UAV wireless communication and video streaming system

ADAM CSIZY
Version 1.1.0
2021-06-24

Data Structure Documentation

ModuleMessage Struct Reference

Structure of module message.

Data Fields

ModuleName_T address
ModuleMessageCode_T code
ModuleMessageData_T data

Detailed Description

Structure of module message.

Field Documentation

ModuleName_T ModuleMessage::address

Target address of module message

ModuleMessageCode_T ModuleMessage::code

Code of module message

ModuleMessageData_T ModuleMessage::data

Data of module message

The documentation for this struct was generated from the following file:

CompanionComputer/includes/**com_utils.h**

ModuleMessageData Union Reference

Union of module message data.

Data Fields

VideoCodingFormat_T codingFormat
VideoStreamPort_T videoStreamPort

Detailed Description

Union of module message data.

Field Documentation

VideoCodingFormat_T ModuleMessageData::codingFormat

Video coding format

VideoStreamPort_T ModuleMessageData::videoStreamPort

Port number on which the ground control accepts the video stream

---

The documentation for this union was generated from the following file:

CompanionComputer/includes/**com_utils.h**

ModuleMessageQueue Struct Reference

Structure of module message queue.

Data Fields

pthread_mutex_t **lock**
pthread_cond_t **update**
size_t **size**
size_t **front**
size_t **back**
ModuleMessage_T ** messages

Detailed Description

Structure of module message queue.

Each module message queue consists of a circular buffer a mutex and a conditional variable to guarantee thread safety.

Field Documentation

size_t ModuleMessageQueue::back

Tail of the circular buffer

size_t ModuleMessageQueue::front

Head of the circular buffer

pthread_mutex_t ModuleMessageQueue::lock

Mutex of the circular buffer

ModuleMessage_T* * ModuleMessageQueue::messages

Array of module messages (the buffer itself)

size_t ModuleMessageQueue::size

Size of the circular buffer

pthread_cond_t ModuleMessageQueue::update

Conditional variable of the circular buffer

The documentation for this struct was generated from the following file:

CompanionComputer/includes/**com_utils.h**

NetworkInitContext Struct Reference

Structure of network module's initialization context.

Data Fields

const char * **serverNodeName**
const char * **serverServiceName**

Detailed Description

Structure of network module's initialization context.

A structure holding arguments and parameters for initializing the network module.

Field Documentation

const char* NetworkInitContext::serverNodeName

Server address

const char* NetworkInitContext::serverServiceName

Server port

The documentation for this struct was generated from the following file:

CompanionComputer/includes/**com_utils.h**

StateContext Struct Reference

Struct of stream state context.

Data Fields

StreamState_T nextState
EventHandler_T **eventHandler**

Detailed Description

Struct of stream state context.

The documentation for this struct was generated from the following file:
CompanionComputer/src/**stream_utils.c**

VideoCodingFormatCaps Struct Reference

Video coding format capabilities.

Data Fields

int **supported**
int **width**
int **height**
int **framerateNumerator**
int **framerateDenominator**

Detailed Description

Video coding format capabilities.

A structure for storing capabilities for any video coding format supported by this software. Each member field describes a capability. Note that the structure functions as a union of capabilities and some of them might not be used for a given video coding format.

Field Documentation

int VideoCodingFormatCaps::framerateDenominator

Framerate denominator

int VideoCodingFormatCaps::framerateNumerator

Framerate numerator

int VideoCodingFormatCaps::height

Frame height

int VideoCodingFormatCaps::supported

Flag whether the format is supported

int VideoCodingFormatCaps::width

Frame width

The documentation for this struct was generated from the following file:

CompanionComputer/includes/**camera_utils.h**

VideoCodingFormatContext Struct Reference

Context for video coding formats.

Data Fields

VideoCodingFormatCaps_T * capsArray
size_t **size**

Detailed Description

Context for video coding formats.

Used as a wrapper to encapsulate the array of video coding format capabilities and its size.

Field Documentation

VideoCodingFormatCaps_T* VideoCodingFormatContext::capsArray

Array of video coding format capabilities

size_t VideoCodingFormatContext::size

Size of the array

The documentation for this struct was generated from the following file:

CompanionComputer/includes/**camera_utils.h**

File Documentation

CompanionComputer/includes/camera_utils.h File Reference

Detailed Description

Camera utilities.

Author

Adam Csizy

Date

2021-03-19

Version

v1.1.0

Macro Definition Documentation

#define NUM_SUP_VID_COD_FMT 7U

Number of supported video coding formats (see VideoCodingFormat_T).

Typedef Documentation

typedef enum VideoCodingFormat VideoCodingFormat_T

Enumeration of video coding formats supported by the GStreamer framework v4l2src element.

Note

This software supports only a subset of the enumerated formats.

The enumeration values also define the priorities of the preferred video coding formats.

typedef struct VideoCodingFormatCaps VideoCodingFormatCaps_T

Video coding format capabilities.

A structure for storing capabilities for any video coding format supported by this software. Each member field describes a capability. Note that the structure functions as a union of capabilities and some of them might not be used for a given video coding format.

typedef struct VideoCodingFormatContext VideoCodingFormatContext_T

Context for video coding formats.

Used as a wrapper to encapsulate the array of video coding format capabilities and its size.

Enumeration Type Documentation

enum VideoCodingFormat

Enumeration of video coding formats supported by the GStreamer framework v4l2src element.

Note

This software supports only a subset of the enumerated formats.

The enumeration values also define the priorities of the preferred video coding formats.

Enumerator:

CAM_FMT_H265	H.265
CAM_FMT_H264	H.264
CAM_FMT_VP8	VP8
CAM_FMT_VP9	VP9
CAM_FMT_JPEG	JPEG
CAM_FMT_H263	H.263
CAM_FMT_RAW	RAW
CAM_FMT_MPEG	MPEG (not used)
CAM_FMT_MPEGTS	MPEGTS (not used)
CAM_FMT_BAYER	BAYER (not used)
CAM_FMT_DV	Digital Video (not used)
CAM_FMT_FWHT	FWHT (not used)
CAM_FMT_PWC1	PWC1 (not used)
CAM_FMT_PWC2	PWC2 (not used)
CAM_FMT_SONIX	Sonix (not used)
CAM_FMT_WMV	WMV (not used)

CAM_FMT_UNK	Unknown format

Function Documentation

int getCameraCapabilities (GstElement * *v4l2srcElement*, VideoCodingFormatContext_T * *ctx*)

Retrieves capabilities for the given camera device.

Retrieves capabilities for the given video camera device represented as a v4l2src pipeline element. This function tries to retrieve the best possible capability configuration for each video encoding format. The retrieved capabilities for each format are being stored in an array encapsulated in the user data context (cameraCapsContext).

Note

This function should be called when the given camera device element is in READY or higher state. It is necessary because the v4l2src element needs to query the underlying hardware to refine and distinct its capabilities from the pad template's capabilities.

Parameters

in	*v4l2srcElement*	Pointer to a v4l2src pipeline element representing the corresponding camera device.
in,out	*ctx*	User data context.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int getCameraDevicePath (char *cameraDevicePath*[], const size_t *size*)

Searches and validates camera device.

Searches for camera devices under '/dev' directory and returns with the path of the first compatible device entry. Validation includes V4L2 interface compatibility and Video Capture capability.

Note

None

Parameters

out	*cameraDevicePath*	Path string.
in	*size*	Size of path string.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int videoCodingFormatToString (const VideoCodingFormat_T *format*, char *string*[], const size_t *size*)

Converts video coding format to string.

Converts the given VideoCodingFormat_T representation of a video coding format into a string representation.

Note

None

Parameters

in	*format*	VideoCodingFormat_T representation.
out	*string*	String representation of video coding format.
in	*size*	Size of the string.

Returns

Result of execution.

Return values

0	Success
-1	Failure

CompanionComputer/includes/com_utils.h File Reference

Detailed Description

Communication utilities and network module.

Author

Adam Csizy

Date

2021-04-01

Version

v1.1.0

Macro Definition Documentation

#define DRONE_ID 12U

Drone ID

#define MOD_MSGQ_BLOCK 0

Module message queue blocking flag

#define MOD_MSGQ_NOBLOCK 1

Module message queue non-blocking flag

#define VideoStreamPort_T uint32_t

Type of video streaming port number

Typedef Documentation

typedef struct ModuleMessageQueue ModuleMessageQueue_T

Structure of module message queue.

Each module message queue consists of a circular buffer a mutex and a conditional variable to guarantee thread safety.

typedef struct NetworkInitContext NetworkInitContext_T

Structure of network module's initialization context.

A structure holding arguments and parameters for initializing the network module.

Enumeration Type Documentation

enum ModuleMessageCode

Enumeration of module message codes.

Enumerator:

MOD_MSG_CODE_LOGIN	Login to ground control (drone)
MOD_MSG_CODE_LOGIN_ACK	Login confirmed (ground control)
MOD_MSG_CODE_STREAM_REQ	Request video stream (ground control)
MOD_MSG_CODE_STREAM_ERROR	Internal error in video stream (drone)
MOD_MSG_CODE_STREAM_START	Start video stream (ground control)
MOD_MSG_CODE_STREAM_STOP	Stop video stream (ground control)
MOD_MSG_CODE_STREAM_TYPE	Type of requested video stream (drone)
MOD_MSG_CODE_LOGIN_NACK	Login not confirmed (ground control)

enum ModuleName

Enumeration of independent modules.

Enumerator:

MOD_NAME_NETWORK	Network module (drone)
MOD_NAME_STREAM	Video streaming module (drone)
MOD_NAME_GCCOMMON	Ground control common module (ground control)

Function Documentation

int deinitModuleMessageQueue (ModuleMessageQueue_T * *messageQueue*)

Deinitialize module message queue.

Deinitializes module message queue object. The remaining messages in the queue and the buffer itself are freed.

Note

Not thread safe. Blocking.

Parameters

in,out	*messageQueue*	Message queue object to be deinitialized.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int initModuleMessageQueue (ModuleMessageQueue_T * *messageQueue*, const size_t *size*)

Initialize module message queue.

Initializes module message queue object. The given message queue object must be in an uninitialized or deinitialized state. Double initialization of a message queue object might result undefined behaviour. The value of the size input argument must be the power of two because the implementation uses bitmask for efficient boundary check.

Note

Not thread safe. Blocking.

Parameters

in,out	*messageQueue*	Message queue object to be initialized.
in	*size*	Size of the message queue buffer.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int initNetworkModule (NetworkInitContext_T * *initCtx*)

Initialize network handler module.

Initializes the network handler module's message queue and starts the network traffic handler threads.

Parameters

in	*initCtx*	Initialization context.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int insertModuleMessage (ModuleMessageQueue_T * *messageQueue*, ModuleMessage_T * *message*, const int *noblock*)

Insert message into module message queue.

Inserts module message into the given module message queue (FIFO). The message object must be dynamically allocated by the source and released after insertion.

Note

Thread safe.

Parameters

in,out	*messageQueue*	Module message queue object.
in	*message*	Dynamically allocated module message object.
in	*noblock*	Flag whether the call should be non-blocking.

Returns

Result of execution.

Return values

0	Success
-1	Failure
-2	Could not lock queue (prevent blocking)
-3	Message queue is full (prevent blocking)

int printModuleMessage (const ModuleMessage_T * *message*)

Print module message.

Prints module message data on the standard output.

Parameters

in	*message*	Module message object to be printed.

Returns

Result of execution.

Return values

0	Success
-1	Failure

ssize_t recvTimeout (int *sockfd*, void * *buf*, size_t *len*, int *flags*, time_t *sec*, useconds_t *usec*)

Wrapper for 'recv()' with timeout option.

This function serves as a wrapper for function 'recv()' with timeout option.

Parameters

in	*sockfd*	Socket file descriptor.
out	*buf*	Received data buffer.
in	*len*	Size of the data to be received in bytes.
in	*flags*	Flags.
in	*sec*	Timeout interval in seconds.
in	*usec*	Timeout interval in microseconds.

Returns

The number of bytes received, or -1 if an error occurred. In the event of an error, errno is set to indicate the error.

int removeModuleMessage (ModuleMessageQueue_T * *messageQueue*, ModuleMessage_T ** *message*, const int *noblock*)

Remove message from module message queue.

Removes module message from the given module message queue (FIFO). The message object must be freed by the receiver.

Note

Thread safe.

Parameters

in,out	*messageQueue*	Module message queue object.
out	*message*	Module message object pointer.
in	*noblock*	Flag whether the call should be non-blocking.

Returns

Result of execution.

Return values

0	Success
-1	Failure
-2	Could not lock queue (prevent blocking)
-3	Message queue is full (prevent blocking)

Variable Documentation

ModuleMessageQueue_T networkMsgq `[extern]`

Module message queue of the network module

CompanionComputer/includes/log_utils.h File Reference

Detailed Description

Logging utilities.

Author

Adam Csizy

Date

2021-03-19

Version

v1.1.0

Enumeration Type Documentation

enum LogSeverity

Enumeration of log severity levels.

Enumerator:

LOG_SVRTY_ERR	Error
LOG_SVRTY_WRN	Warning
LOG_SVRTY_INF	Information
LOG_SVRTY_DBG	Debug information

Function Documentation

void createLogMessage (const char *message*[], const LogSeverity_T *severity*)

Create log message.

Generates a log entry in the system log and prints the log message on the standard output with the given severity level if debug mode is enabled.

Note

None

Parameters

in	*message*	Log message.
in	*severity*	Log severity level.

CompanionComputer/includes/stream_utils.h File Reference

Detailed Description

Streaming utilities and video streaming module.

Author

Adam Csizy

Date

2021-04-05

Version

v1.1.0

Function Documentation

int initStreamModule (void)

Initialize streaming module.

Initializes the streaming module's message queue and starts the streaming control thread.

Returns

Result of execution.

Return values

0	Success
-1	Failure

Variable Documentation

ModuleMessageQueue_T streamMsgq `[extern]`

Module message queue of the video streaming module

CompanionComputer/src/camera_utils.c File Reference

Detailed Description

Camera utilities.

Author

Adam Csizy

Date

2021-03-19

Version

v1.1.0

Macro Definition Documentation

#define CAM_FMT_NOT_SUPPORTED 0

Unsupported camera output video coding format

#define CAM_FMT_SUPPORTED 1

Supported camera output video coding format

#define STR_CAM_OUT_FMT_H263 "video/x-h263"

Camera output video coding format: H.263

#define STR_CAM_OUT_FMT_H264 "video/x-h264"

Camera output video coding format: H.264

#define STR_CAM_OUT_FMT_H265 "video/x-h265"

Camera output video coding format: H.265

#define STR_CAM_OUT_FMT_JPEG "image/jpeg"

Camera output video coding format: JPEG

#define STR_CAM_OUT_FMT_RAW "video/x-raw"

Camera output video coding format: RAW

#define STR_CAM_OUT_FMT_UNK "unknown"

Camera output video coding format: unknown

#define STR_CAM_OUT_FMT_VP8 "video/x-vp8"

Camera output video coding format: VP8

#define STR_CAM_OUT_FMT_VP9 "video/x-vp9"

Camera output video coding format: VP9

#define STR_DEV_DIR_PATH "/dev"

Path to device directory

#define STR_FORMAT_CAPS_COMMON "\tWidth: %d\n\tHeight: %d\n\tFramerate: %d/%d\n"

Common video coding capabilities string format

#define STR_FORMAT_DEV_CAPS "V4L2 Device Capabilities\n\nDriver name:\t%s\ nDriver version:\t%d\nDevice name:\t%s\nBus info:\t%s\nCapabilities:\t%d\n"

Device capabilities string format

#define STR_FORMAT_FMT_GENERAL "\tFormat: %s\n"

General video coding prefix string format

#define STR_FORMAT_FMTS_CAPS_TITLE "Supported camera formats and capabilities:\n\n"

Video coding formats and capabilities title string format

#define STR_HINT_CAM_DEV_NAME "video"

Hint for video device name

#define STR_SIZE_DEV_PATH 16U

Size of string containing device path

Function Documentation

int getCameraDevicePath (char *cameraDevicePath*[], const size_t *size*)

Searches and validates camera device.

Searches for camera devices under '/dev' directory and returns with the path of the first compatible device entry. Validation includes V4L2 interface compatibility and Video Capture capability.

Note

None

Parameters

out	*cameraDevicePath*	Path string.
in	*size*	Size of path string.

Returns

Result of execution.

Return values

0	Success
-1	Failure

static int printDeviceCapabilities (const struct v4l2_capability * *capabilities*)`[static]`

Prints V4L2 device capabilities on the standard output.

Prints the capabilities of the given V4L2 compatible device on the standard output.

Note

Displayed capabilities indicate only the device node capabilities and not the physical device's capabilities.

Parameters

in	*capabilities*	V4L2 device capabilities.

Returns

Result of execution.

Return values

0	Success
-1	Failure

static int stringToVideoCodingFormat (const char * *string*, VideoCodingFormat_T * *format*) `[static]`

Converts string to video coding format.

Converts the given string representation of a video encoding format into a VideoCodingFormat_T representation. If encoding format cannot be identified an unknown format (CAM_FMT_UNK) is set.

Note

None

Parameters

in	*string*	String representation of the video coding format.
out	*format*	Video coding format.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int videoCodingFormatToString (const VideoCodingFormat_T *format*, char *string*[], const size_t *size*)

Converts video coding format to string.

Converts the given VideoCodingFormat_T representation of a video coding format into a string representation.

Note

None

Parameters

in	*format*	VideoCodingFormat_T representation.
out	*string*	String representation of video coding format.
in	*size*	Size of the string.

Returns

Result of execution.

Return values

0	Success
-1	Failure

CompanionComputer/src/com_utils.c File Reference

Detailed Description

Communication utilities and network module.

Author

Adam Csizy

Date

2021-04-01

Version

v1.1.0

Macro Definition Documentation

#define IDX_LOGIN_MSG_CODE 0U

Index of module message code in login message array

#define IDX_LOGIN_MSG_ID 1U

Index of drone ID in login message array

#define IDX_MSG_HEADER_CODE 1U

Index of module message code in message header array

#define IDX_MSG_HEADER_MODULE 0U

Index of module name in message header array

#define IDX_SOCK 0U

Socket index

#define LoginMessageField_T uint32_t

Type of the fields in the login netork message

#define MessageHeaderField_T uint32_t

Type of the fields in the header of network messages

#define NUM_GC_ADDR_SIZE 64U

Size of ground control address string

#define NUM_GC_PORT_SIZE 16U

Size of ground control port string

#define NUM_LOGIN_MSG_SIZE 2U

Size of login message array in LoginMessageField_T

#define NUM_MSG_HEADER_SIZE 2U

Size of message header array in MessageHeaderField_T

#define NUM_NETWORK_MSGQ_SIZE 16U

Size of network module's message queue

#define NUM_POLL_ARRAY_SIZE 1U

Size of poll array

#define RECONNECT_COOLDOWN_SEC 10U

Cooldown in seconds before initiating a reconnection to the ground control

#define RECV_LOGIN_MSG_COOLDOWN_SEC 4U

Cooldown in seconds between attempts to receive login message from ground control

#define SOCK_FD_INVAL -1

Invalid socket file descriptor

#define STR_FORMAT_MOD_MSG "\nModule Message:\n\tAddress: %d\n\tCode: %d\n"

Module message string format

#define STR_GC_ADDR_DEFAULT_LAN "195.441.0.134"

Default address of ground control (LAN)

#define STR_GC_ADDR_DEFAULT_WAN "any_custom_domain.ddns.net"

Default address of ground control (WAN)

#define STR_GC_PORT_DEFAULT_LAN "5010"

Default port of ground control (LAN)

#define STR_GC_PORT_DEFAULT_WAN "17010"

Default port of ground control (WAN)

Function Documentation

static void cleanupInputMessages (const int *sockFd*) `[static]`

Clean up input messages.

Cleans up input messages available through the given network socket file descriptor by reading network RX buffer as long as data is available.

Note

Not thread safe. In case of reading the network socket from multiple threads simultaneously mutual exclusion must be applied using static variable 'socketFDLock'.

Parameters

in	*sockFd*	Network socket file descriptor.

static int connectToGroundControl (const char * *node*, const char * *service*, int * *fd*) [static]

Estabilish connection with ground control.

Estabilishes TCP connection with a ground control node specified by the input arguments (IP, port). In case of successful connection the given socket file descriptor is set and can be used for further communication.

Note

The companion computer sends a LOGIN message code and the drone's ID. In turn it receives a LOGIN_ACK message code and the drone's ID.

Parameters

in	*node*	Network address (IP) of ground control node.
in	*size*	Service name (port) of ground control node.
out	*fd*	Socket file descriptor associated with the ground control connection.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int deinitModuleMessageQueue (ModuleMessageQueue_T * *messageQueue*)

Deinitialize module message queue.

Deinitializes module message queue object. The remaining messages in the queue and the buffer itself are freed.

Note

Not thread safe. Blocking.

Parameters

in,out	*messageQueue*	Message queue object to be deinitialized.

Returns

Result of execution.

Return values

0	Success
-1	Failure

static int gccommonMessageToNetwork (const int * *sockFd*, const ModuleMessage_T * *message*) [static]

Convert GC common module message to network data.

Converts GC (Ground Control) common module message to network data. This function parses the given modul message object and creates a byte stream which is then sent to the ground control over IP/TCP using the given network socket descriptor.

Note

Thread safe.

Parameters

in	*sockFd*	Network socket file descriptor.

in	*message*	Message object to be parsed.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int initModuleMessageQueue (ModuleMessageQueue_T * *messageQueue*, const size_t *size*)

Initialize module message queue.

Initializes module message queue object. The given message queue object must be in an uninitialized or deinitialized state. Double initialization of a message queue object might result undefined behaviour. The value of the size input argument must be the power of two because the implementation uses bitmask for efficient boundary check.

Note

Not thread safe. Blocking.

Parameters

in,out	*messageQueue*	Message queue object to be initialized.
in	*size*	Size of the message queue buffer.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int initNetworkModule (NetworkInitContext_T * *initCtx*)

Initialize network handler module.

Initializes the network handler module's message queue and starts the network traffic handler threads.

Parameters

in	*initCtx*	Initialization context.

Returns

Result of execution.

Return values

0	Success
-1	Failure

static int inputMessageHandler (const int *sockFd*) `[static]`

Handle input message.

Handles input messages received over IP/TCP from the ground control. Message handling includes parsing message code and target module name specified in the message header as well as calling the corresponding message handlers per module. On failure the network RX buffer is cleaned up to preserve consistency.

Note

Not thread safe. In case of reading the network socket from multiple threads simultaneously mutual exclusion must be applied using static variable 'socketFDLock'.

Parameters

in	*sockFd*	Network socket file descriptor.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int insertModuleMessage (ModuleMessageQueue_T * *messageQueue*, ModuleMessage_T * *message*, const int *noblock*)

Insert message into module message queue.

Inserts module message into the given module message queue (FIFO). The message object must be dynamically allocated by the source and released after insertion.

Note

Thread safe.

Parameters

in,out	*messageQueue*	Module message queue object.
in	*message*	Dynamically allocated module message object.
in	*noblock*	Flag whether the call should be non-blocking.

Returns

Result of execution.

Return values

0	Success
-1	Failure
-2	Could not lock queue (prevent blocking)
-3	Message queue is full (prevent blocking)

static int networkToStreamMessage (const int *sockFd*, const ModuleMessageCode_T *code*) `[static]`

Convert network data to stream module message.

Converts network data to stream module message. This function (optionally) reads data from the network and creates a module message object based on the given module message code and network data. On successful message object creation the object is inserted into the stream module's message queue.

Note

Not thread safe. In case of reading the network socket from multiple threads simultaneously mutual exclusion must be applied using static variable 'socketFDLock'.

Parameters

in	*sockFd*	Network socket file descriptor.
in	*code*	Stream module message code.

Returns

Result of execution.

Return values

0	Success
-1	Failure

int printModuleMessage (const ModuleMessage_T * *message*)

Print module message.

Prints module message data on the standard output.

Parameters

in	*message*	Module message object to be printed.

Returns

Result of execution.

Return values

0	Success
-1	Failure

ssize_t recvTimeout (int *sockfd*, void * *buf*, size_t *len*, int *flags*, time_t *sec*, useconds_t *usec*)

Wrapper for 'recv()' with timeout option.

This function serves as a wrapper for function 'recv()' with timeout option.

Parameters

in	*sockfd*	Socket file descriptor.
out	*buf*	Received data buffer.
in	*len*	Size of the data to be received in bytes.
in	*flags*	Flags.
in	*sec*	Timeout interval in seconds.
in	*usec*	Timeout interval in microseconds.

Returns

The number of bytes received, or -1 if an error occurred. In the event of an error, errno is set to indicate the error.

int removeModuleMessage (ModuleMessageQueue_T * *messageQueue*, ModuleMessage_T ** *message*, const int *noblock*)

Remove message from module message queue.

Removes module message from the given module message queue (FIFO). The message object must be freed by the receiver.

Note

Thread safe.

Parameters

in,out	*messageQueue*	Module message queue object.
out	*message*	Module message object pointer.
in	*noblock*	Flag whether the call should be non-blocking.

Returns

Result of execution.

Return values

0	Success

-1	Failure
-2	Could not lock queue (prevent blocking)
-3	Message queue is full (prevent blocking)

static void * threadFuncNetworkIn (void * *arg*) `[static]`

Start routine of network input handler thread.

Estabilishes connection with ground control and handles incoming network traffic over IP/TCP. Incoming messages are parsed and forwarded to the corresponding module's message queue.

Note

Critical thread. On failure connection with the ground control is lost.

Parameters

in	*arg*	Network module initialization context.

Returns

Any (not used).

static void * threadFuncNetworkOut (void * *arg*) `[static]`

Start routine of network output handler thread.

Handles outgoing network traffic over IP/TCP. Messages from the network module's message queue are parsed and forwarded to the ground control.

Note

Critical thread. On failure connection with the ground control is lost.

Parameters

in	*arg*	Launch argument (not used).

Returns

Any (not used).

Variable Documentation

ModuleMessageQueue_T networkMsgq

Module message queue of the network module

pthread_mutex_t socketFdLock = PTHREAD_MUTEX_INITIALIZER `[static]`

Mutex to protect network socket

pthread_t threadNetworkIn `[static]`

Thread object for handling network input

pthread_t threadNetworkOut `[static]`

Thread object for handling network output

CompanionComputer/src/log_utils.c File Reference

Detailed Description

Logging utilities.

Author

Adam Csizy

Date

2021-03-19

Version

v1.1.0

Function Documentation

void createLogMessage (const char *message*[], const LogSeverity_T *severity*)

Create log message.

Generates a log entry in the system log and prints the log message on the standard output with the given severity level if debug mode is enabled.

Note

None

Parameters

in	*message*	Log message.
in	*severity*	Log severity level.

CompanionComputer/src/main.c File Reference

Detailed Description

Main program module.

Author

Adam Csizy

Date

2021-03-04

Version

v1.1.0

Macro Definition Documentation

#define STR_SYSLOG_PROG_NAME "DroneVideoStreamer"

Program's name in the system logger

Function Documentation

int main (int *argc*, char * *argv*[])

The application's main function.

The application's main function is responsible for program module initializations.

Returns

Result of execution.

Return values

0	Success
<0	Failure

CompanionComputer/src/stream_utils.c File Reference

Detailed Description

Streaming utilities module.

Author

Adam Csizy

Date

2021-04-05

Version

v1.1.0

Macro Definition Documentation

#define NUM_CAM_DEV_PATH_SIZE 64U

Camera device path size

#define NUM_STREAM_DEST_PORT 17000

Default service port of RTP stream destination (WAN)

#define NUM_STREAM_EVENT_NUM 4U

Number of stream events

#define NUM_STREAM_MSGQ_SIZE 8U

Size of streaming module's message queue

#define NUM_STREAM_STATE_NUM 2U

Number of stream states

#define NUM_UDP_MTU 64000

MTU for UDP packets in bytes. Theoretical ceiling is 64kB but GStreamer payloaders might not support such a high value.

#define PIPE_INITIAL_STATE GST_STATE_READY

Initial state of the video streaming pipeline

#define SM_UPDATE_NOT_REQUIRED 0U

State machine update not required

#define SM_UPDATE_REQUIRED 1U

State machine update required

#define STR_PIPE_ELEM_NAME_CAPSFLTR "Video_Caps_Filter"

Name of the video capabilities-filter pipeline element

#define STR_PIPE_ELEM_NAME_ENCODER "Video_Encoder"

Name of the video encoder pipeline element

#define STR_PIPE_ELEM_NAME_NETSINK "Network_Sink"

Name of the network sink pipeline element

#define STR_PIPE_ELEM_NAME_PAYLDR "Payloader"

Name of the payloader pipeline element

#define STR_PIPE_ELEM_NAME_VIDCONV "Video_Converter"

Name of the video converter pipeline element

#define STR_PIPE_ELEM_NAME_VIDSRC "Video_Source"

Name of the video source pipeline element

#define STR_STREAM_DEST_ADDR "any_custom_domain.ddns.net"

Default address of RTP stream destination (WAN)

#define STR_STREAM_DEST_PORT "17000"

Default service port of RTP stream destination (WAN)

Enumeration Type Documentation

enum StreamEvent

Enumeration of video streaming events.

Enumerator:

STREAM_EVENT_STREAM_REQ	Ground control requested video stream (type)
STREAM_EVENT_STREAM_START	Ground control requested to start video stream
STREAM_EVENT_STREAM_STOP	Ground control requested to stop video stream
STREAM_EVENT_PIPE_ERROR	Error occured in streaming pipeline

enum StreamState

Enumeration of video streaming states.

Enumerator:

STREAM_STATE_STANDBY	Pipeline in standby state

STREAM_STATE_PLAYING	Pipeline in playing state

Function Documentation

static void * emptyHandler (ModuleMessage_T ** *message*, GstElement ** *pipeline*) [static]

Empty event handler.

Event handler for events which require no action. The given module message is freed.

Parameters

in,out	*message*	Module message.
in,out	*pipeline*	GStreamer video streaming pipeline.

Returns

Any (not used).

static int initCameraCapabilities (const char * *camDevPath*, VideoCodingFormatContext_T * *initCtx*) [static]

Initializes camera capabilities.

Initializes camera capabilities array in the given initialization context with capabilities of the camera device under path 'camDevPath'.

Note

GStreamer core and plugins must be initialized using 'gst_init()' before invoking this function.

Parameters

in	*camDevPath*	Path to camera device.
in	*initCtx*	Capabilities's initialization context.

Returns

Result of execution.

Return values

0	Success
-1	Failure

static void initStreamController (StateContext_T *controller*[NUM_STREAM_STATE_NUM][NUM_STREAM_EVENT_NUM]) [static]

Initialize stream controller.

Initializes the given stream controller state machine.

Parameters

in	*controller*	Stream controller to be initialized.

int initStreamModule (void)

Initialize streaming module.

Initializes the streaming module's message queue and starts the streaming control thread.

Returns

Result of execution.

Return values

0	Success
-1	Failure

static int pipeBuilder (GstElement ** *pipeline*, const char * *camDevPath*, const VideoCodingFormat_T *codingFormat*, const VideoCodingFormatCaps_T *caps*[NUM_SUP_VID_COD_FMT]) [static]

Build media pipeline.

Builds a GStreamer media pipeline compatible with the given video coding format as the camera device's output. The given capabilities argument contains optimal capability configurations for each supported video coding format and is used to enhance the video stream quality. The video stream is forwarded over UDP/RTP to the ground control.

Note

GStreamer core and plugins must be initialized using 'gst_init()' before invoking this function.

Parameters

in,out	*pipeline*	Pointer to a pipeline to be built.
in	*camDevPath*	Path to camera device.
in	*codingFormat*	Video encoding format.
in	*caps*	Video coding capabilities.

Returns

Result of execution.

Return values

0	Success
-1	Failure

static void pipelineEosCallback (GstBus * *bus*, GstMessage * *message*, gpointer *data*) [static]

Pipeline end-of-stream (EOS) signal callback.

Callback function for handling end-of-stream signals. On EOS signal a log record is created and the stream controller is notified using the streaming module's message queue.

Parameters

in,out	*bus*	Pipeline's bus.
in	*message*	Pipeline EOS message.
in	*data*	Custom data passed to the callback function (not used).

static void pipelineErrorCallback (GstBus * *bus*, GstMessage * *message*, gpointer *data*) [static]

Pipeline error signal callback.

Callback function for handling pipeline error signals. On pipeline error a log record is created and the stream controller is notified using the streaming module's message queue.

Parameters

in,out	*bus*	Pipeline's bus.
in	*message*	Pipeline error message.
in	*data*	Custom data passed to the callback function (not used).

static void pipelineStatechangedCallback (GstBus * *bus*, GstMessage * *message*, gpointer *data*) `[static]`

Pipeline state-changed signal callback.

Callback function for handling the pipeline's state changed signal. On state changed event a log record is created. This callback function only handles state changed messages coming from the pipeline itself. For such filtering a pointer to the pipeline is given to detect the appropriate message source.

Note

Intended for debug purposes.

Parameters

in,out	*bus*	Pipeline's bus.
in	*message*	Pipeline state changed message.
in	*data*	Pointer to the pipeline.

static int registerCallbackFunctions (GstElement * *pipeline*) `[static]`

Register signal callback functions.

Registers signal callback functions at the given GStreamer pipeline's bus to handle the occurance of different events coming from the pipeline elements. By invoking this function a bus signal watch is also being added to the default main context.

Note

On cleanup 'gst_bus_remove_signal_watch()' should be called to remove the signal watch from the main context.

Parameters

in,out	*pipeline*	GStreamer pipeline.

Returns

Result of execution.

Return values

0	Success
-1	Failure

static void * streamErrorHandler (ModuleMessage_T ** *message*, GstElement ** *pipeline*) `[static]`

Stream error event handler.

Event handler for stream error events. On errors coming from the GStreamer pipeline elements the video streaming is stopped and the pipeline is set back to NULL state resulting an internal state reset for each pipeline component. The given module message is forwarded to the ground control over the network module.

Parameters

in,out	*message*	Module message.
in,out	*pipeline*	GStreamer video streaming pipeline.

Returns

Any (not used).

static void * streamRequestHandler (ModuleMessage_T ** *message*, GstElement ** *pipeline*) `[static]`

Stream request event handler.

Event handler for stream request events. The stream's video coding format is sent back to the ground control and the given module message is freed. The 'port' property of the network sink element is set according to the request message data.

Parameters

in,out	*message*	Module message.
in,out	*pipeline*	GStreamer video streaming pipeline.

Returns

Any (not used).

static void * streamStartHandler (ModuleMessage_T ** *message*, GstElement ** *pipeline*) `[static]`

Stream start event handler.

Event handler for stream start events. Starts the video streaming and frees the given module message.

Parameters

in,out	*message*	Module message.
in,out	*pipeline*	GStreamer video streaming pipeline.

Returns

Any (not used).

static void * streamStopHandler (ModuleMessage_T ** *message*, GstElement ** *pipeline*) `[static]`

Stream stop event handler.

Event handler for stream stop events. Stops the video streaming and sets the pipeline back to its initial state. The given module message is freed.

Parameters

in,out	*message*	Module message.
in,out	*pipeline*	GStreamer video streaming pipeline.

Returns

Any (not used).

static void * threadFuncStreamControl (void * *arg*) `[static]`

Start routine of stream controller thread.

Initializes and controls video streaming media pipeline on a state machine basis. This function also responsible for handling the streaming module's message traffic

Parameters

in	*arg*	Launch argument (not used).

Returns

Any (not used).

static void * threadFuncStreamMainLoop (void * *arg*) [static]

Start routine of stream main loop thread.

Initializes and starts a GMainLoop object using the default context. In terms of the video streaming application the main loop is responsible for periodically checking the pipeline's bus and emitting the asynchronous message signals. The registered signal callback functions are invoked in the main loop's thread context (i.e. in this thread context).

Parameters

in	*arg*	Launch argument (not used).

Returns

Any (not used).

Variable Documentation

VideoCodingFormat_T currentCodingFormat = CAM_FMT_UNK [static]

Current coding format used by the video streaming pipeline

ModuleMessageQueue_T streamMsgq

Module message queue of the video streaming module

pthread_t threadStreamControl [static]

Thread object for handling video stream state machine

pthread_t threadStreamMainLoop [static]

Thread object for handling main loop context of the video stream