hamas Documentation

Release

Manoel Brunnen

Sep 14, 2017

Contents:

This package provides a framework for building a Multi-Agent System in a domestic area. The agents can be easily hosted on different platforms and still communicate seamless with each other. The communication channels are typical for the in-house use, like WLAN or ZigBee.

CHAPTER 1

The Agents

The agent module implements the base *Agent* class. All agents should be derived from this class. The agents can have special methods, which can be called by other agents and are decorated with *provide()*. To handle their conversations they have with other agents they use the *ConversationRegister*

class hamas.agents.**Agent**(*mts*, *aid*)

Bases: object

The base class for all other agents. An instance is already able to communicate to other instances. In particular, it uses a *ConversationRegister* and a MessageTransportSystem for the connunication with other agents.

Parameters

- **mts** (MessageTransportSystem) – The *MessageTransportSystem* used by this agent.
- **aid** (*str*) – The agent's unique identifier.

aid

str – The unique agent identifier. Composed of the platform name and an agent name.

am_aid

str – The agent identifier of the AgentManager.

get_aid()

coroutine get_reply(*message*, *timeout=5*)

Opens a conversation and returns directly the reply

Sometimes it is more useful to get the reply instead of the future.

Parameters

- **message** (Message) – The request message to open the conversation.
- **timeout** (*float*) – The amount of time the requesting agent is willing to wait

Returns

Return type reply (Reply,list)

open_conversation(*message*, *timeout=5*)
This method transmits a request and returns a reply future.

It creates a Future with the reply of the addressed agent. When the addressed agent replies, the future's result will be set. If the timeout is reached before any reply, an timeout exception will be thrown If the request is sent as broadcast, the communication will set the result with the received replies before the timeout after a certain timeout. This is a coroutine because it has to wait for send to finish.

Parameters

- **message** (Message) – The conversation opener, i.e. a remote process call
- **timeout** (*float*) – After this amount of time the conversation times out

Returns The reply future contains the reply of the the call

Return type reply (Future)

platform_name
str – The first part of the agent's indentifier, the platform name

coroutine receive(*message*)

coroutine remote_process_call(*function*, **args*, *recipient=None*, *timeout=5*, *routing='unicast'*, ***kwargs*)

coroutine send(*message*)
Send a message from this agent to another

Parameters **message** (Message) – The message for the recipient

coroutine send_reply(*content*, *message*, *performative=None*)

class hamas.agents.**ConversationRegister**
Bases: object

A dictionary for storing the conversations. When a new conversation is created, by calling *new_conversation()*, a new asyncio.Queue is created and a asyncio.Future is returned. Every message has a conversation identifier, so the agent can group all messages with the same conversation ID in this register. The agent can *put()* new messages in this register, but it can also *get()* a message when it has time to process a message. When the agent considers a conversation as finished, it can set a result for this conversation in the corresponding *future*.

_queues
dict – A dictionary with the conversation ID *conv_id* as key and a asyncio.Queue as value.

_futs
dict – A dictionary with the conversation ID *conv_id* as key and a asyncio.Future as value.

__contains__(*conv_id*)
conv_id in self returns *True* or *False*, whether *conv_id* is an active conversation or not.

__len__()
len(self) returns the quantity of unfinished conversations.

coroutine get(*conv_id*)
Remove and return an item from the queue. If queue is empty, wait until an item is available.

Parameters **conv_id** (*bytes*) – The key which allows access to the conversation.

Returns

The incoming message associated with this conversation.

Return type msg(*Message*)

new_conversation(*conv_id*)
Allocate a new conversation including a asyncio.Queue and a asyncio.Future.

Parameters **conv_id** (*bytes*) – The key which allows access to the conversation.

coroutine put(*conv_id*, *msg*)
Put a new message in the register.

Parameters

- **conv_id** (*bytes*) – The key which allows access to the conversation.
- **msg** (Message) – The incoming message associated with this conversation.

set_result(*conv_id*, *result*)

hamas.agents.**provide**(*func*)
provide() acts as function decorator and is used for methods, that the agent provides as a service to other agents.

CHAPTER 2

Agent Management

The agent management module contains the two basic classes for the management of the agents: The *AgentManager* and the *AgentPlatform*. The *AgentManager.create()* classmethod is used to initiate the whole system, including the *AgentManager* and the *AgentPlatform*.

Agent Manager

class hamas.management.**AgentManager**(*platform*, **args*, ***kwargs*)

Bases: *hamas.agents.Agent*

Initial *hamas.agents.Agent* running which creates, runs and manages the agents.

Parameters

- **has_platform** (*bool*) – True if the PlatformConnector should be used.
- **has_zigbee** (*bool*) – True if the ZigBeeConnector should be used.
- **has_mqtt** (*bool*) – True if the MqttConnector should be used.
- **has_uds** (*bool*) – True if the class:*UnixConnector* should be used.

_white_pages

dict – Dictionary which contains the agent description, actually the class name.

_platform

AgentPlatform – The platform, on which the *AgentManager* is managing the agents.

classmethod create(*loop*, *config*)

Factory function which instantiates a AgentManager agent.

Do not instantiate the AgentManager directly. It is a special agent, which has no ID given, so it will create it's own ID. Also it will create a hamas.MessageTransportSystem.

Parameters

- **loop** (*BaseEventLoop*) – The event loop.

- **config** (Configuration) – A configuration, which is an instance of hamas.Configuration.

Returns AgentManager

create_agent(*agent_class*, **args*, ***kwargs*)
Create an agent

Parameters

- **agent_class** (*class*) – Passes the class of the required Agent
- **args** (*list*) – Passes the arguments to the constructor of the agent class
- **kwargs** (*dict*) – Passes the keyword arguments to the constructor of
- **agent class** (*the*) –

destroy_agent(*aid*)

get_agents(*agent_class_names=None*)

other_platforms

perform_create_agent(*agent_class_name*)

perform_destroy_agent(*aid*)

coroutine start()
This method starts all the tasks of the multi-agent system. This includes sending network discovery beacons.

stop()
Stops all the tasks of the multi-agent system.

coroutine wait_for_zigbee()
This couroutine blocks until other participants are found in the ZigBee network and is used for testing purposes.

white_pages
dict – Dictionary which contains the agent description, actually the class name.

Agent Platform

class hamas.management.**AgentPlatform**(*loop*, *name*, *has_platform*, *has_zigbee*, *has_mqtt*, *has_uds*, *regex*, *broker*, *update_interval=60*)

Bases: object

The *AgentPlatform* contains all the elements of a multi-agent system. This includes the message transport system and all the agents.

Parameters

- **loop** (*asyncio.BaseEventLoop*) – The loop in which the platform should run.
- **name** (*str*) – The unique name of the platform.
- **has_platform** (*bool*) – True if the PlatformConnector should be used.
- **has_zigbee** (*bool*) – True if the ZigBeeConnector should be used.
- **has_mqtt** (*bool*) – True if the MqttConnector should be used.
- **has_uds** (*bool*) – True if the class:*UnixConnector* should be used.

- **regex** (*str*) – The device path of the ZigBee module.
- **broker** (*str*) – The address of the MQTT broker.
- **update_interval** (*int*, *float*) – The interval of updating the *MessageTransportSystem*.

agents

create_agent(*agent_class*, **args*, ***kwargs*)
Create an agent

Parameters

- **agent_class** (*class*) – Passes the class of the required Agent
- **args** – Passes the arguments to the constructor of the agent class.
- **kwargs** – Passes the keyword arguments to the constructor of the agent class

destroy_agent(*aid*)

loop
BaseEventLoop – The `asyncio.BaseEventLoop` in which the coroutines of the application will run.

name
str – The name of the platform.

coroutine start()

stop()

CHAPTER 3

Messsage Transport

Connectors

Communication Connectors

The Connector Base Class

Abstract base class for connectors.

class hamas.transport.connectors.connector.**Connector**
Bases: abc.ABC

Base Class for connectors

address

broadcast(*message*)

other_platforms

unicast(*platform_name*, *message*)

The Local Connector

Communication interface for agents on the same platform.

class hamas.transport.connectors.local_connector.**LocalConnector**(*platform*)
Bases: object

Connector class for communication on the same platform.

coroutine broadcast(*message*)

Parameters **message** –

Raises InterfaceError – Raises if the list of recipients is empty.

local_aids

other_agents(*me*)

coroutine unicast(*aid*, *message*)

The Platform Connector

Connector for multiple logical platforms on one physical machine.

class hamas.transport.connectors.platform_connector.**PlatformConnector**(*mts*)
Bases: *hamas.transport.connectors.connector.Connector*

Connector for connecting multiple agent platforms on the same physical machine. :param mts: The calling MessageTransportSystem. :type mts: MessageTransportSystem

address

coroutine broadcast(*message*)

other_platforms

coroutine unicast(*platform_name*, *message*)

hamas.transport.connectors.uds_connector module

Connector for multiple unix platforms running in parallel

class hamas.transport.connectors.uds_connector.**UnixConnector**(*mts*)
Bases: *hamas.transport.connectors.connector.Connector*

Connector for agent communication on the same computer.

Parameters **mts** (MessageTransportSystem) – The calling MessageTransportSystem.

__contains__(*platform_name*)
The *UnixConnector* implements a membership test. *platform_name in self* returns *True* if *platform_name* is a reachable address by using this connector *UnixConnector* or *False* otherwise.

address
str – The unique address of this connector.

coroutine broadcast(*message*)

other_platforms
list(str) – Returns a list of addresses, which are reachable with the *UnixConnector*.

coroutine start()

stop()

coroutine unicast(*platform_name*, *message*)

hamas.transport.connectors.zigbee_connector module

Connector to control to XBee-ZigBee modules.

This corresponds to the data link layer in the OSI model. Frames are sent from one node to another. Multiple frames can be part of one message. The purpose of this module is to fragment messages in fractions, regarding the MTU, and to send them. Also received frames will be assembled to a message and be delivered to the message transport system. Another Function will be a integrity check.

class hamas.transport.connectors.zigbee_connector.**ZigBeeConnector**(*loop*, *platform_name*, *regex*, *baud=230400*, *callback=None*, *serial_timeout=5*, *ucast_timeout=5*)

Bases: *hamas.transport.connectors.connector.Connector*

Class which wraps the ZigBee class in the xbee package.

_serial
Serial – Serial communication port.

_loop
BaseEventLoop – Asyncio event loop.

_zigbee
ZigBee – The XBee module with ZigBee functionality.

_response_futs
dict – The futures handling incoming response frames

_serial_timeout
float – Time that will be waited for another frame from the xbee module to arrive.

_unicast_timeout
float – Time that will be waited for another fraction to arrive

coroutine _frame_received(*frame*)
Appending new messages.

This method is called when frame arrives. It will simply distribute the frame to the right queue. If no queue with the same frame ID exists, it will be created. The method should be called thread-safe, that means it is in the same thread as the event loop, the main thread.

Parameters **frame** (*dict*) – The arriving frame.

_generate_frame_id()
Create an approximately unique frame ID between 1-255.

Only if an Frame ID is set and is not zero, the xbee module will send an feedback frame.

_receive_cb(*data*)
XBee callback function for new messages.

This method is called by another thread, than the main thread where the event loop resides. To use the message futures in the event loop, we have to set the results of the futures in the same thread as the event loop, the main thread. Therefore, the data must be passed to the event loop in the main thread. This is done by using the run_coroutine_threadsafe method.

address

coroutine broadcast(*message*)

coroutine broadcast_fraction(*fraction*, *hops=None*)

coroutine close_ports()

static compose_address(*long_address*, *short_address=None*)
Convert the addresses, readable by xbee to a uniform form

Parameters

- **long_address** – A 8 byte long byte string, e.g. b'Þ¾ïB'
- **short_address** – A 2 byte long byte string, e.g. b'ïB'

Returns For example 'DE:AD:BE:EF:42:00:00:00!EF:42'

Return type address_string

deliver(*serialized_message*)

mtu

coroutine open_ports()

other_platforms

classmethod parse_address(*address_string*)

platform_name

coroutine send_command(*address=None*, ***kwargs*)
Send a zigbee related AT command

Supported AT commands are for example ND (Network discovery) and NP (Maximum payload size per frame in bytes).

Parameters **address** (*str*) – For example 'DE:AD:BE:EF:42:00:00:00!EF:42' or 'DE:AD:BE:EF:42:00:00:00' the module does not response in time.

Returns The response of the XBee module with e.g. the transmission status.

Return type response(dict)

coroutine send_fraction(*address*, *fraction*)
Send a message with low level transmission control

Parameters

- **address** (*str*) – The recipient's address
- **fraction** (Fraction) – The fraction to be transmitted

coroutine start()
Start the ZigBeeConnector

Start the send queue handler and make a network discovery.

stop()
Cancel running coroutines.

coroutine unicast(*platform_name*, *message*)

Parameters

- **platform_name** –
- **message** –

Raises TransmissionError – When every port unavailable or the transmission failed, an error is thrown.

coroutine update_others()
Get the other participants in the network.

coroutine wait_for_others()

Ports

These are the TCP-like Ports. The ZigbeeConnector has the unicast ports 0-254 and the the broadcast port 255 respectively 0xff. The unicast ports have real transmission control, whereas the broadcast port has no transmission control. Enabling broadcasting on the transmission controlled ports would add a lot of complexity and would make them potentially less reliable. Furthermore, broadcasting is in our use case only for short messages and is considered unreliable. It suffices if any other platform receives the message and it is unnecessary to ensure that all other platforms receive the broadcast. The broadcast port is also used for network discovery, that means to map the platform name to the ZigBee address. The unicast ports are implemented as Moore automata, i.e. the outputs are defined by the states.

class hamas.transport.connectors.ports.**Port**(*number*, *zigbee_connector*, *timeout*, *mtu*)
Bases: object

coroutine _syn_sent()
Checking if the other port accepts data and send the size of the data

Raises

- PortError – When the other Port is occupied, try another Port.
- TransmissionError – Transmission failed.

coroutine close()
Close the port nicely.

Setting the port to the initial state, by waiting for the transmission to finish. In the closed state are no pending coroutines.

is_free()

number

coroutine open()
Open the port so it's listening.

coroutine receive(*source_address*, *fraction*)

coroutine send(*dest_address*, *message*)
Open the port for sending a message. :param dest_address: The zigbee address of the receiver. :type dest_address: str :param message: The message to be send. :type message: bytes

Raises TransmissionError – When the port is in use or an error while sending occurred.

stop()
Stop running coroutines.

hamas.transport.contents module

Class for the payload in the messages.

This class represent the application layer of the agent communication protocol. Whereas the message class can be seen as the envelope, this class can be seen as the letter inside the envelope. The payload is not important for the delivery of the message, but is used by the receiving Agent itself.

class hamas.transport.contents.**Content**
Bases: *hamas.transport.serializable.Serializable*

class hamas.transport.contents.**DictionaryContent**(*dictionary*)
Bases: *hamas.transport.contents.Content*

class hamas.transport.contents.**RemoteProcess**(*function*)
Bases: *hamas.transport.contents.Content*

function

class hamas.transport.contents.**RemoteProcessCall**(*function*, *args*, *kwargs*)
Bases: *hamas.transport.contents.RemoteProcess*

args

kwargs

class hamas.transport.contents.**RemoteProcessReply**(*function*, *returns*)
Bases: *hamas.transport.contents.RemoteProcess*

returns

class hamas.transport.contents.**StringContent**(*string*)
Bases: *hamas.transport.contents.Content*

hamas.transport.fractions module

Fractions are the units on the transport layer.

class hamas.transport.fractions.**Fraction**(*port*, *flag*, *seq_id=None*, *sdu=None*)
Bases: object

static assemble_msg(*fractions*)
Assemble a serialized message out of fractions.

The sequence ID can start at any number and can be overlapped, thus it needs to be passed as argument.

Parameters **fractions** (*list*) –

Returns The assembled, serialized message.

Return type msg (bytes)

classmethod deserialize(*serialized*)
Disassemble a single fraction.

Parameters **serialized** (*bytes*) –

classmethod disassemble(*port*, *msg*, *mtu*)
Prepare a message for sending.

A agent to agent message is probably too long, so it needs to be separated in fractions.

Parameters

- **port** (*int*) – The used port on both sides
- **msg** (*bytes*) – The message which will be separated in fractions.
- **mtu** (*int*) – The maximum transmission unit

Returns the ready to send PDUs

Return type pdus (list)

flag

flag_decodings = {b'\x01': 'SYN-ACK', b'\x00': 'SYN', b'\x02': 'DATA', b'\x05': 'URL', b'\x04': 'RST', b'\x03': 'FI

flag_encodings = {'RST': b'\x04', 'SYN-ACK': b'\x01', 'FIN': b'\x03', 'URL': b'\x05', 'DATA': b'\x02', 'SYN': b'\x0

flag_fmt = 'c'

static get_port (*serialized*)

header_fmt = '>BHc'

classmethod header_len ()

classmethod max_ports ()

classmethod max_sdu_len (*mtu*)

classmethod max_size (*mtu*)

classmethod num_fractions (*msg*, *mtu*)

port

port_fmt = 'B'

sdu

seq_id

seq_id_fmt = 'H'

serialize ()

hamas.transport.message_transport module

class hamas.transport.message_transport.**MessageTransportSystem**(*platform*, *has_platform*, *has_zigbee*, *has_mqtt*, *has_uds*, *regex*, *broker*, *update_interval=60*)

Bases: object

Provides the communication between agents.

Parameters

- **platform** (AgentPlatform) – The agent platform on which the message transport system resides.
- **has_platform** (*bool*) – Define if a PlatformConnector should be initialised.
- **has_zigbee** (*bool*) – Define if a ZigBeeConnector should be initialised.
- **has_mqtt** (*bool*) – Define if a MqttConnector should be initialised.
- **has_uds** (*bool*) – Define if a UnixConnector should be initialised.
- **regex** (*str*) – A regular expression to find the ZigBee device, if needed.
- **broker** (*str*) – The address of the MQTT broker.
- **update_interval** (*int*, *float*) – Define in what interval the connectors should update their network.

_init_mqtt_conn (*broker*)
Initiate the MQTT Connector.

_init_platform_conn ()
Initiate the platform connector.

_init_unix_conn()
Initiate the unix domain socket connector.

_init_zigbee_conn(*regex*)
Initiate the ZigBee Connector.

coroutine _unicast(*message*, *recipient*)
Send a message to one specific agent.

loop

other_platforms

classmethod parse_aid(*aid*)

platform_name

coroutine receive(*message*)

coroutine send(*message*)
This method checks the routing option.

This is a coroutine because it has to wait for the xbee module in another thread.

coroutine start()

stop()

coroutine wait_for_zigbee()

hamas.transport.messages module

The classes that agents can send to each other

A message is mapped to the transport and session layer of the agent communication protocol. The sender, recipient and routing options are used to deliver the message. The content field is used by the receiving agent. The message class is the envelope and the content is the letter. The request class is necessary to open a session, which is called conversation. A conversation are all messages with the same conversation ID. The normal use case is that an agent sends a Request and the receiving agent will answer with an instance of Reply. The platform will then match these two messages and establish in that way a conversation.

class hamas.transport.messages.**Message**(*sender*, *content=None*, *recipient=None*, *routing='unicast'*, *conversation_id=None*, *performative=None*)
Bases: *hamas.transport.serializable.Serializable*

The object that agents sends to other agents.

The sending agent doesn't expect any reply.

recipient
str, None – The recipient

_sender
str – The sender

content
Content – The content data

_routing
str – The routing scheme of the message

__conversation_id
bytes – Control of conversation. This number shall be unique for one conversation. Also this field can only be read.

conversation_id

classmethod deserialize(*serialized*, **args*)

performative

recipient

routing
Returns the routing scheme of the message.

Unicast: To one defined recipient. Broadcast: To all agents.

sender
Returns the AID of the sender

hamas.transport.serializable module

Serializable objects like messages and Payloads

Classes which inherits from this class have to overwrite _get_init_args to be serializable.

class hamas.transport.serializable.**Serializable**
Bases: abc.ABC

Objects which can be transformed to a bytes object

_get_init_args()
Helper function for the serialization

Returns the positional constructor arguments as tuple

Return type *args* (tuple)

classmethod deserialize(*serialized*)
Deserialize a bytes object to a Content object

Parameters **serialized** (*bytes*) – The serialized object.

Returns Returns an instance of the class Content

Return type payload(*Content*)

pickle()

serialize()
Serialize this instance to a bytes object

Returns The serialized object.

Return type serialized (bytes)

serializers = {b'\x01': 'hamas.transport.contents.RemoteProcessCall', b'\x04': 'hamas.transport.contents.Dictionary

static unpickle(*serialized*)

CHAPTER 4

Configuration

This package is configured through a special configuration file, usually located in *configuration/hamas.conf*. The user can set the environment variable *HAMAS_CONFIG* to define the location of the configuration file.

The file is formatted as defined in `configparser`. It consists of one general section and a ZigBee specific section. The general section contains information about the machine name, the location of the logging configuration file and which communication protocols shall be used. In the ZigBee section is defined on which port the ZigBee module is connected to the machine.

HAMAS_CONFIG

If the *HAMAS_CONFIG* environment variable is set, the file specified in the variable will be used instead of the default one. On a Linux system the environment variable is set as follows:

```
export set HAMAS_CONFIG=~/.my_hamas.conf
```

Example

A configuration file can look like this:

```
; A exemplary configuration file
[General]
machine_name = local
use_platform = false
use_zigbee = true
use_mqtt = true
use_uds = false

[ZigBee]
device = /dev/ttyUSB

[MQTT]
broker = localhost
; vim:ft=dosini
```

class hamas.configuration.**Configuration**(*conf_file=None*)

Bases: object

The *Configuration* class parses the configuration file and contains the retrieved information. The AgentManager.create() accept a instance of this class and will build the multi-agent system according to the given configuration. When importing the hamas package, a default configuration will be created: def_config. This default configuration is either created from the default configuration file or the one given by *HAMAS_CONFIG*. This default configuration will be used by the AgentManager.create()

broker
str – Returns the address of the MQTT broker.

device
str – Returns the file descriptor of connected ZigBee hardware.

file
str – Returns the path of the configuration file.

machine_name
str – Returns the machine name specified in the configuration file.

use_mqtt
bool – Returns if MQTT shall be used for communication.

use_platform
bool – Returns if the platform connector shall be used for communication.

use_uds
bool – Returns if the Unix Domain Socket shall be used for communication.

use_zigbee
bool – Returns if ZigBee shall be used for communication.

CHAPTER 5

Miscelleanous

Exceptions

Exceptions used by hamas

exception hamas.exceptions.**AgentError**(**args*, ***kwargs*)
Bases: *hamas.exceptions.HamasError*

Raised if an agent fails

exception hamas.exceptions.**ConnectorError**
Bases: FileNotFoundError, *hamas.exceptions.HamasError*

Raised when a connector is not responding.

exception hamas.exceptions.**DeviceHandlerError**(**args*, ***kwargs*)
Bases: *hamas.exceptions.HamasError*

Raised on device handler specific errors

exception hamas.exceptions.**HamasError**(**args*, ***kwargs*)
Bases: Exception

Base Class for all exceptions defined by hamas

exception hamas.exceptions.**MarketError**(**args*, ***kwargs*)
Bases: *hamas.exceptions.HamasError*

Raised on market specific errors

exception hamas.exceptions.**PortError**(**args*, ***kwargs*)
Bases: *hamas.exceptions.HamasError*

Raised, when a port is not ready to communicate.

exception hamas.exceptions.**TransmissionError**(**args*, ***kwargs*)
Bases: *hamas.exceptions.HamasError*

Raised if a transmission over the air or cable fails

Logging

Implement the HAMAS Logger.

class hamas.logger.**CSVFilter**(*context*)
Bases: object

filter(*record*)

class hamas.logger.**PatternFilter**(*patterns*)
Bases: object

filter(*record*)

hamas.logger.**config_logger**(*logger_config_file=None*)
Call this function to enable a fully configurable logger for this package.

Kwargs: logger_config_file (str): Path of logging configuration file.

Helper Functions

Useful functions

hamas.utils.**bytes2hexstr**(*bytestr*)
Convert bytes, e.g. a parameter, to a human readable string.

For example, normally b'w' will be printed as b'w', which is not wanted for hexadecimal addresses. To avoid this, this method transforms the bytes object b'w' to the string '77'

Parameters **bytestr** (*bytes*) – A bytestring with hexadecimal values.

Returns A string containing the hexdecimal numbers as string, e.g. b'J' will become '4A'

Return type hexstring(str)

hamas.utils.**bytes2uint**(*bytestring*)

hamas.utils.**hexstr2bytes**(*hexstring*)
Convert a hexstring like '04' to b"'

hamas.utils.**int2bytes**(*i*)
Convert an int to a byte, like 255 to b'ÿ'

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

Symbols

A

B

C

D

E

F

G

H

S

T

U

W

Z