AHEAD OF WHAT'S POSSIBLE™

Machine to Machine Messaging V.2 (M2M2) System Introduction

1/18/2021

Analog Devices Confidential Information—Not for External Distribution

M2M2 - Basic Idea

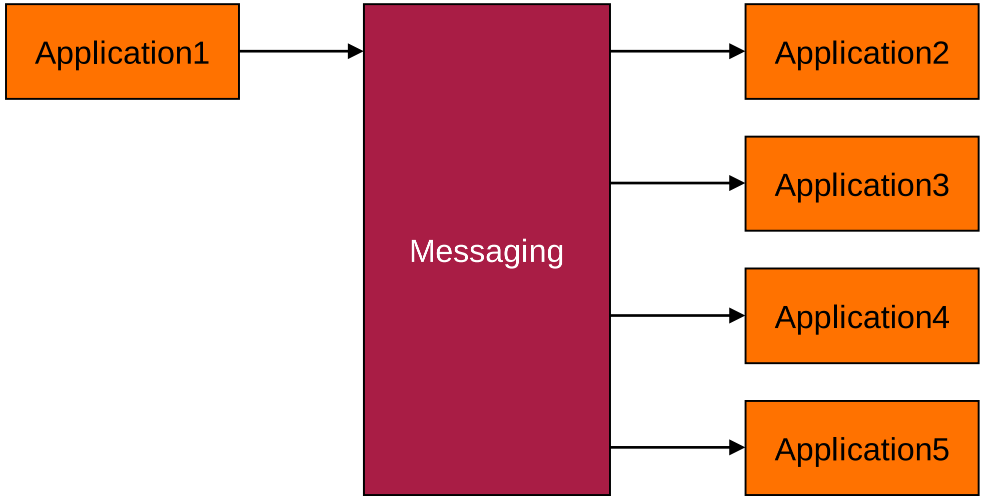
- Instead of writing a monolithic firmware that's very specialized to the exact hardware it runs on, we break it up into separate *applications* that communicate with *messages* and decouple software from board-level changes.
- This is <u>not</u> just a hardware abstraction layer (HAL), or a set of packets. It's an ecosystem of tools, processes, applications, and design patterns.
- Study Watch is <u>built on</u> M2M2; they are not the same thing.

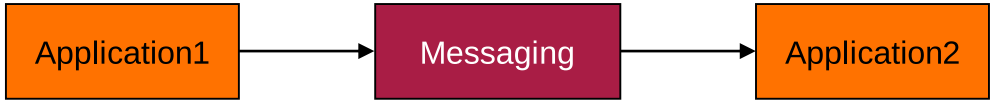
<u>Problems to solve:</u>

- Firmware can quickly get very complex and inter-dependent
  - If a new algorithm shows up that takes 5s to run; can your main loop accommodate it?
- It's very hard to re-use software that's very closely tied to hardware *<u>and to other software</u>*
  - A sensor_read() function is usually closely tied into hardware (i.e. MCU drivers) and to other software functions (i.e. an algorithm)
- It's difficult to re-route data within the system if it needs to go somewhere else
  - How do we get data from sensor_read() on processor1 to process_data() on processor2?
    - What if the data needs to get to log_data() on processor1, process_data() on processor2, graph_data() on a mobile phone, and plot_data() on a PC?
  - How do we re-use data routing functionality?
- How do we decouple embedded software from the hardware *and the rest of the software*?

M2M2 Architecture - Introduction

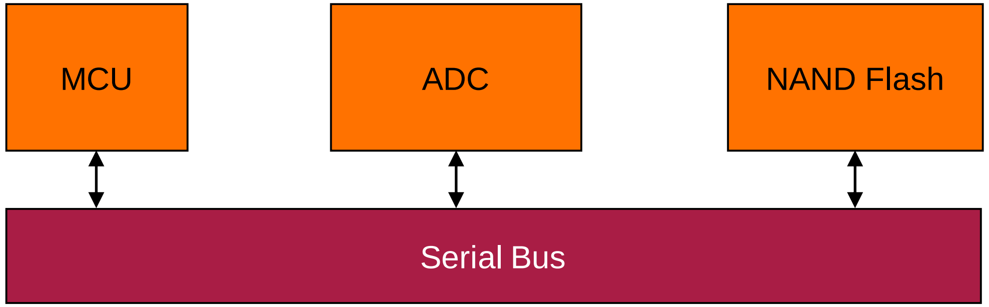
- M2M2 was designed to enable embedded systems to use the “micro-kernel” and “micro-services” system design architectures with minimal overhead and footprint
  - System is broken up into well-defined applications
  - Applications are designed and implemented to be completely independent of each other
  - Applications interact by passing messages
    - Point-to-point messages
    - Publish-subscribe messages
  - Applications can be easily moved between processors
  - System features can easily be added, removed and shared

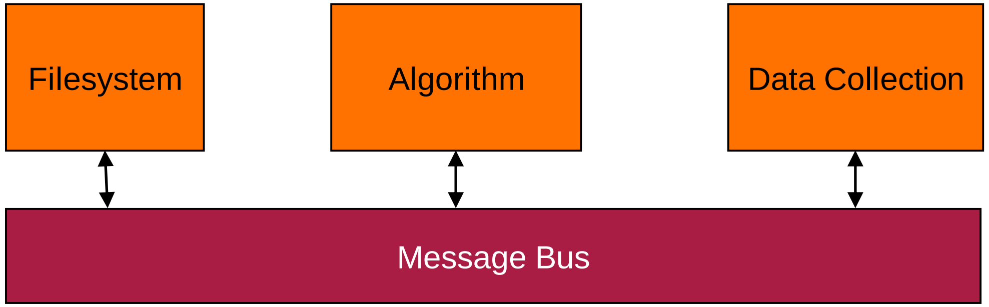




M2M2 Architecture - Comparison to Digital Serial Bus

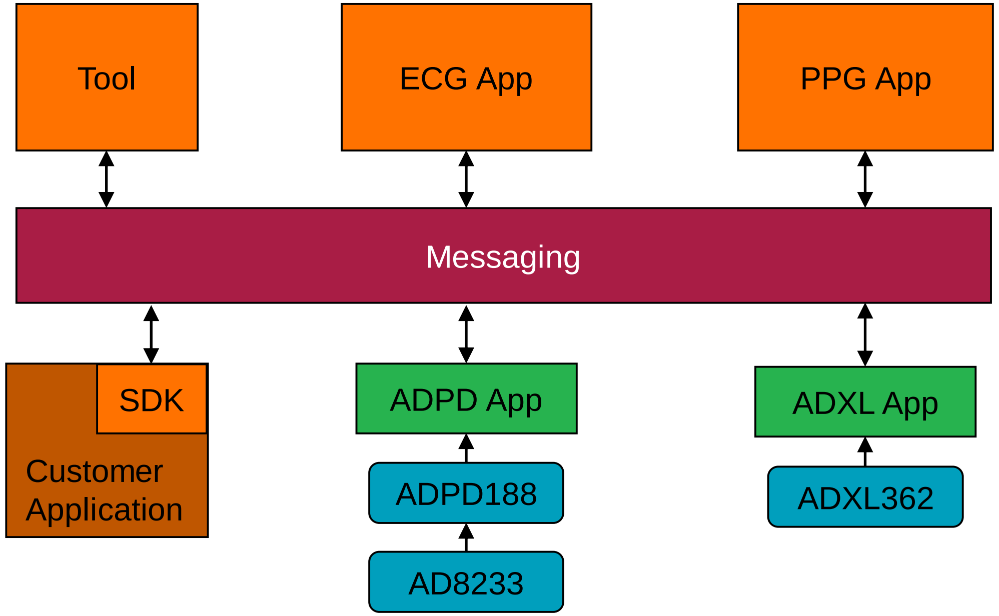
- The M2M2 message bus is similar in concept to a digital serial bus
- Similarities:
  - Devices/Applications can be easily added and removed from the bus
  - Devices/Applications are used with a set of well-defined commands
  - The bus can span multiple PCBs or processors
- Differences:
  - More than one application can use the M2M2 message bus simultaneously
  - The M2M2 message bus provides more advanced addressing features (mailboxes)





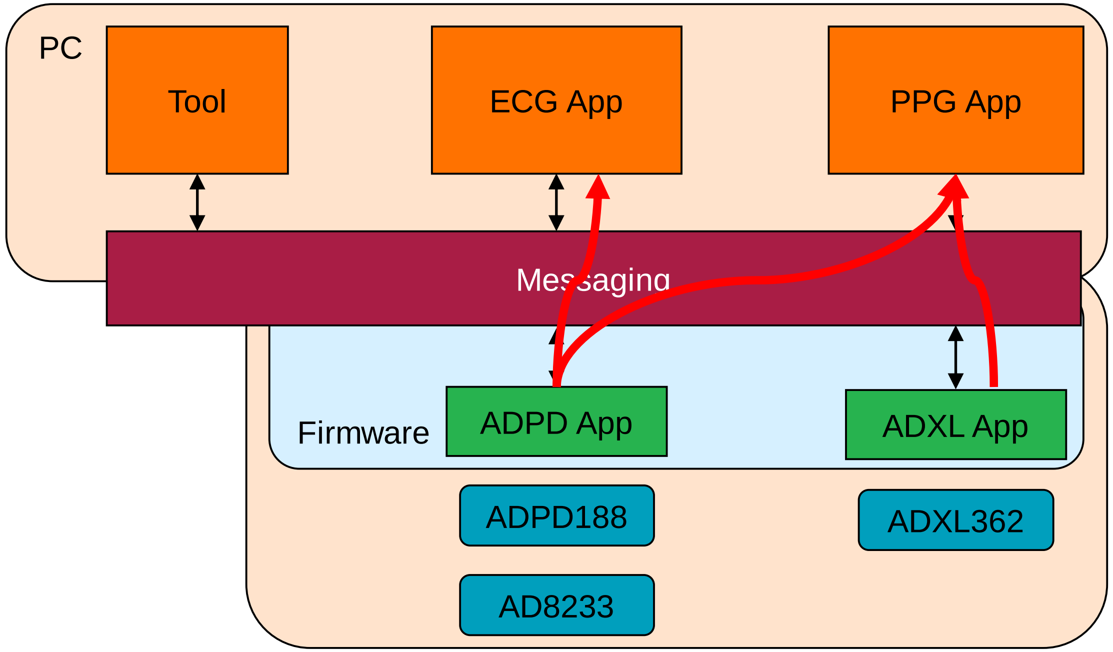
M2M2 Architecture - Example

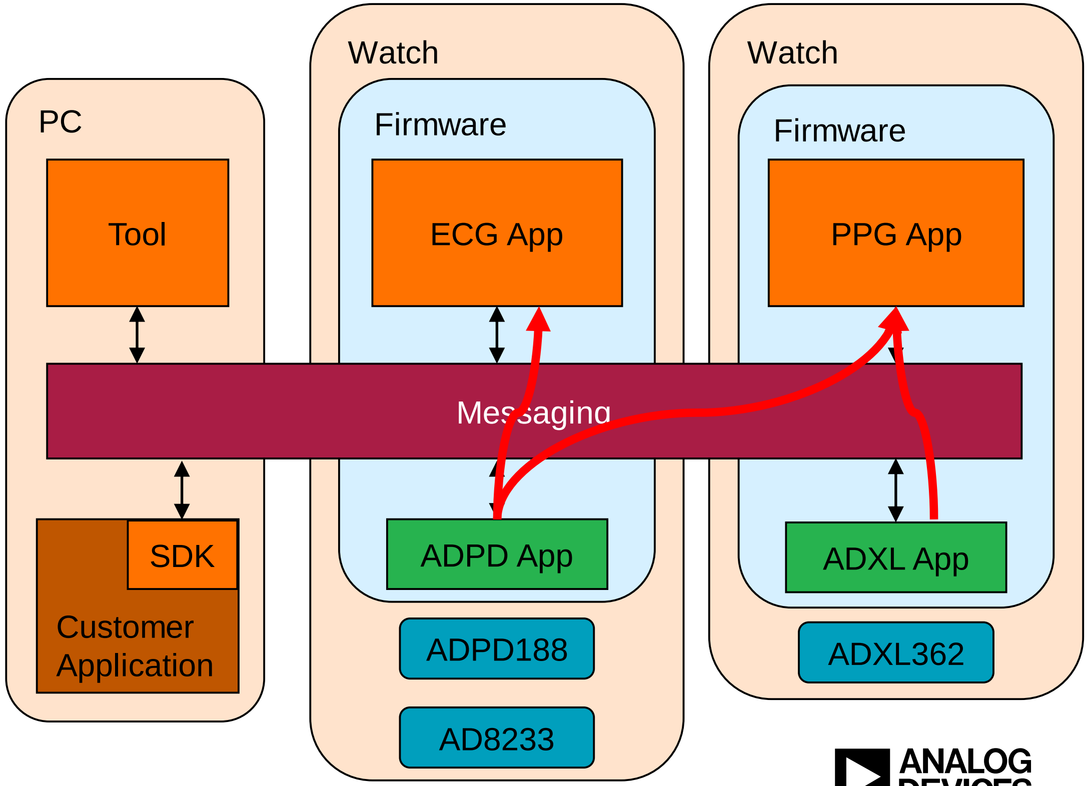
- Sensors
  - Not accessed directly by data consumers
- Applications
  - Group logical sets of functionality
    - A particular sensor (i.e. ADPD, ADXL)
    - A particular measurement (i.e. ECG, PPG)
    - "Real" (Manage a piece of HW) or "Abstract" (pure software)
    - A tool
- Messaging
  - Post Office
    - Packet Routing
  - Post Office::Mailbox
    - Broadcast messaging
    - Publish-Subscribe messaging



M2M2 Architecture - Example

- System is easy to reconfigure – just place different applications on different processors
  - Embedded processors or PC/cloud processors
  - Message routing still works as expected
  - Data producers/consumers don’t need to be updated with the new system layout





M2M2 - Pros and Cons

Pros

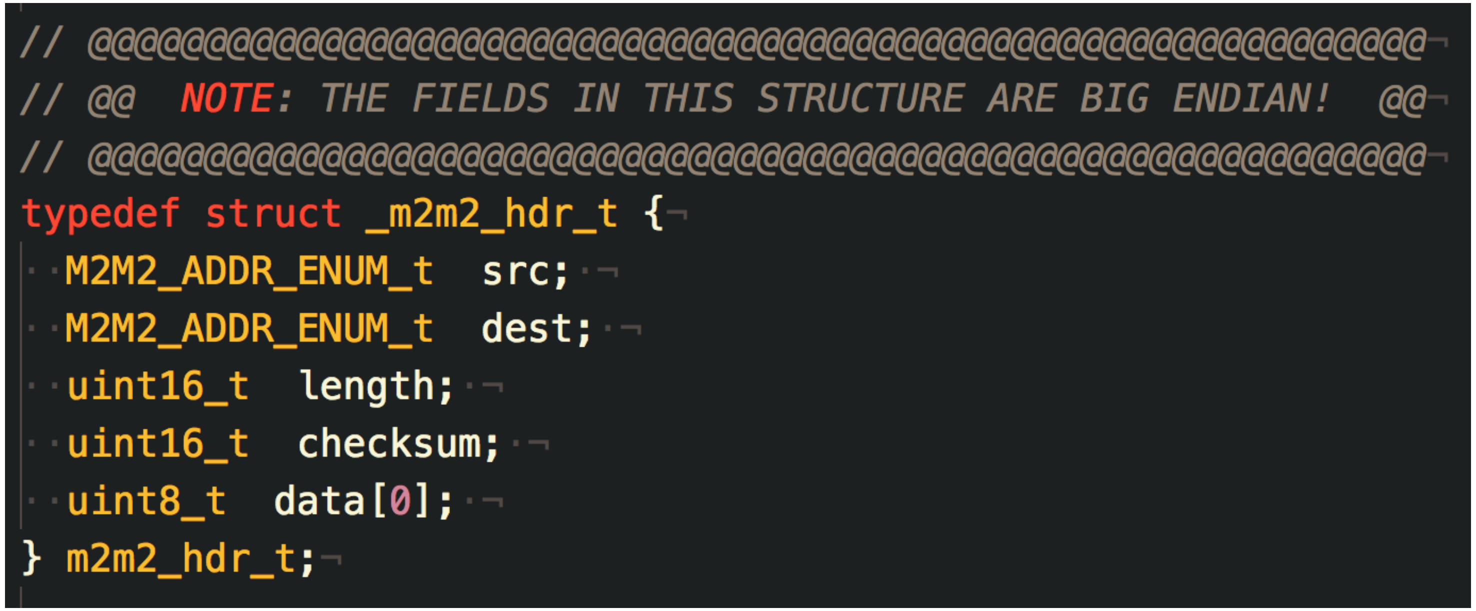
- Applications are loosely coupled
  - Easy to move them between processors
  - Easier to implement complex system behaviours and data flows
- Applications are easier to think about
  - Changes only require local knowledge
- Software re-use is easier
  - New hardware variants can use identical applications with minimal changes
- Flexible data flow
  - Producers don't need to know about consumers

Cons

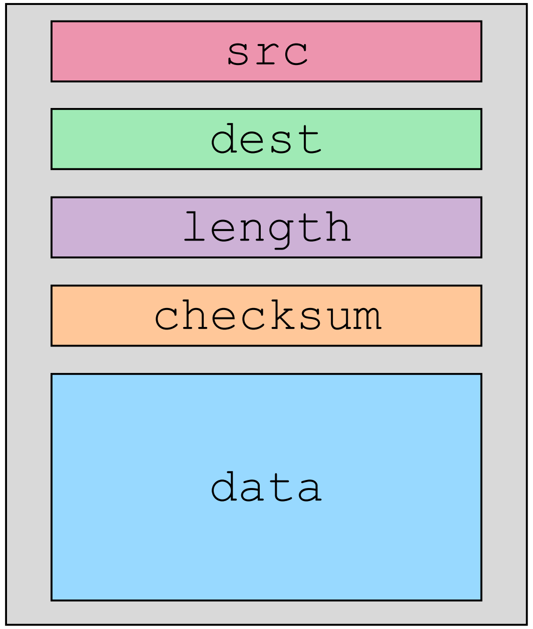
- Small run-time overhead
  - O(n) lookup for message sends (n = # of applications in system)
  - Fixed 512 bytes for entire Post Office task
  - Variable 6 bytes RAM for each application
  - Variable 6 + 2n bytes RAM for each mailbox (n = # of subscribers to the mailbox)
- Initial application implementation is a bit more involved for a first time user

M2M2 Message Passing - Header Structure

- 8-byte UDP-style header
  - Does NOT use a UDP stack
- Variable length data field
- Applications define their message own payloads

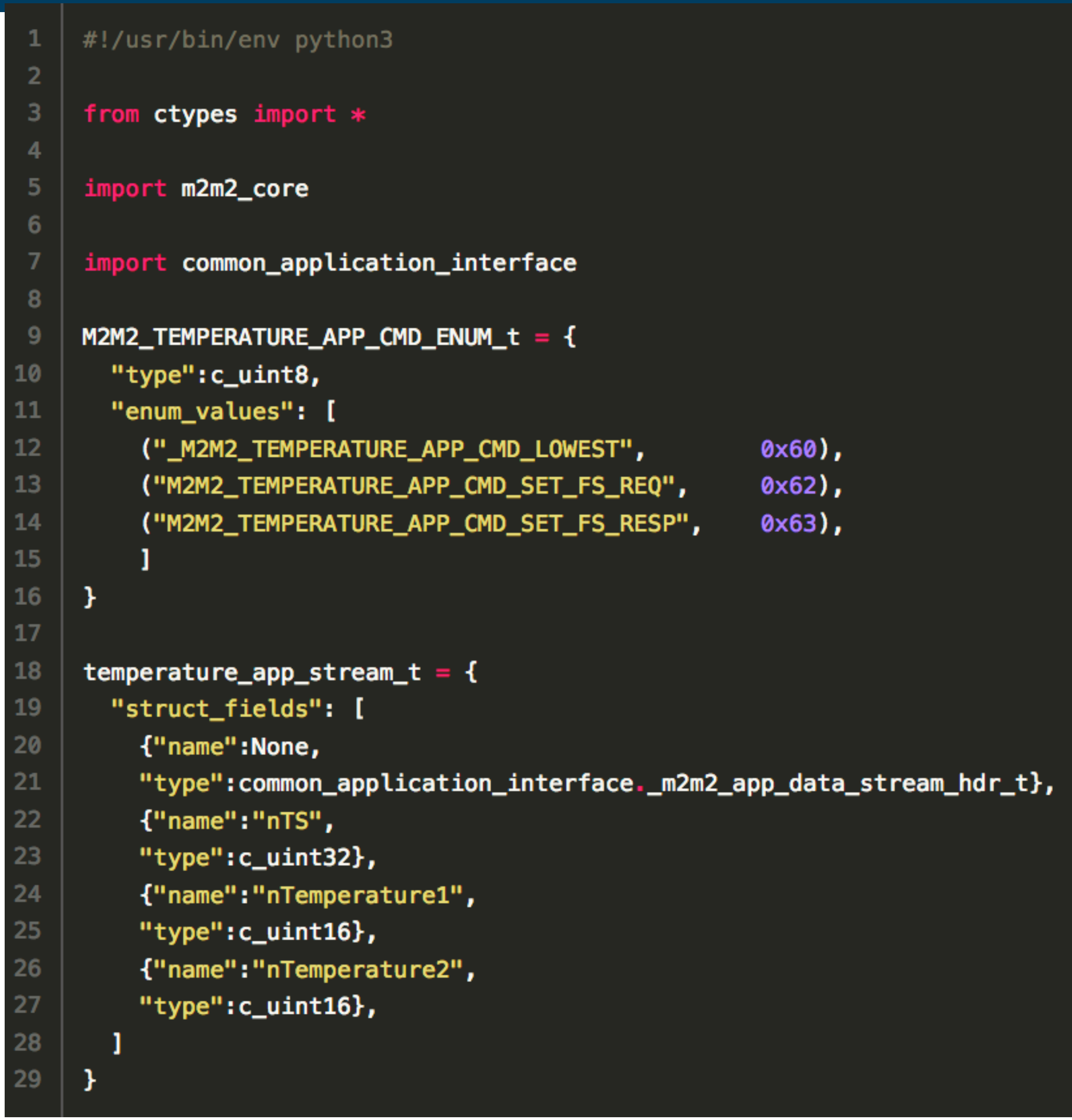


```c
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
// @@  NOTE: THE FIELDS IN THIS STRUCTURE ARE BIG ENDIAN!  @@
// @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
typedef struct _m2m2_hdr_t {
  M2M2_ADDR_ENUM_t  src;
  M2M2_ADDR_ENUM_t  dest;
  uint16_t  length;
  uint16_t  checksum;
  uint8_t  data[0];
} m2m2_hdr_t;
```



M2M2 Message Passing - Application Interfaces

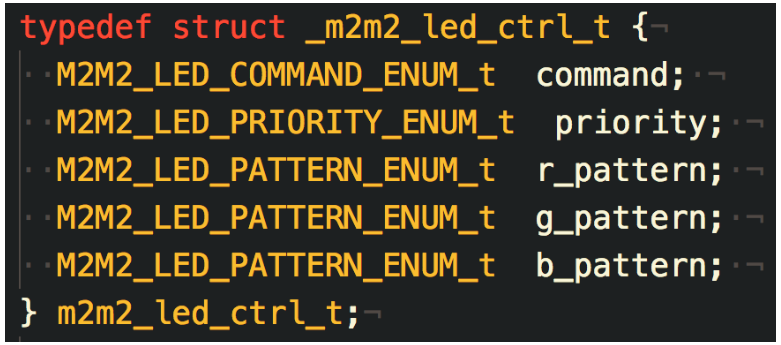
- Each application has a [application_name]_interface.py file
  - Interface contains all structs and enums used to interact with an application
  - Interfaces are defined in Python
    - Allows language-specific definitions to be easily generated
  - Provides a single place to update the interfaces and have them propagate through the entire system (and tools!)
- Some basic inheritance is used
  - common_application_interface
  - common_sensor_interface
- Interface generator supports user-defined types, variable-length fields, and anonymous structs



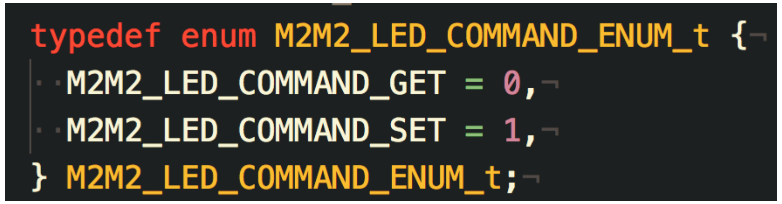
```
#!/usr/bin/env python3

from ctypes import *

import m2m2_core

import common_application_interface

M2M2_TEMPERATURE_APP_CMD_ENUM_t = {
  "type":c_uint8,
  "enum_values": [
    ("_M2M2_TEMPERATURE_APP_CMD_LOWEST",        0x60),
    ("M2M2_TEMPERATURE_APP_CMD_SET_FS_REQ",     0x62),
    ("M2M2_TEMPERATURE_APP_CMD_SET_FS_RESP",    0x63),
    ]
}

temperature_app_stream_t = {
  "struct_fields": [
    {"name":None,
    "type":common_application_interface._m2m2_app_data_stream_hdr_t},
    {"name":"nTS",
    "type":c_uint32},
    {"name":"nTemperature1",
    "type":c_uint16},
    {"name":"nTemperature2",
    "type":c_uint16},
  ]
}
```

M2M2 Message Passing - Application Interface Definitions

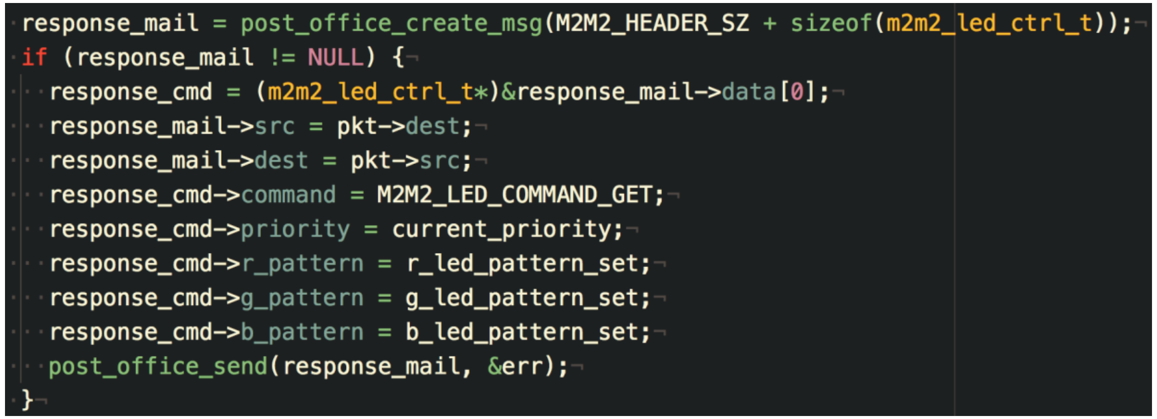
Python definitions and C/C++ structs are generated by the *python_data_generator.py* script



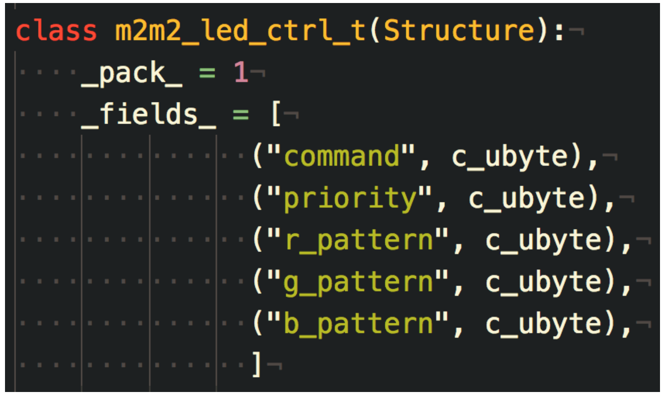
```c
typedef struct _m2m2_led_ctrl_t {
  M2M2_LED_COMMAND_ENUM_t  command;
  M2M2_LED_PRIORITY_ENUM_t  priority;
  M2M2_LED_PATTERN_ENUM_t  r_pattern;
  M2M2_LED_PATTERN_ENUM_t  g_pattern;
  M2M2_LED_PATTERN_ENUM_t  b_pattern;
} m2m2_led_ctrl_t;
```



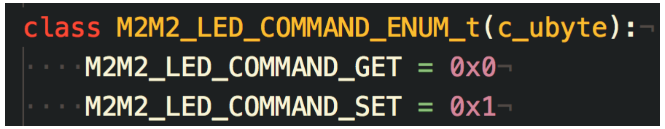
```c
typedef enum M2M2_LED_COMMAND_ENUM_t {
  M2M2_LED_COMMAND_GET = 0,
  M2M2_LED_COMMAND_SET = 1,
} M2M2_LED_COMMAND_ENUM_t;
```



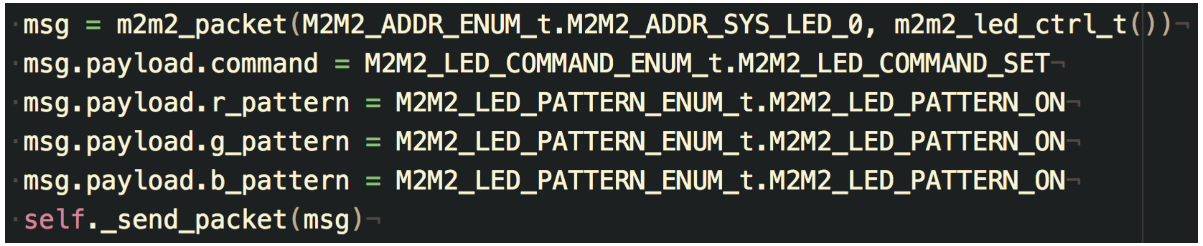
```c
response_mail = post_office_create_msg(M2M2_HEADER_SZ + sizeof(m2m2_led_ctrl_t));
if (response_mail != NULL) {
  response_cmd = (m2m2_led_ctrl_t*)&response_mail->data[0];
  response_mail->src = pkt->dest;
  response_mail->dest = pkt->src;
  response_cmd->command = M2M2_LED_COMMAND_GET;
  response_cmd->priority = current_priority;
  response_cmd->r_pattern = r_led_pattern_set;
  response_cmd->g_pattern = g_led_pattern_set;
  response_cmd->b_pattern = b_led_pattern_set;
  post_office_send(response_mail, &err);
}
```



```python
class m2m2_led_ctrl_t(Structure):
    _pack_ = 1
    _fields_ = [
              ("command", c_ubyte),
              ("priority", c_ubyte),
              ("r_pattern", c_ubyte),
              ("g_pattern", c_ubyte),
              ("b_pattern", c_ubyte),
              ]
```



```python
class M2M2_LED_COMMAND_ENUM_t(c_ubyte):
    M2M2_LED_COMMAND_GET = 0x0
    M2M2_LED_COMMAND_SET = 0x1
```



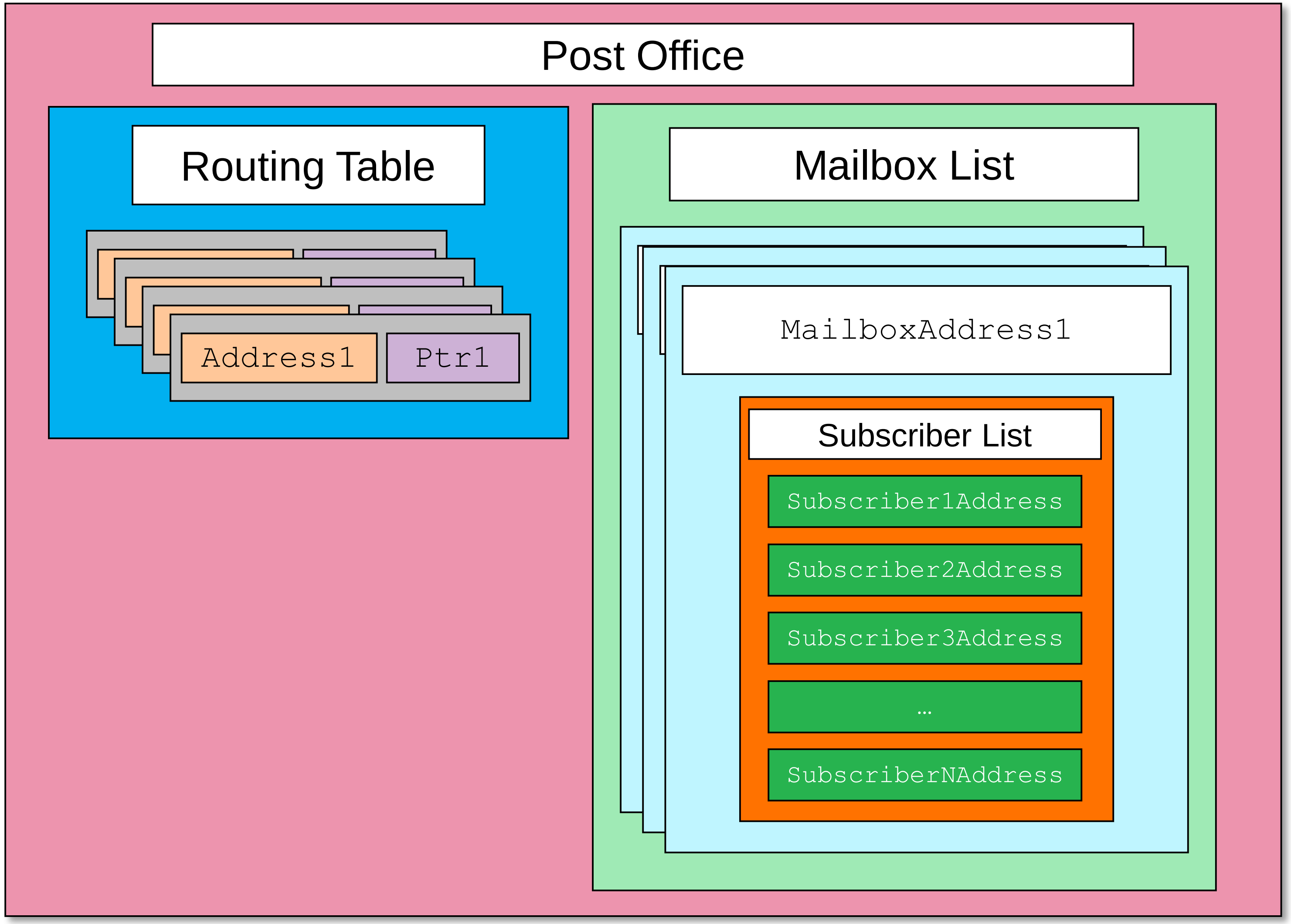
```python
msg = m2m2_packet(M2M2_ADDR_ENUM_t.M2M2_ADDR_SYS_LED_0, m2m2_led_ctrl_t())
msg.payload.command = M2M2_LED_COMMAND_ENUM_t.M2M2_LED_COMMAND_SET
msg.payload.r_pattern = M2M2_LED_PATTERN_ENUM_t.M2M2_LED_PATTERN_ON
msg.payload.g_pattern = M2M2_LED_PATTERN_ENUM_t.M2M2_LED_PATTERN_ON
msg.payload.b_pattern = M2M2_LED_PATTERN_ENUM_t.M2M2_LED_PATTERN_ON
self._send_packet(msg)
```

M2M2 Message Passing - Implementation

- M2M2 provides two entities used to perform message passing
  - Post Office
    - Provides message routing within a processor
    - Broken up into components:
      - Message routing
      - Digital interface handling between processors (i.e. UART to PC)
      - Latency can be <1 system tick
  - Mailbox
    - Provides many-to-many publish-subscribe messaging
    - An asynchronous “data pipe”
      - Producers place data into the pipe
      - Pipe is connected to a reconfigurable splitter
      - Consumers tap into the splitter and receive data put in by producers

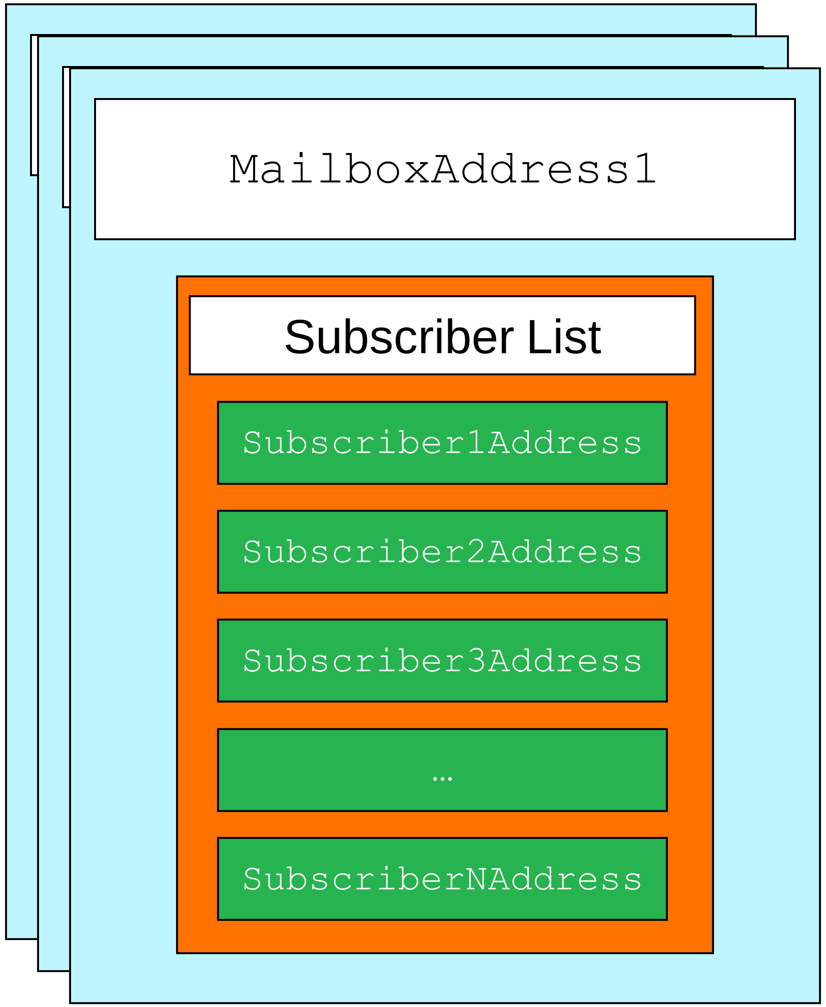
M2M2 Message Passing - Post Office

- Contains a routing table that maps M2M2 addresses to functions
- Contains a list of mailboxes
- Routes messages based on their destination address
- If routing table entry is NULL, checks the mailbox list
- If the destination is a mailbox, sends a copy of the message contents to each subscriber in the mailbox



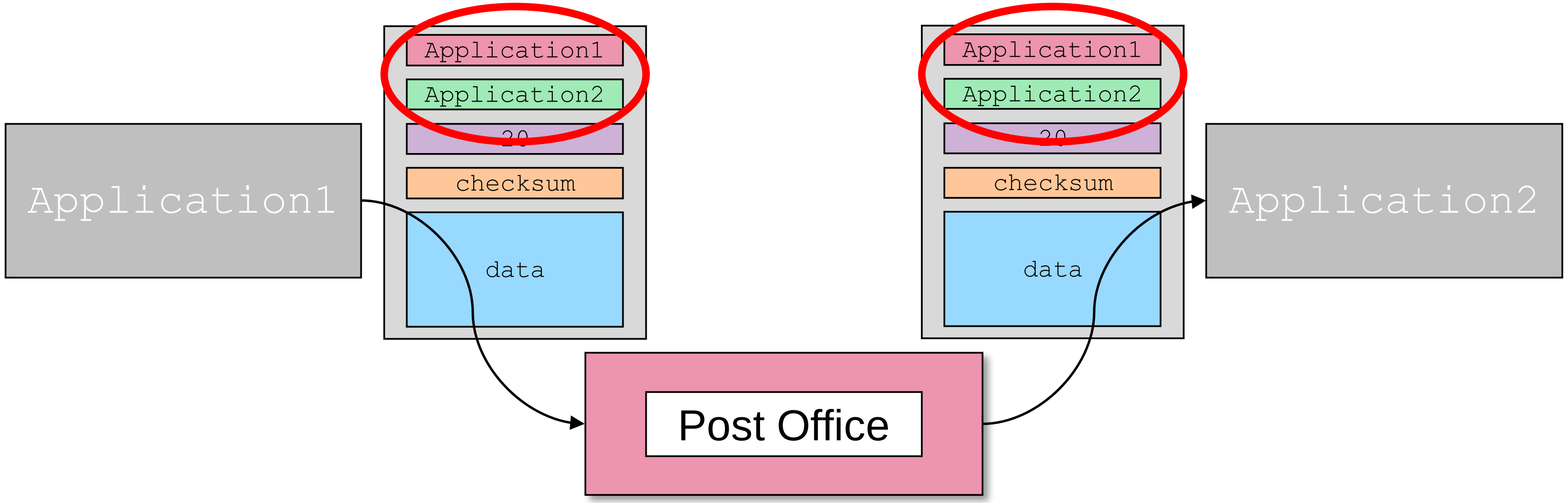
M2M2 Message Passing - Mailbox

- Implements a “stream”, “data pipe”, or “publish-subscribe” construct
- Is defined by its M2M2 address
- Maps a single M2M2 address to one or more different M2M2 addresses
- Contains a list of subscribers
  - Each subscriber is defined by their M2M2 address



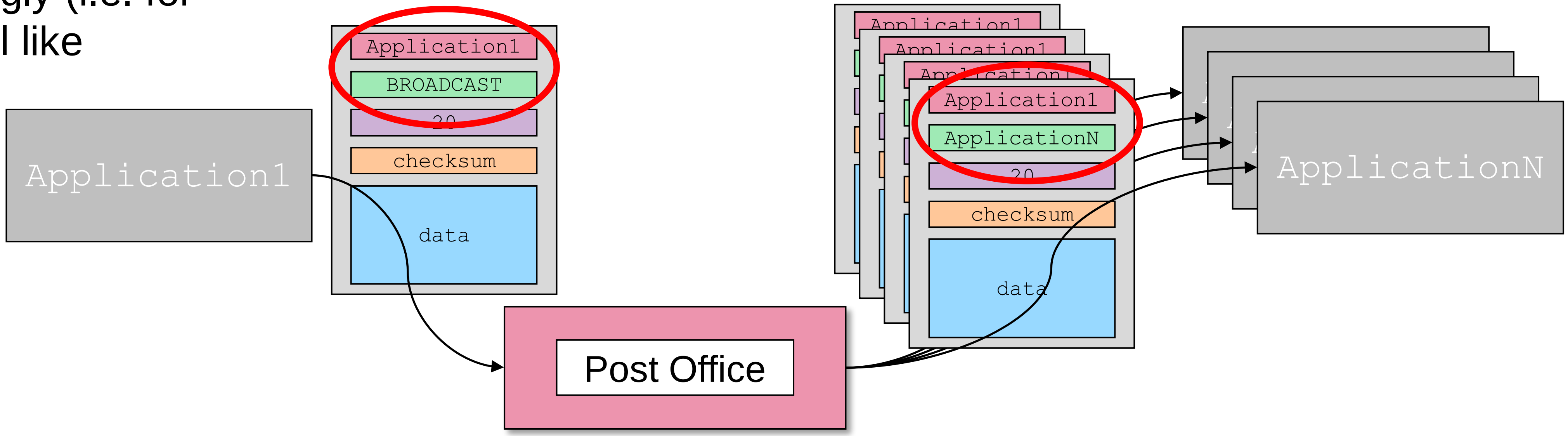
M2M2 Message Passing - Point-to-Point Messaging

- Messages are routed directly from one application to another
- The message itself is not copied, a pointer is passed instead

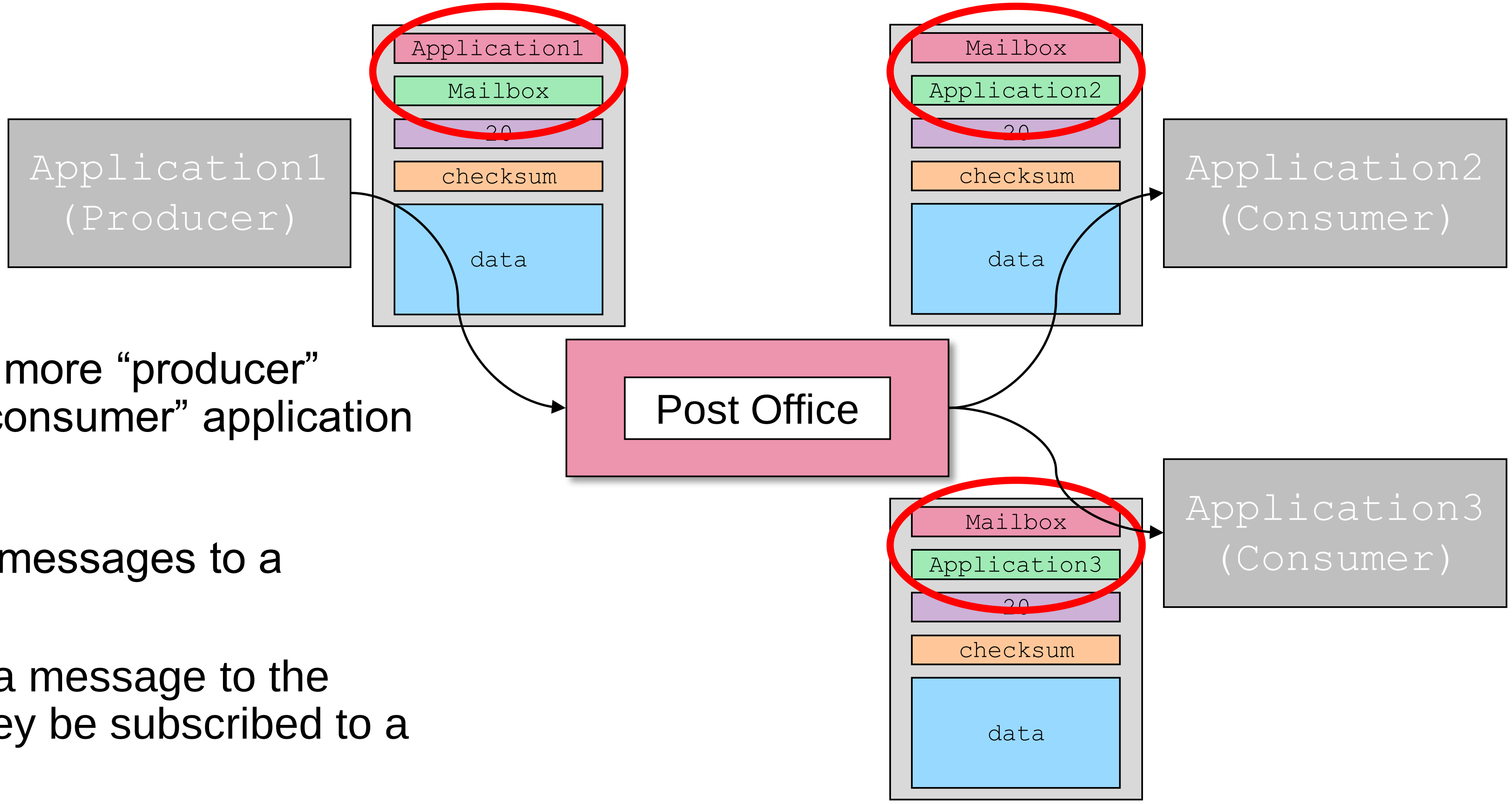


M2M2 Message Passing - Broadcast Messaging

- A message is sent to every application in the system
- Used very sparingly (i.e. for something critical like battery level)



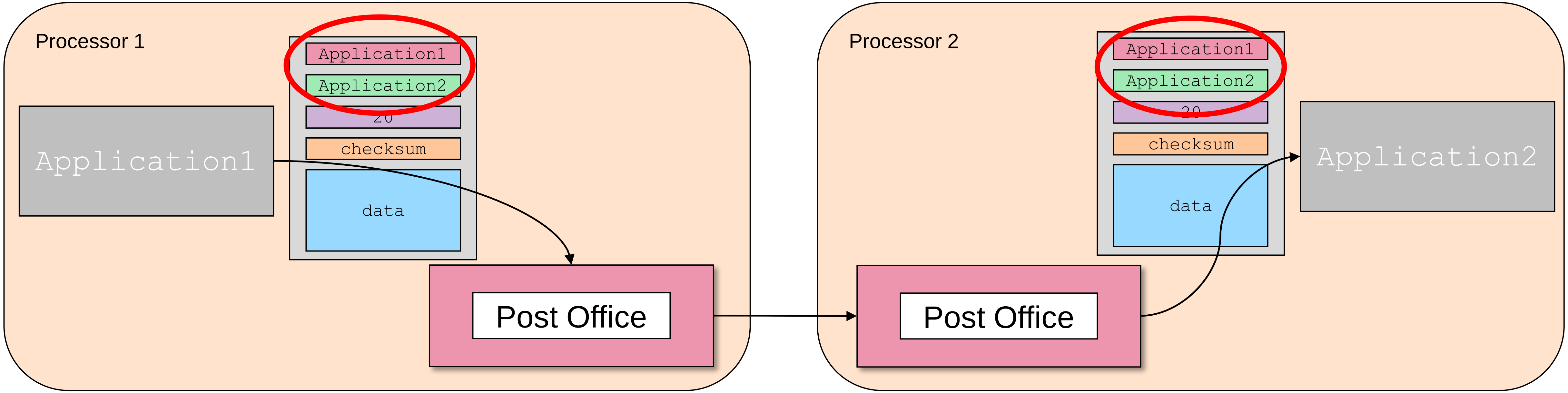
M2M2 Message Passing - Publish-Subscribe Messaging



- Messages are sent by one or more “producer” applications to one or more “consumer” application
- Message flow is one-way
- ”Producer” applications send messages to a mailbox
- Consumer applications send a message to the Post Office requesting that they be subscribed to a mailbox
- “Consumer” applications receive asynchronous messages from the mailbox

M2M2 Message Passing - Inter-Processor Messaging

- Post Offices sends message over a data bus to next Post Office
- Post Offices can be connected in any topology (ring, mesh, point-to-point)



M2M2 Message Passing - Example Point-to-Point Message Flow

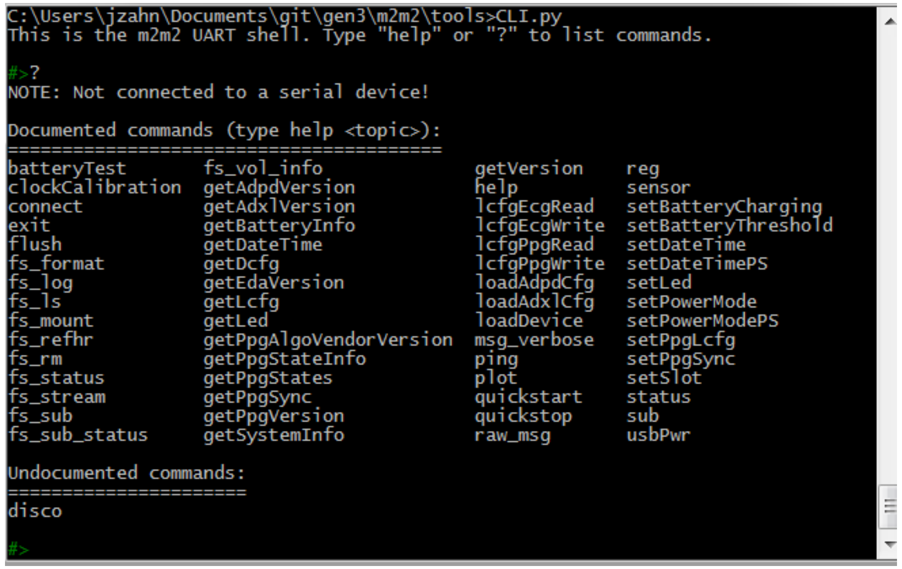
1. Application constructs message
2. Application sends message to Post Office
3. Post Office receives message pointer and:
    1. Looks up the function associated with the packet’s destination address
    2. The task’s message_send() function is called with the message pointer as an argument
    3. The message_send() function places the message pointer in the destination application’s task queue
4. The destination application receives the message pointer
5. The destination application processes the message contents
6. The destination application de-allocates the message

M2M2 Message Passing - Example Message Flow - Mailbox

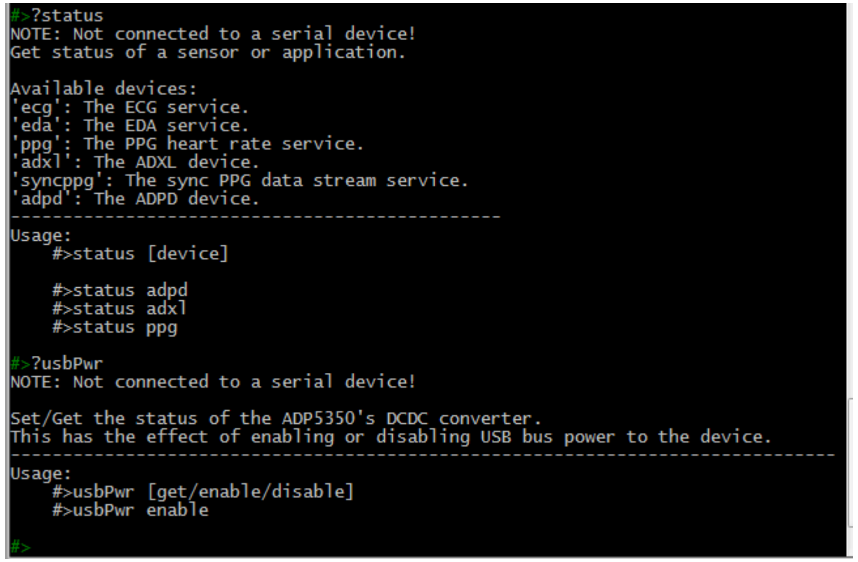
1. Application constructs message
2. Application sends message to Post Office
3. Post Office receives message pointer and:
    1. Looks up the function associated with the packet’s destination address
    2. Finds that the function pointer is NULL
    3. Looks up the destination address in the mailbox list
    4. Finds the subscriber list for the destination address
    5. Sends a copy of the message to every subscriber in the list
4. Subscriber applications all receive a copy of the message

M2M2 Tooling - Python Command Line Interface (CLI)

- Written in Python
- Just an M2M2 application
- Used as the reference tool for development
  - Firmware applications are developed and tested with the CLI first
- Easy to add new commands
- Easy to extend
- Easy to automate
- Built-in helptext



```
C:\Users\jzahn\Documents\git\gen3\m2m2\tools>CLI.py
This is the m2m2 UART shell. Type "help" or "?" to list commands.

#>?
NOTE: Not connected to a serial device!

Documented commands (type help <topic>):
========================================
batteryTest       fs_vol_info              getVersion    reg
clockCalibration  getAdpdVersion           help          sensor
connect           getAdxlVersion           lcfgEcgRead   setBatteryCharging
exit              getBatteryInfo           lcfgEcgWrite  setBatteryThreshold
flush             getDateTime              lcfgPpgRead   setDateTime
fs_format         getDcfg                  lcfgPpgWrite  setDateTimePS
fs_log            getEdaVersion            loadAdpdCfg   setLed
fs_ls             getLcfg                  loadAdxlCfg   setPowerMode
fs_mount          getLed                   loadDevice    setPowerModePS
fs_refhr          getPpgAlgoVendorVersion  msg_verbose   setPpgLcfg
fs_rm             getPpgStateInfo          ping          setPpgSync
fs_status         getPpgStates             plot          setSlot
fs_stream         getPpgSync               quickstart    status
fs_sub            getPpgVersion            quickstop     sub
fs_sub_status     getSystemInfo            raw_msg       usbPwr

Undocumented commands:
======================
disco

#>
```



```
#>?status
NOTE: Not connected to a serial device!
Get status of a sensor or application.

Available devices:
'ecg': The ECG service.
'eda': The EDA service.
'ppg': The PPG heart rate service.
'adxl': The ADXL device.
'syncppg': The sync PPG data stream service.
'adpd': The ADPD device.
-----------------------------------------------
Usage:
    #>status [device]

    #>status adpd
    #>status adxl
    #>status ppg

#>?usbPwr
NOTE: Not connected to a serial device!

Set/Get the status of the ADP5350's DCDC converter.
This has the effect of enabling or disabling USB bus power to the device.
-------------------------------------------------------------------------------
Usage:
    #>usbPwr [get/enable/disable]
    #>usbPwr enable

#>
```

M2M2 Tooling - Why Python?

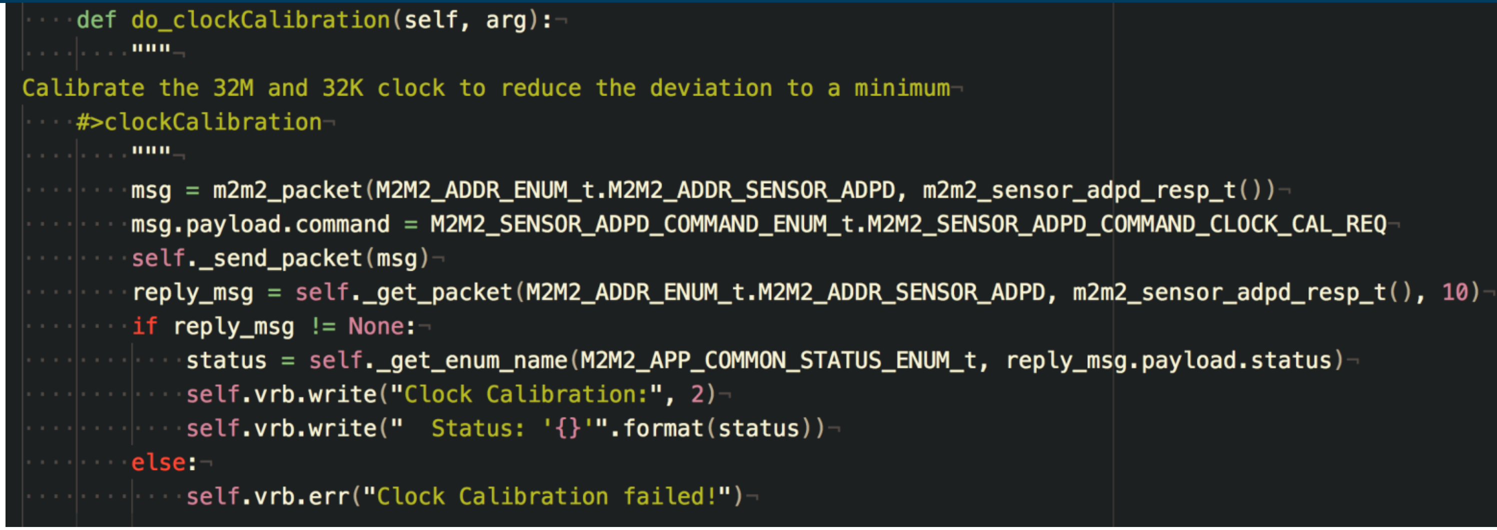
- Cross-platform
  - x86, ARM, Windows, macOS, Linux, Android
- Free, open-sourced
  - No IDEs/licenses to pay for (looking at you, Visual Studio/Labview/Matlab)
- Well-supported
  - Google, Dropbox, Amazon, Linux foundation, Sun, IBM, Apple
- Package Manager
- Interpreted
- Huge catalog of libraries
  - pyVISA to control test instruments over USB/RS-232/Ethernet/GPIB
  - pySerial to interface with serial ports
  - FTDI
  - GUIs
  - MatplotLib/numpy
    - Capable of replacing Matlab

M2M2 Tooling - Sample (non-trivial!) CLI Commands

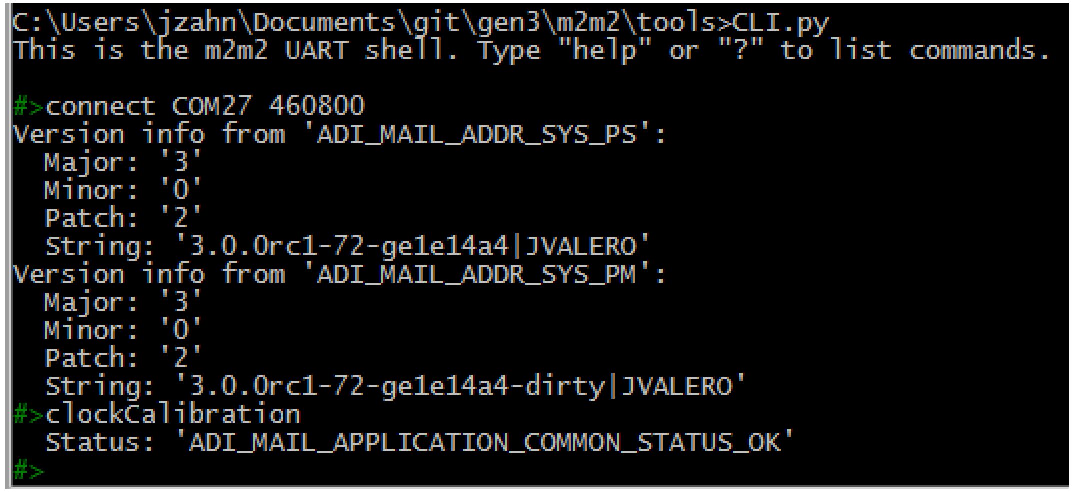
- batteryTest
  - Will either charge or discharge the battery to a particular level
  - Tracks charge %, voltage, and battery temperature every 10s until the target level is reached
  - Enables automated battery profiling
- fs_stream
  - Reads a file from the NAND flash
  - Tracks file transfer rate and progress
- quickstart/quickrun
  - Run a pre-defined set of CLI commands with one shortcut
- msg_verbose
  - Changes the CLI's verbosity level
  - Can dump raw hex values of packets
  - Coloured messages to distinguish between verbosity levels

M2M2 Tooling - CLI Code Example: Clock Calibration

- A new packet (***msg***) is created, with destination and payload specified
- The ***command*** field of ***msg***’s payload is set
- The packet is sent
- We block waiting for a message from ***M2M2_ADDR_SENSOR_ADPD***, that will be unpacked into a ***m2m2_sensor_adpd_resp_t***, with a timeout of 10s
- Look up the enum name for the status value in ***reply_msg***
- Print the command status



```
    def do_clockCalibration(self, arg):
        """
Calibrate the 32M and 32K clock to reduce the deviation to a minimum
    #>clockCalibration
        """
        msg = m2m2_packet(M2M2_ADDR_ENUM_t.M2M2_ADDR_SENSOR_ADPD, m2m2_sensor_adpd_resp_t())
        msg.payload.command = M2M2_SENSOR_ADPD_COMMAND_ENUM_t.M2M2_SENSOR_ADPD_COMMAND_CLOCK_CAL_REQ
        self._send_packet(msg)
        reply_msg = self._get_packet(M2M2_ADDR_ENUM_t.M2M2_ADDR_SENSOR_ADPD, m2m2_sensor_adpd_resp_t(), 10)
        if reply_msg != None:
            status = self._get_enum_name(M2M2_APP_COMMON_STATUS_ENUM_t, reply_msg.payload.status)
            self.vrb.write("Clock Calibration:", 2)
            self.vrb.write("  Status: '{}'".format(status))
        else:
            self.vrb.err("Clock Calibration failed!")
```



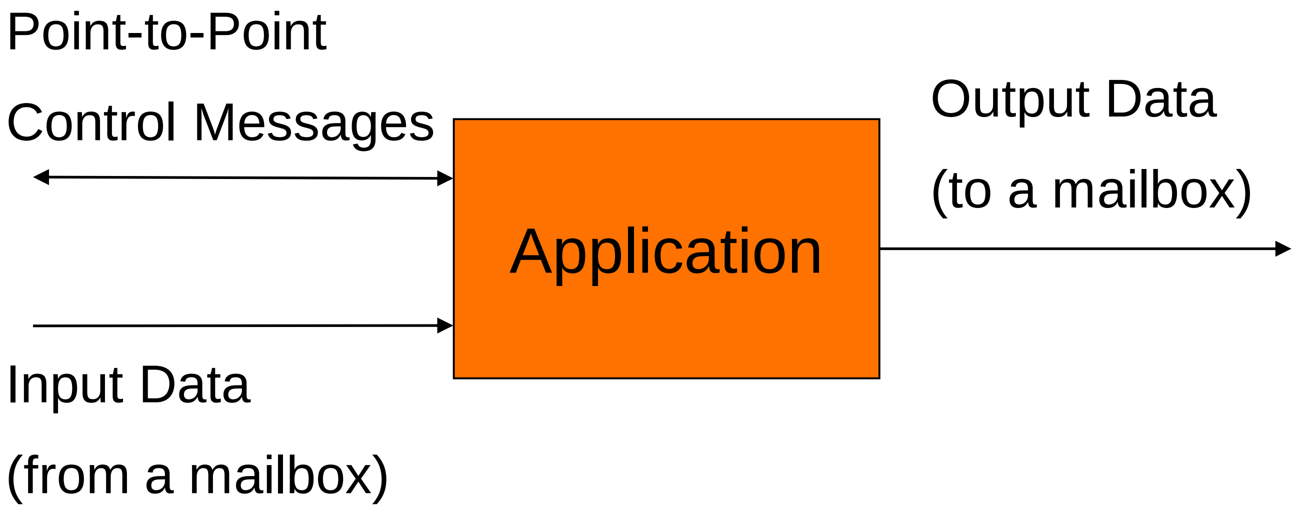
```
C:\Users\jzahn\Documents\git\gen3\m2m2\tools>CLI.py
This is the m2m2 UART shell. Type "help" or "?" to list commands.

#>connect COM27 460800
Version info from 'ADI_MAIL_ADDR_SYS_PS':
  Major: '3'
  Minor: '0'
  Patch: '2'
  String: '3.0.0rc1-72-ge1e14a4|JVALERO'
Version info from 'ADI_MAIL_ADDR_SYS_PM':
  Major: '3'
  Minor: '0'
  Patch: '2'
  String: '3.0.0rc1-72-ge1e14a4-dirty|JVALERO'
#>clockCalibration
  Status: 'ADI_MAIL_APPLICATION_COMMON_STATUS_OK'
#>
```

M2M2 Application - Components

- Each M2M2 application has the following parts:
  - Task init function
    - Run during system start-up
    - Set up application's internal state
    - Set up application's RTOS task
    - Set up M2M2 mailboxes published to by the application
  - Send message function
    - Called by the post office whenever a message is to be sent to the application
    - Just puts the message into the application's message queue
  - Application function
    - Main body of the application
    - The function running as the RTOS task
    - Contains a main loop which blocks waiting for events

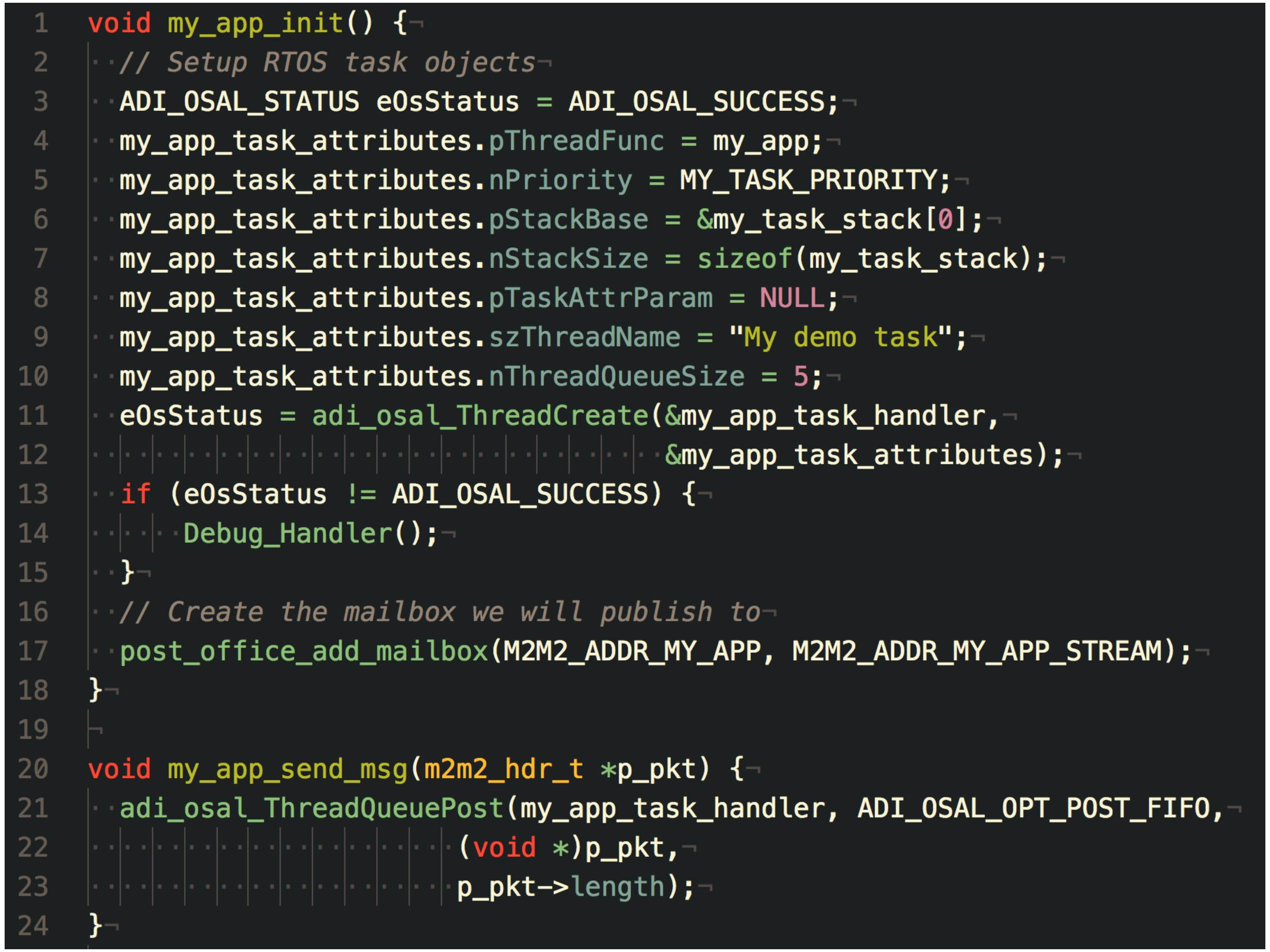
M2M2 Application - Example Application - Introduction

- We're going to implement a simple M2M2 application
  - Init function
  - Message send function
  - Main body
- The application will:
  - Subscribe to a mailbox
  - Process data it receives from the mailbox
  - Process command messages and send responses
  - Publish new data to a new mailbox



M2M2 Application - Example Application - Init and Message Send Functions

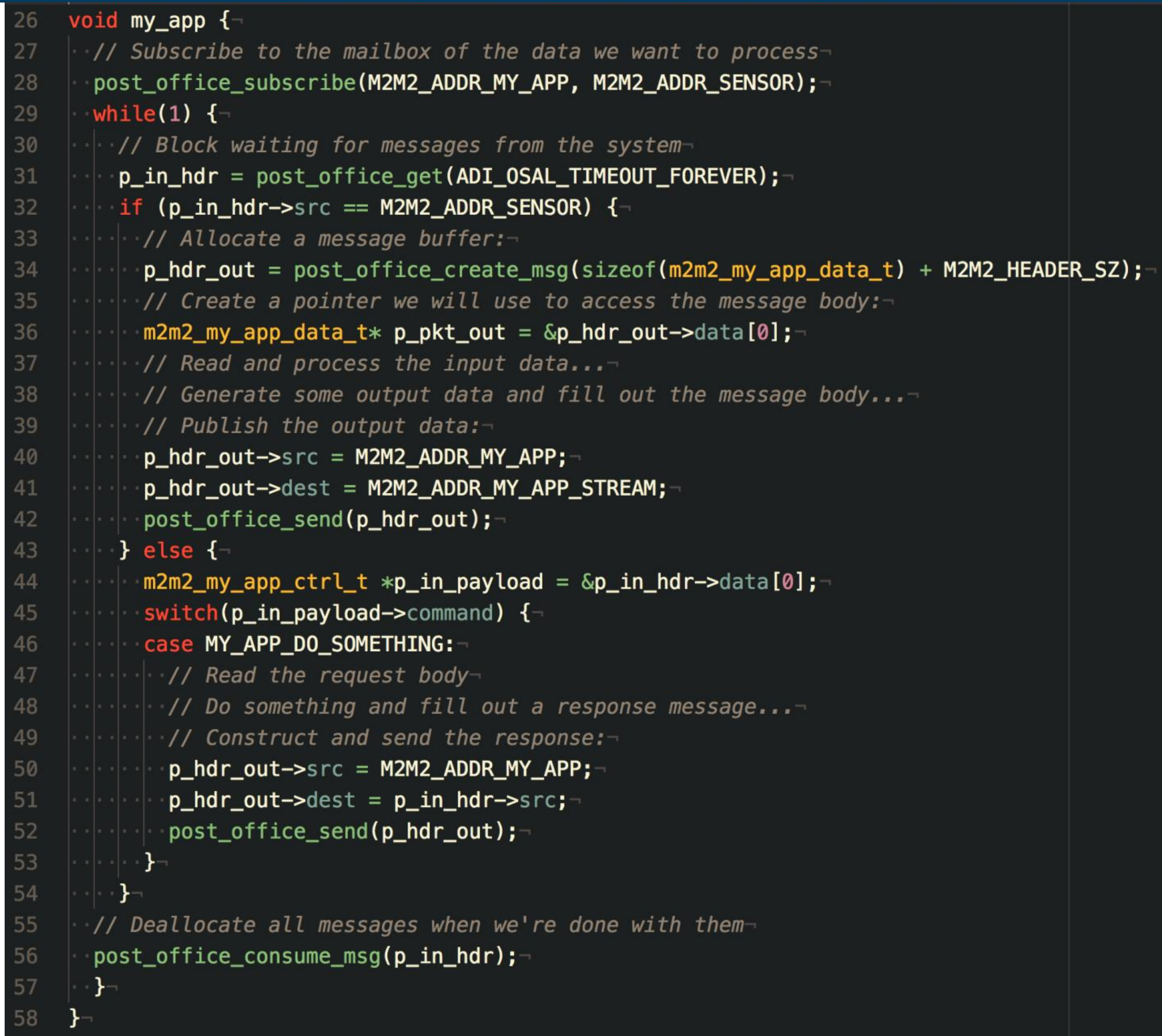
- Init function
  - Configures RTOS task objects
  - Creates RTOS task for the application
    - The task is started later by the boot task
  - Creates a new mailbox with the address M2M2_ADDR_MY_APP_STREAM
- Send Message function
  - Puts a message in the application's task queue



```
void my_app_init() {
  // Setup RTOS task objects
  ADI_OSAL_STATUS eOsStatus = ADI_OSAL_SUCCESS;
  my_app_task_attributes.pThreadFunc = my_app;
  my_app_task_attributes.nPriority = MY_TASK_PRIORITY;
  my_app_task_attributes.pStackBase = &my_task_stack[0];
  my_app_task_attributes.nStackSize = sizeof(my_task_stack);
  my_app_task_attributes.pTaskAttrParam = NULL;
  my_app_task_attributes.szThreadName = "My demo task";
  my_app_task_attributes.nThreadQueueSize = 5;
  eOsStatus = adi_osal_ThreadCreate(&my_app_task_handler,
                                    &my_app_task_attributes);
  if (eOsStatus != ADI_OSAL_SUCCESS) {
      Debug_Handler();
  }
  // Create the mailbox we will publish to
  post_office_add_mailbox(M2M2_ADDR_MY_APP, M2M2_ADDR_MY_APP_STREAM);
}

void my_app_send_msg(m2m2_hdr_t *p_pkt) {
  adi_osal_ThreadQueuePost(my_app_task_handler, ADI_OSAL_OPT_POST_FIFO,
                       (void *)p_pkt,
                       p_pkt->length);
}
```

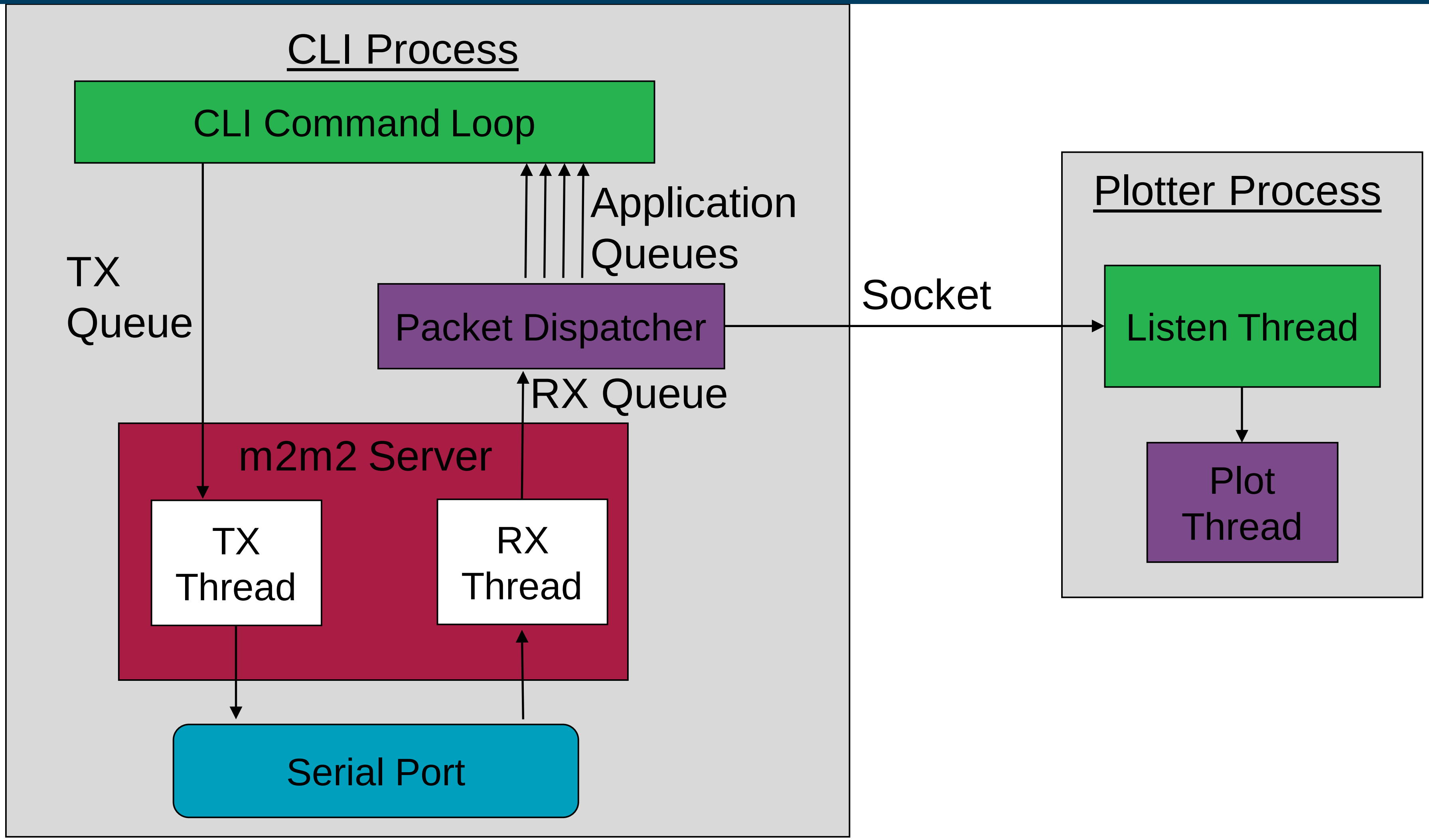
M2M2 Application - Example Application - Main Body

- Subscribe to the mailbox whose data we're going to process
- Block waiting for a message to arrive
  - The application consumes no CPU time while blocked
- When a message arrives, the task is unblocked
- Check the source of the message
  - If it's input data, we process it and produce some output
  - If it's a command message, we perform the requested action and send a response
- De-allocate all received messages



```c
void my_app {
  // Subscribe to the mailbox of the data we want to process
  post_office_subscribe(M2M2_ADDR_MY_APP, M2M2_ADDR_SENSOR);
  while(1) {
    // Block waiting for messages from the system
    p_in_hdr = post_office_get(ADI_OSAL_TIMEOUT_FOREVER);
    if (p_in_hdr->src == M2M2_ADDR_SENSOR) {
      // Allocate a message buffer:
      p_hdr_out = post_office_create_msg(sizeof(m2m2_my_app_data_t) + M2M2_HEADER_SZ);
      // Create a pointer we will use to access the message body:
      m2m2_my_app_data_t* p_pkt_out = &p_hdr_out->data[0];
      // Read and process the input data...
      // Generate some output data and fill out the message body...
      // Publish the output data:
      p_hdr_out->src = M2M2_ADDR_MY_APP;
      p_hdr_out->dest = M2M2_ADDR_MY_APP_STREAM;
      post_office_send(p_hdr_out);
    } else {
      m2m2_my_app_ctrl_t *p_in_payload = &p_in_hdr->data[0];
      switch(p_in_payload->command) {
      case MY_APP_DO_SOMETHING:
        // Read the request body
        // Do something and fill out a response message...
        // Construct and send the response:
        p_hdr_out->src = M2M2_ADDR_MY_APP;
        p_hdr_out->dest = p_in_hdr->src;
        post_office_send(p_hdr_out);
      }
    }
  // Deallocate all messages when we're done with them
  post_office_consume_msg(p_in_hdr);
  }
}
```

CLI Architecture



Existing Applications

- Sensors
  - ADPD
  - ADXL
  - AD7156
  - AD5940
- Applications
  - System
  - EDA
  - ECG
  - Filesystem
  - Post Office
  - Command Line (CLI)
  - WaveTool

- Streams
  - ADPD
  - ADXL
  - ECG
  - EDA
  - PPG
  - SyncPPG
  - Pedometer