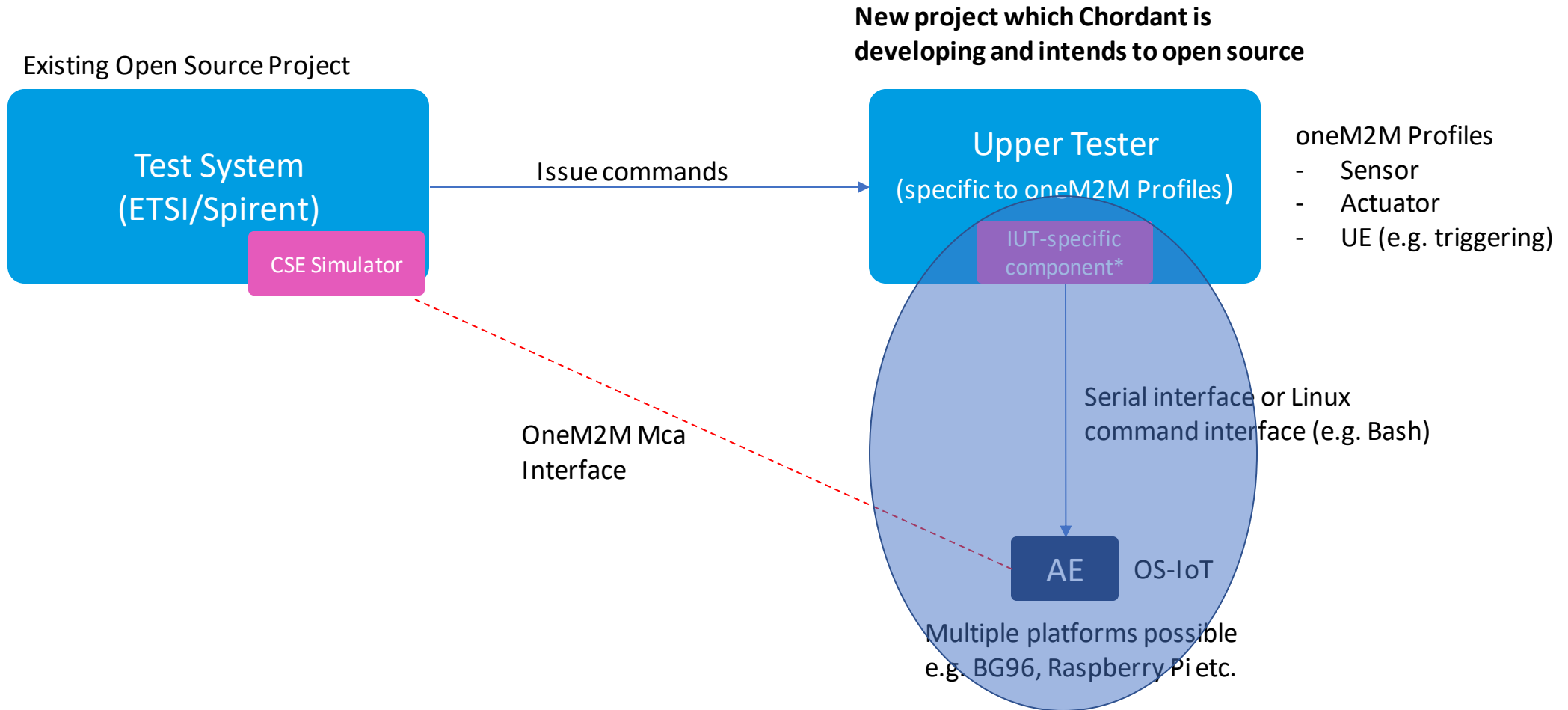




## Context – provide open-source testing tools for AE developers to enhance oneM2M certification

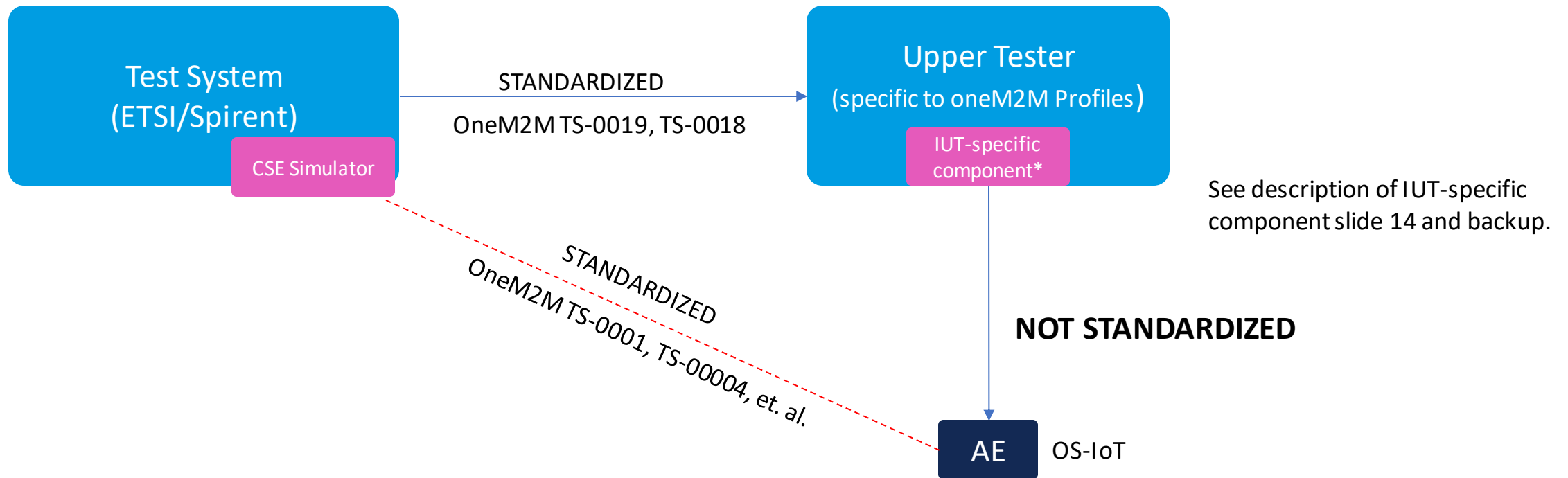
- oneM2M seeks to expand certification of devices (AEs), an under addressed topic in comparison to platform (CSE) certification
- The target audience is AE developers who will benefit from open-source support (code, libraries, tools) to make their AEs testable, a pre-cursor to certification.
- oneM2M has done some preliminary work by:
  - defining a testing framework and message structures
  - defining a few AE certification profiles. With input from Chordant, the plan is to cover additional AE host devices (e.g. cellular based devices supporting features such as device triggering, etc.)
- The purpose of this discussion is to explore an approach to build on ATIS' OS-IoT foundational work

# Testing Framework



\* Implementation Under Test (IUT)

# Testing Framework – Standardized Interfaces View

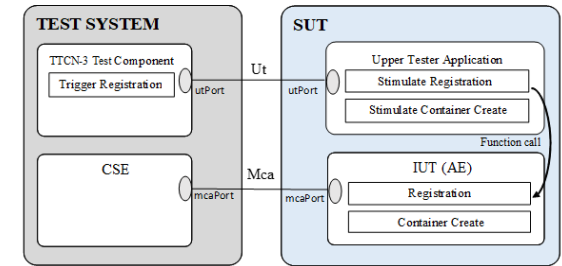


\* Implementation Under Test (IUT)

# Reference AT Command Test Interface



# AEs and Certification Processes



- AEs are typically resource-constrained devices.
- The interface between the Upper Tester and the AE is not standardized
- The Ut interface between the Test System and the SUT is standardized (TS-0019) but is not mandatory to implement by the AE/device
- The AE implementation is not standardized, only its interface to the CSE.
- An AE going through certification should have the following properties:
  - **Properly chosen certification profile** – match the AEs normal oneM2M capabilities with the certification profile
  - **Minimized complexity** – the test interface kept minimal in complexity and appropriate to the device’s capabilities, to keep the device footprint small.
  - **Design for test** – leverage operational code as much as possible to support the required profile test functions. The test interface should be an interface, and nothing more.
  - **Binary-same** – the AE that is certified should be binary-same to that deployed in the field. This means we don’t conditionally link code that is “for test only.”

## Choosing the Device Profile

- From the oneM2M viewpoint, the ATIS AE does three things: create AE, create container, create content instances
- This aligns well with ADN Profile #3 of oneM2M [[TS-0025](#)]

**Table 5.4.3-1: Fundamental feature set for ADN profile 3**

| Function | Feature Set  | Feature            | Remark   |
|----------|--------------|--------------------|--|
| GEN      | AE/GEN/00001 | At least one       | Support one of the format of resource identification |
|          | AE/GEN/00002 | AE/GEN/00002/00001 | Support Create request targeting one resource        |
| REG      | AE/REG/00002 | AE/REG/00002/00001 | Create <AE> with mandatory attributes                |
| DMR      | AE/DMR/00001 | AE/DMR/00001/00001 | Create <container> with no attribute set             |
|          | AE/DMR/00002 | AE/DMR/00002/00001 | Create <contentInstance> with mandatory attributes   |

# Choosing the Test Interface

- The BG96 is not a highly constrained device, but it is a useful reference platform for cellular IoT. So we try to adhere to the constrained-device “shoulds” as a matter of best practice, not necessity.
- The BG96 utilizes the Qualcomm QAPI, which has a set of APIs for AT Command Forwarding -- i.e. an application can register AT custom commands with the QAPI, and those commands and their arguments are forwarded to the application via a callback function.
- This is the chosen method to implement the upper tester interface to the ATIS BG96 AE, custom AT command over the BG96 AT Command COM Port.





# Design for Test

- To support design for test, the AE was refactored such that:
- there is a function interface for creating AEs, creating containers, and creating content instances. This interface is used by both the operational mode AE and the test interface in response to AT commands.
- the main loop modified such that it can receive an additional ThreadX event for AT commands that are forwarded by the QAPI
- a new module added to support the AT command interface



# Code Review

- Ref. branch: atcmd in repo atis-os-iot-bg96
- New files
  - onem2m\_at\_cmds.c
  - onem2m\_at\_cmds.h
  - http.h
- Modified file
  - http.c
- An overview of the code changes is in the backup slides.

# oneM2M Upper Tester





# Interface to Test System and ATIS AE

- The Upper Tester implements the utPort as defined in TS-0019 to communicate with the Test System
  - Protocol: HTTP
  - Content Type: JSON
- The Upper Tester contains an IUT-specific component that implements a serial port interface to communicate with the ATIS AE
  - Currently supports Windows only
- The Upper Tester supports ADN Profile #3 of oneM2M [TS-0025]

# Implementation

- The Upper Tester is a Spring Boot app built with maven
  - Java 1.8
  - Spring Boot v2.1.2.RELEASE
  - Spring v5.1.4.RELEASE
  - Maven 3.6
- The Upper Tester was developed/tested/debugged using Spring Tool Suite 3.9.6.RELEASE
- Use Maven to build a runnable jar file
  - `mvn clean package -Dmaven.test.skip=true`



## Code – IUTServiceATISAEImpl.java

- IUT-specific component for the ATIS AE
- Implements IUTService.java
- Marked with profile "atisae"
- Load this component at runtime by specifying the "atisae" profile
  - `java -jar <jar file> --spring.profiles.active=atisae`

# Demo

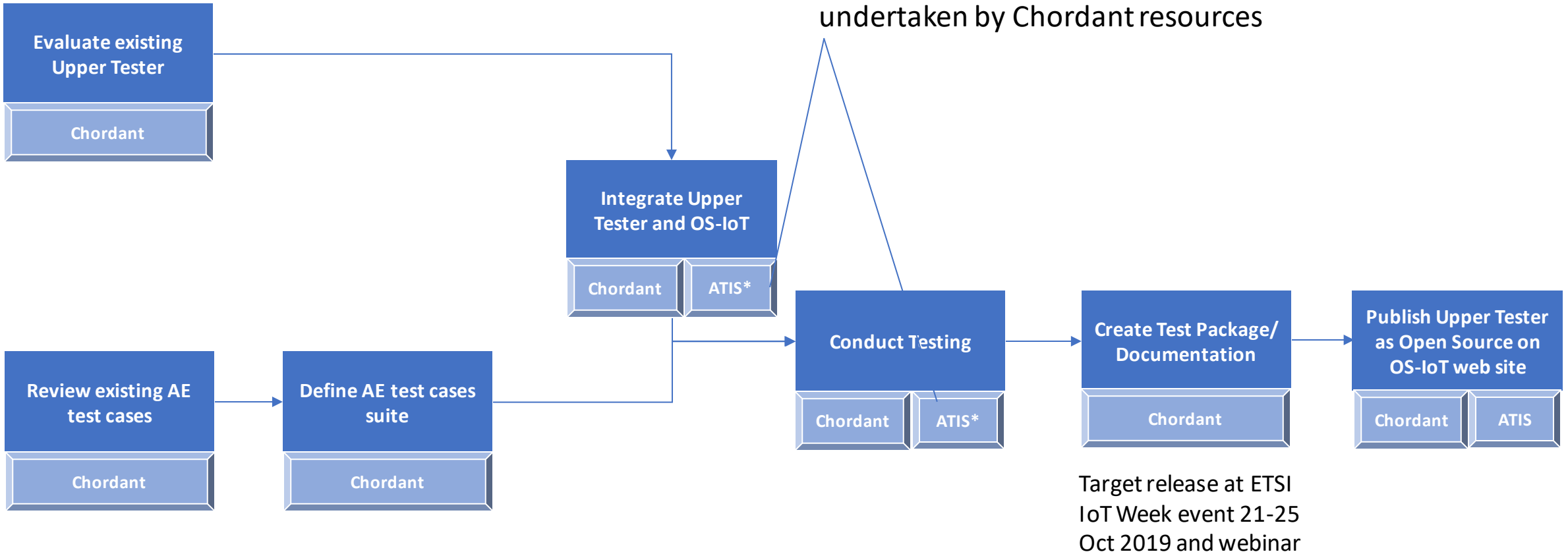


# Backup





# Workplan and roles



## Upper Tester Code – TriggerPrimitiveController.java

```
@PostMapping("/")
public ResponseEntity<String> triggerPrimitive(@RequestBody String body)
{
    // receive HTTP POST from Test System
    // validate trigger primitive
    // if validation fails
    //     send BAD_REQUEST to test system
    // else
    //     send trigger primitive to IUT-specific component
    //     send OK to test system
}
```



## Upper Tester Code – IUTService.java

- This interface will be implemented by each IUT-specific component
- Each IUT-specific component must be marked with a unique profile so it can be loaded at runtime according to the active profiles

```
public interface IUTService {  
    // Setup the IUT  
    public void setupIUT() throws Exception;  
  
    // Tear down the IUT  
    public void teardownIUT() throws Exception;  
  
    // Send trigger primitive to the IUT  
    public void sendTriggerPrimitiveToIUT(String triggerPrimitive) throws  
Exception;  
}
```



## Upper Tester Code – IUTServiceATISAEImpl.java

```
@Service
@Profile("atisae")
public class IUTServiceATISAEImpl implements IUTService {
    public void setupIUT() throws Exception {
        // Get the AT serial port and configure it
        // Setup the ATIS AE for testing
    }

    public void teardownIUT() throws Exception {
        // Put the ATIS AE back to normal mode
    }

    public void sendTriggerPrimitiveToIUT(String triggerPrimitive) throws
Exception {
        // Create custom AE command based on trigger primitive
        // Send AT command to ATIS AE
    }
}
```



## AE Code – http.c

Refactor of `http_request_using_state()`, to create an interface for http requests that the test interface can also use.

```
qapi_Status_t http_request_using_state() {
    qapi_Status_t status;
    char content_string [MAX_CONTENT_LEN];

    switch (op_state) {
        case create_ae:
            status = http_request_create_ae(json_object_get_string(config_object, SERVER_PATH_KEY));
            break;

        case create_container:
            status = http_request_create_container(
                json_object_get_string(config_object, SERVER_PATH_KEY),
                json_object_get_string(config_object, CONTAINER_RESOURCE_NAME_KEY));
            break;

        case reporting:
            get_content_string(content_string);
            status = http_request_create_content_inst(
                json_object_get_string(config_object, SERVER_PATH_KEY),
                json_object_get_string(config_object, CONTAINER_RESOURCE_NAME_KEY),
                content_string);
            break;
    }
}
```



## AE Code – http.c

```
tx_status = tx_event_flags_get(events_p, EVENT_CALL_DISCONNECT | EVENT_AT_COMMAND,  
TX_OR_CLEAR, &res_events, REQUEST_INTERVAL*TX_TIMER_TICKS_PER_SECOND) ;
```

In the AE main loop:

- removed qapi\_sleep() call in favor of a wait-for-events-with-timeout, such that the AE can respond to asynchronous AT command events.

- added a flag to enable/disable normal main loop processing via the test interface. (not shown here)



## AE Code – http.h

```
qapi_Status_t http_request_create_ae(char *server_path_key);  
qapi_Status_t http_request_create_container(char *server_path_key, char  
*container_res_name_key);  
qapi_Status_t http_request_create_content_inst(char *server_path_key, char  
*container_res_name_key, char *ci_string);  
  
qapi_Status_t http_connect_with_retry(const char * host, uint16_t port);  
void http_disconnect(void);
```

This header file added to expose interfaces to functions needed by the test interface: (a) the http request functions, and (b) the connect() and disconnect() functions required pre and post test request.



## AE Code – onem2m\_at\_cmds

```
qapi_Status_t register_onem2m_at_cmd(TX_EVENT_FLAGS_GROUP * events_p)
{
    // store events_p so the AT callback can signal the main thread
    // allocate byte pool for use by the AT callback function
    // qapi_atfwd_Pass_Pool_Ptr(atfwd_cmd_handler_cb, byte_pool_at);
    // qapi_atfwd_reg(AT_COMMAND_STR, atfwd_cmd_handler_cb);
}
```

Called by the main thread to initialize AT command forwarding for the AT\_COMMAND\_STR "ONEM2M"





## AE Code – onem2m\_at\_cmds

```
void atfwd_cmd_handler_cb(boolean is_reg, char *atcmd_name,  
    uint8* at_fwd_params, uint8 mask, uint32 at_handle)  
{  
    // store AT command parameter pointer  
    // signal EVENT_AT_COMMAND to the main thread  
}
```

Callback from the QAPI AT forwarding,  
Called when the registered ONEM2M  
command is received.

```
int parse_at_command()  
{  
    // parse AT command and its parameters  
    // call the appropriate http request function (http.h)  
}
```

Called from main thread when signaled  
by the callback. other sub-functions not shown