Mongoose Flow

run_mongoose

main.c

```
#define run_mongoose() \
  do {                 \
    mongoose_init();   \
    for (;;) {         \
      mongoose_poll(); \
    }                  \
  } while (0)
```

glue.h

mongoose_init

Impl.c

```
#if WIZARD_ENABLE_MQTT
  MG_INFO(("Starting MQTT reconnection timer"));
  mg_timer_add(&g_mgr, 1000, MG_TIMER_REPEAT, mqtt_timer,
&g_mgr);
#endif
```

glue_init_1 in mongoose_glue.c

```
static void mqtt_timer(void *arg) {
  struct mg_mgr *mgr = (struct mg_mgr *) arg;
  if (g_mqtt_conn == NULL) {
    g_mqtt_conn = glue_mqtt_connect(mgr, mqtt_event_handler);
  }
}
#endif  // WIZARD_ENABLE_MQTT
```

```c
static void mqtt_event_handler(struct mg_connection
*c, int ev, void *ev_data) {
  if (ev == MG_EV_CONNECT) {
    glue_mqtt_tls_init(c);
  } else if (ev == MG_EV_MQTT_OPEN) {
    glue_mqtt_on_connect(c, *(int *) ev_data);
  } else if (ev == MG_EV_MQTT_CMD) {
    glue_mqtt_on_cmd(c, ev_data);
  } else if (ev == MG_EV_MQTT_MSG) {
    struct mg_mqtt_message *mm = (struct
mg_mqtt_message *) ev_data;
    glue_mqtt_on_message(c, mm->topic, mm->data);
  } else if (ev == MG_EV_CLOSE) {
    MG_DEBUG(("%lu Closing", c->id));
    g_mqtt_conn = NULL;
  }
}
```

```c
struct mg_mgr g_mgr;  // Mongoose event manager


struct mg_mgr {
  struct mg_connection *conns;  // List of active connections
  struct mg_dns dns4;           // DNS for IPv4
  struct mg_dns dns6;           // DNS for IPv6
  int dnstimeout;               // DNS resolve timeout in milliseconds
  bool use_dns6;                // Use DNS6 server by default, see #1532
  unsigned long nextid;         // Next connection ID
  unsigned long timerid;        // Next timer ID
  void *userdata;               // Arbitrary user data pointer
  void *tls_ctx;                // TLS context shared by all TLS sessions
  uint16_t mqtt_id;             // MQTT IDs for pub/sub
  void *active_dns_requests;    // DNS requests in progress
  struct mg_timer *timers;      // Active timers
  int epoll_fd;                 // Used when MG_EPOLL_ENABLE=1
  void *priv;                   // Used by the MIP stack
  size_t extraconnsize;         // Used by the MIP stack
  MG_SOCKET_TYPE pipe;          // Socketpair end for mg_wakeup()
#if MG_ENABLE_FREERTOS_TCP
  SocketSet_t ss;  // NOTE(lsm): referenced from socket struct
#endif
};
```

```c
struct mg_connection *g_mqtt_conn;

struct mg_connection {
  struct mg_connection *next;  // Linkage in struct mg_mgr :: connections
  struct mg_mgr *mgr;          // Our container
  struct mg_addr loc;          // Local address
  struct mg_addr rem;          // Remote address
  void *fd;                    // Connected socket, or LWIP data
  unsigned long id;            // Auto-incrementing unique connection ID
  struct mg_iobuf recv;        // Incoming data
  struct mg_iobuf send;        // Outgoing data
```

```
  struct mg_iobuf prof;        // Profile data enabled by MG_ENABLE_PROFILE
  struct mg_iobuf rtls;        // TLS only. Incoming encrypted data
  mg_event_handler_t fn;       // User-specified event handler function
  void *fn_data;               // User-specified function parameter
  mg_event_handler_t pfn;      // Protocol-specific handler function
  void *pfn_data;              // Protocol-specific function parameter
  char data[MG_DATA_SIZE];     // Arbitrary connection data
  void *tls;                   // TLS specific data
  unsigned is_listening : 1;   // Listening connection
  unsigned is_client : 1;      // Outbound (client) connection
  unsigned is_accepted : 1;    // Accepted (server) connection
  unsigned is_resolving : 1;   // Non-blocking DNS resolution is in progress
  unsigned is_arplooking : 1;  // Non-blocking ARP resolution is in progress
  unsigned is_connecting : 1;  // Non-blocking connect is in progress
  unsigned is_tls : 1;         // TLS-enabled connection
  unsigned is_tls_hs : 1;      // TLS handshake is in progress
  unsigned is_udp : 1;         // UDP connection
  unsigned is_websocket : 1;   // WebSocket connection
  unsigned is_mqtt5 : 1;       // For MQTT connection, v5 indicator
  unsigned is_hexdumping : 1;  // Hexdump in/out traffic
  unsigned is_draining : 1;    // Send remaining data, then close and free
  unsigned is_closing : 1;     // Close and free the connection immediately
  unsigned is_full : 1;        // Stop reads, until cleared
  unsigned is_resp : 1;        // Response is still being generated
  unsigned is_readable : 1;    // Connection is ready to read
  unsigned is_writable : 1;    // Connection is ready to write
  unsigned is_io_err : 1;      // Remember IO_ERR condition for later use
};
```

```
struct mg_mqtt_opts opts;

struct mg_mqtt_opts {
  struct mg_str user;               // Username, can be empty
  struct mg_str pass;               // Password, can be empty
  struct mg_str client_id;          // Client ID
  struct mg_str topic;              // message/subscription topic
  struct mg_str message;            // message content
  uint8_t qos;                      // message quality of service
  uint8_t version;                  // Can be 4 (3.1.1), or 5. If 0, assume 4
  uint16_t keepalive;               // Keep-alive timer in seconds
  uint16_t retransmit_id;           // For PUBLISH, init to 0
  bool retain;                      // Retain flag
  bool clean;                       // Clean session flag
  struct mg_mqtt_prop *props;       // MQTT5 props array
  size_t num_props;                 // number of props
  struct mg_mqtt_prop *will_props;  // Valid only for CONNECT packet (MQTT5)
  size_t num_will_props;            // Number of will props
};
```
