Coinbits Documentation

Release 0.3

Brian Muller

February 28, 2015

Contents

Note: This library assumes you have a working familiarity with the Bitcoin Protocol.

Coinbits provides the basic serialization / deserialization code necessary to operate as a peer on the Bitcoin network. Many utilities are provided to help with buffering input, creating transactions, and key management.

This library could be used to do any of the following things easily:

- To create a full Peer node that accepts and validates transactions, stores blocks, and responds to inventory requests
- To query blocks from nodes on the network
- To map the bitcoin network, asking each peer for a list of peers, then those peers for peers, etc.

Basically, anything that requires interaction on the P2P network could utilize this library.

CHAPTER 1

Quick Example

Coinbits includes a basic client example for interacting on the peer-to-peer network. Here's an example of a client that extends `BitcoinClient` and requests information on a block hash:

```
from coinbits.client import BitcoinClient
from coinbits.protocol.serializers import GetBlocks


class MyClient(BitcoinClient):
    def message_received(self, message_header, message):
        print "Got a message:", message_header.command, message
        super(MyClient, self).message_received(message_header, message)

    def send_message(self, message):
        print "Sending a message:", str(message)
        super(MyClient, self).send_message(message)

    def connected(self):
        hash = int('00000000000000000f69e991ee47a3536770f5d452967ec7edeb8d8cb28f9f28', 16)
        gh = GetBlocks([hash])
        self.send_message(gh)

    def handle_inv(self, message_header, message):
        print "Got some inventory:", message

MyClient("bitcoin.sipa.be").loop()
```

The `connected` method will be called as soon as the client has connected and finished handshaking. The `serializers` module contains all of the messages that can be serialized on the network, like the `GetBlocks` message command (described here). In this case, the send_message and message_received methods have been overwritten just for debugging. The handle_inv method is an example of the method dispatch - any message command type can have an associated handle_* method that will be called whenever a message of that type is received.

CHAPTER 2

Getting Started

2.1 Installation

The easiest (and best) way to install coinbits is through pip:

```
$ pip install coinbits
```

2.2 Quick Example

Coinbits includes a basic client example for interacting on the peer-to-peer network. Here's an example of a client that extends `BitcoinClient` and requests information on a block hash:

```
from coinbits.client import BitcoinClient
from coinbits.protocol.serializers import GetBlocks


class MyClient(BitcoinClient):
    def message_received(self, message_header, message):
        print "Got a message:", message_header.command, message
        super(MyClient, self).message_received(message_header, message)

    def send_message(self, message):
        print "Sending a message:", str(message)
        super(MyClient, self).send_message(message)

    def connected(self):
        hash = int('00000000000000000f69e991ee47a3536770f5d452967ec7edeb8d8cb28f9f28', 16)
        gh = GetBlocks([hash])
        self.send_message(gh)

    def handle_inv(self, message_header, message):
        print "Got some inventory:", message

MyClient("bitcoin.sipa.be").loop()
```

The `connected` method will be called as soon as the client has connected and finished handshaking. The `serializers` module contains all of the messages that can be serialized on the network, like the `GetBlocks` message command (described here). In this case, the send_message and message_received methods have been overwritten just for debugging. The handle_inv method is an example of the method dispatch - any message command type can have an associated handle_* method that will be called whenever a message of that type is received.

2.3 Sending a Transaction

Creating a transaction and sending it on the network is pretty straightforward. All you need to know is the private key that will be "sending" the money, the receipient's address, and the output transaction to use as the input for this transaction. Here's an example that sends 2M Satoshis after connecting to the P2P network:

```
from coinbits.client import BitcoinClient
from coinbits.txns.keys import PrivateKey
from coinbits.txns.wallet import Teller
from coinbits.protocol.serializers import OutPoint


class MyClient(BitcoinClient):
    def connected(self):
        # build a teller that will spend from the given private key
        key = PrivateKey('e1385343f7ea362b0de7e5772a6c766d44ce4bf69e1380381630bf1892c638d5')
        teller = Teller(key)

        # specify the origin transaction hash and output index to use for this transaction's input
        hexouthash = '8ed9e37a3c585ad2b28ebc9a7a76ff0bf250bd4a1d19cb42f8d29d62da8d3e67'
        outpoint = OutPoint()
        outpoint.out_hash = int(hexouthash, 16)
        outpoint.index = 0

        # pay 2M Satoshis to 1wYiNC2EERnKPWP7QbvWGEfNprtHg1bsz
        tx = teller.make_standard_tx(outpoint, '1wYiNC2EERnKPWP7QbvWGEfNprtHg1bsz', 2000000)

        print "New transaction's hash:", tx.calculate_hash()
        self.send_message(tx)

    def handle_inv(self, message_header, message):
        print "Got some inventory:", message
        for txn in message.inventory:
            print txn

MyClient("bitcoin.sipa.be").loop()
```

2.4 Running Tests

To run tests:

```
$ trial coinbits
```

2.5 coinbits

2.5.1 coinbits package

Subpackages

coinbits.protocol package

Submodules

coinbits.protocol.buffer module

class coinbits.protocol.buffer.**ProtocolBuffer**

Bases: object

receive_message()

This method will attempt to extract a header and message. It will return a tuple of (header, message) and set whichever can be set so far (None otherwise).

write(*data*)

coinbits.protocol.exceptions module

exception coinbits.protocol.exceptions.**NodeDisconnectException**

Bases: exceptions.Exception

This exception is thrown when a client is disconnected.

exception coinbits.protocol.exceptions.**UnknownMessageException**

Bases: exceptions.Exception

This exception is thrown when trying to (de)serialize an unknown message type

coinbits.protocol.fields module

class coinbits.protocol.fields.**BlockLocator**

Bases: coinbits.protocol.fields.Field

A block locator type used for getblocks and getheaders

datatype = '<I'

parse(*values*)

serialize()

class coinbits.protocol.fields.**Field**

Bases: object

Base class for the Fields. This class only implements the counter to keep the order of the fields on the serializer classes.

counter = 74

deserialize(*stream*)

This method must read the stream data and then deserialize and return the deserialized content.

Returns the deserialized content

Parameters **stream** – stream of data to read

parse(*value*)

This method should be implemented to parse the value parameter into the field internal representation.

Parameters **value** – value to be parsed

serialize()

Serialize the internal representation and return the serialized data.

Returns the serialized data

class coinbits.protocol.fields.**FixedStringField**(*length*)

Bases: coinbits.protocol.fields.Field

A fixed length string field.

Example of use:

```
class MessageHeaderSerializer(Serializer):
    model_class = MessageHeader
    magic = fields.UInt32LEField()
    command = fields.FixedStringField(12)
    length = fields.UInt32LEField()
    checksum = fields.UInt32LEField()
```

deserialize(*stream*)

parse(*value*)

serialize()

class coinbits.protocol.fields.**Hash**
Bases: coinbits.protocol.fields.Field

A hash type field.

datatype = '<I'

deserialize(*stream*)

parse(*value*)

serialize()

coinbits.protocol.fields.**INVENTORY_TYPE = {'MSG_BLOCK': 2, 'MSG_TX': 1, 'ERROR': 0}**
The type of the inventories

class coinbits.protocol.fields.**IPv4AddressField**
Bases: coinbits.protocol.fields.Field

An IPv4 address field without timestamp and reserved IPv6 space.

deserialize(*stream*)

parse(*value*)

reserved = '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xff\xff'

serialize()

class coinbits.protocol.fields.**Int16LEField**
Bases: coinbits.protocol.fields.PrimaryField

16-bit little-endian integer field.

datatype = '<h'

class coinbits.protocol.fields.**Int32LEField**
Bases: coinbits.protocol.fields.PrimaryField

32-bit little-endian integer field.

datatype = '<i'

class coinbits.protocol.fields.**Int64LEField**
Bases: coinbits.protocol.fields.PrimaryField

64-bit little-endian integer field.

datatype = '<q'

class coinbits.protocol.fields.**ListField**(*serializer_class*)
Bases: coinbits.protocol.fields.Field

A field used to serialize/deserialize a list of serializers.

Example of use:

```
class TxSerializer(Serializer):
    model_class = Tx
    version = fields.UInt32LEField()
    tx_in = fields.ListField(TxInSerializer)
    tx_out = fields.ListField(TxOutSerializer)
    lock_time = fields.UInt32LEField()
```

deserialize(*stream*)

parse(*value*)

serialize()

coinbits.protocol.fields.**MAGIC_VALUES = {'bitcoin_testnet': 3669344250, 'litecoin': 3686187259, 'bitcoin_testnet3'**
The network magic values

class coinbits.protocol.fields.**NestedField**(*serializer_class*)
Bases: coinbits.protocol.fields.Field

A field used to nest another serializer.

Example of use:

```
class TxInSerializer(Serializer):
    model_class = TxIn
    previous_output = fields.NestedField(OutPointSerializer)
    signature_script = fields.VariableStringField()
    sequence = fields.UInt32LEField()
```

deserialize(*stream*)

parse(*value*)

serialize()

coinbits.protocol.fields.**PROTOCOL_VERSION = 60002**
The protocol version

class coinbits.protocol.fields.**PrimaryField**
Bases: coinbits.protocol.fields.Field

This is a base class for all fields that has only one value and their value can be represented by a Python struct datatype.

Example of use:

```
class UInt32LEField(PrimaryField):
    datatype = "<I"
```

deserialize(*stream*)
Deserialize the stream using the struct data type specified.

Parameters **stream** – the data stream

parse(*value*)
This method will set the internal value to the specified value.

Parameters **value** – the value to be set

serialize()
Serialize the internal data and then return the serialized data.

coinbits.protocol.fields.**SERVICES = {'NODE_NETWORK': 1}**
The available services

class coinbits.protocol.fields.**UInt16BEField**
Bases: coinbits.protocol.fields.PrimaryField

16-bit big-endian unsigned integer field.

datatype = '>H'

class coinbits.protocol.fields.**UInt16LEField**
Bases: coinbits.protocol.fields.PrimaryField

16-bit little-endian unsigned integer field.

datatype = '<H'

class coinbits.protocol.fields.**UInt32LEField**
Bases: coinbits.protocol.fields.PrimaryField

32-bit little-endian unsigned integer field.

datatype = '<I'

class coinbits.protocol.fields.**UInt64LEField**
Bases: coinbits.protocol.fields.PrimaryField

64-bit little-endian unsigned integer field.

datatype = '<Q'

class coinbits.protocol.fields.**VariableIntegerField**
Bases: coinbits.protocol.fields.Field

A variable size integer field.

deserialize(*stream*)

parse(*value*)

serialize()

class coinbits.protocol.fields.**VariableStringField**
Bases: coinbits.protocol.fields.Field

A variable length string field.

deserialize(*stream*)

parse(*value*)

serialize()

coinbits.protocol.serializers module

class coinbits.protocol.serializers.**AddressVector**
Bases: coinbits.protocol.serializers.SerializableMessage

A vector of addresses.

command = 'addr'

class coinbits.protocol.serializers.**AddressVectorSerializer**
Bases: coinbits.protocol.serializers.Serializer

Serializer for the addresses vector.

model_class
alias of AddressVector

class coinbits.protocol.serializers.**Block**
Bases: coinbits.protocol.serializers.BlockHeader

The block message. This message contains all the transactions present in the block.

command = 'block'

class coinbits.protocol.serializers.**BlockHeader**
Bases: coinbits.protocol.serializers.SerializableMessage

The header of the block.

calculate_hash()
This method will calculate the hash of the block.

class coinbits.protocol.serializers.**BlockHeaderSerializer**
Bases: coinbits.protocol.serializers.Serializer

The serializer for the block header.

model_class
alias of BlockHeader

class coinbits.protocol.serializers.**BlockSerializer**
Bases: coinbits.protocol.serializers.Serializer

The deserializer for the blocks.

model_class
alias of Block

class coinbits.protocol.serializers.**GetAddr**
Bases: coinbits.protocol.serializers.SerializableMessage

The getaddr command.

command = 'getaddr'

class coinbits.protocol.serializers.**GetAddrSerializer**
Bases: coinbits.protocol.serializers.Serializer

The serializer for the getaddr command.

model_class
alias of GetAddr

class coinbits.protocol.serializers.**GetBlocks**(*hashes*)
Bases: coinbits.protocol.serializers.SerializableMessage

The getblocks command.

command = 'getblocks'

class coinbits.protocol.serializers.**GetBlocksSerializer**
Bases: coinbits.protocol.serializers.Serializer

model_class
alias of GetBlocks

class coinbits.protocol.serializers.**GetData**
Bases: coinbits.protocol.serializers.InventoryVector

GetData message command.

command = 'getdata'

class coinbits.protocol.serializers.**GetDataSerializer**
Bases: coinbits.protocol.serializers.Serializer

Serializer for the GetData command.

model_class
alias of GetData

class coinbits.protocol.serializers.**GetHeaders**(*hashes*)
Bases: coinbits.protocol.serializers.GetBlocks

command = 'getheaders'

class coinbits.protocol.serializers.**GetHeadersSerializer**
Bases: coinbits.protocol.serializers.GetBlocksSerializer

model_class
alias of GetHeaders

class coinbits.protocol.serializers.**HeaderVector**
Bases: coinbits.protocol.serializers.SerializableMessage

The header only vector.

command = 'headers'

class coinbits.protocol.serializers.**HeaderVectorSerializer**
Bases: coinbits.protocol.serializers.Serializer

Serializer for the block header vector.

model_class
alias of HeaderVector

class coinbits.protocol.serializers.**IPv4Address**
Bases: object

The IPv4 Address (without timestamp).

class coinbits.protocol.serializers.**IPv4AddressSerializer**
Bases: coinbits.protocol.serializers.Serializer

Serializer for the IPv4Address.

model_class
alias of IPv4Address

class coinbits.protocol.serializers.**IPv4AddressTimestamp**
Bases: coinbits.protocol.serializers.IPv4Address

The IPv4 Address with timestamp.

class coinbits.protocol.serializers.**IPv4AddressTimestampSerializer**
Bases: coinbits.protocol.serializers.Serializer

Serializer for the IPv4AddressTimestamp.

model_class
alias of IPv4AddressTimestamp

class coinbits.protocol.serializers.**Inventory**
Bases: coinbits.protocol.serializers.SerializableMessage

The Inventory representation.

type_to_text()
Converts the inventory type to text representation.

class coinbits.protocol.serializers.**InventorySerializer**
Bases: coinbits.protocol.serializers.Serializer

The serializer for the Inventory.

model_class
alias of Inventory

class coinbits.protocol.serializers.**InventoryVector**
Bases: coinbits.protocol.serializers.SerializableMessage

A vector of inventories.

command = 'inv'

class coinbits.protocol.serializers.**InventoryVectorSerializer**
Bases: coinbits.protocol.serializers.Serializer

The serializer for the vector of inventories.

model_class
alias of InventoryVector

class coinbits.protocol.serializers.**MemPool**
Bases: coinbits.protocol.serializers.SerializableMessage

The mempool command.

command = 'mempool'

class coinbits.protocol.serializers.**MemPoolSerializer**
Bases: coinbits.protocol.serializers.Serializer

The serializer for the mempool command.

model_class
alias of MemPool

class coinbits.protocol.serializers.**MessageHeader**(*coin='bitcoin'*)
Bases: object

The header of all bitcoin messages.

class coinbits.protocol.serializers.**MessageHeaderSerializer**
Bases: coinbits.protocol.serializers.Serializer

Serializer for the MessageHeader.

static calc_checksum(*payload*)
Calculate the checksum of the specified payload.

Parameters **payload** – The binary data payload.

static calcsize()

model_class
alias of MessageHeader

class coinbits.protocol.serializers.**NotFound**
Bases: coinbits.protocol.serializers.GetData

NotFound command message.

command = 'notfound'

class coinbits.protocol.serializers.**NotFoundSerializer**
Bases: coinbits.protocol.serializers.Serializer

Serializer for the NotFound message.

model_class
alias of NotFound

class coinbits.protocol.serializers.**OutPoint**
Bases: object

The OutPoint representation.

class coinbits.protocol.serializers.**OutPointSerializer**
Bases: coinbits.protocol.serializers.Serializer

The OutPoint representation serializer.

model_class
alias of OutPoint

class coinbits.protocol.serializers.**Ping**
Bases: coinbits.protocol.serializers.SerializableMessage

The ping command, which should always be answered with a Pong.

command = 'ping'

class coinbits.protocol.serializers.**PingSerializer**
Bases: coinbits.protocol.serializers.Serializer

The ping command serializer.

model_class
alias of Ping

class coinbits.protocol.serializers.**Pong**
Bases: coinbits.protocol.serializers.SerializableMessage

The pong command, usually returned when a ping command arrives.

command = 'pong'

class coinbits.protocol.serializers.**PongSerializer**
Bases: coinbits.protocol.serializers.Serializer

The pong command serializer.

model_class
alias of Pong

class coinbits.protocol.serializers.**Reject**
Bases: coinbits.protocol.serializers.SerializableMessage

command = 'reject'

class coinbits.protocol.serializers.**RejectSerializer**
Bases: coinbits.protocol.serializers.Serializer

model_class
alias of Reject

class coinbits.protocol.serializers.**SerializableMessage**
Bases: object

get_field_names()

get_message(*coin='bitcoin'*)
Get the binary version of this message, complete with header.

class coinbits.protocol.serializers.**Serializer**
Bases: coinbits.protocol.serializers.SerializerABC

The main serializer class, inherit from this class to create custom serializers.

Example of use:

```
class VerAckSerializer(Serializer):
    model_class = VerAck
```

deserialize(*stream*)
This method will read the stream and then will deserialize the binary data information present on it.

Parameters **stream** – A file-like object (StringIO, file, socket, etc.)

serialize(*obj*, *fields=None*)
This method will receive an object and then will serialize it according to the fields declared on the serializer.

Parameters **obj** – The object to serializer.

class coinbits.protocol.serializers.**SerializerABC**
Bases: object

The serializer abstract base class.

class coinbits.protocol.serializers.**SerializerMeta**
Bases: type

The serializer meta class. This class will create an attribute called '_fields' in each serializer with the ordered dict of fields present on the subclasses.

classmethod **get_fields**(*meta*, *bases*, *attrs*, *field_class*)
This method will construct an ordered dict with all the fields present on the serializer classes.

class coinbits.protocol.serializers.**Tx**
Bases: coinbits.protocol.serializers.SerializableMessage

The main transaction representation, this object will contain all the inputs and outputs of the transaction.

calculate_hash()
This method will calculate the hash of the transaction.

command = 'tx'

class coinbits.protocol.serializers.**TxIn**
Bases: object

The transaction input representation.

class coinbits.protocol.serializers.**TxInSerializer**
Bases: coinbits.protocol.serializers.Serializer

The transaction input serializer.

model_class
alias of TxIn

class coinbits.protocol.serializers.**TxOut**
Bases: object

The transaction output.

get_btc_value()

class coinbits.protocol.serializers.**TxOutSerializer**
Bases: coinbits.protocol.serializers.Serializer

The transaction output serializer.

model_class
alias of TxOut

class coinbits.protocol.serializers.**TxSerializer**
Bases: coinbits.protocol.serializers.Serializer

The transaction serializer.

model_class
alias of Tx

class coinbits.protocol.serializers.**VerAck**
Bases: coinbits.protocol.serializers.SerializableMessage

The version acknowledge (verack) command.

command = 'verack'

class coinbits.protocol.serializers.**VerAckSerializer**
Bases: coinbits.protocol.serializers.Serializer

The serializer for the verack command.

model_class
alias of VerAck

class coinbits.protocol.serializers.**Version**
Bases: coinbits.protocol.serializers.SerializableMessage

The version command.

command = 'version'

class coinbits.protocol.serializers.**VersionSerializer**
Bases: coinbits.protocol.serializers.Serializer

The version command serializer.

model_class
alias of Version

coinbits.protocol.serializers.**getSerializer**(*msgtype*)
Return a new serializer of the given msg type.

coinbits.protocol.utils module

coinbits.protocol.utils.**nonce**()
Return a random int between 0 and (2^32)-1

Module contents

coinbits.txns package

Submodules

coinbits.txns.exceptions module

exception coinbits.txns.exceptions.**KeyDecodeError**

Bases: exceptions.Exception

This exception is thrown when trying to decode a key from one format to another.

coinbits.txns.keys module

class coinbits.txns.keys.**PrivateKey**(*hexkey=None*)

Bases: object

This is a representation for Bitcoin private keys. In this class you'll find methods to import/export keys from multiple formats. Use a hex string representation to construct a new PublicKey or use the clas methods to import from another format.

Construct a new PrivateKey object, based optionally on an existing hex representation.

Parameters

- **hexkey** – The key in hex string format. If one isn't
- **a new private key will be generated.** (*provided,*) –

__str__()

This method will convert the PrivateKey to a string representation.

classmethod from_string(*klass*, *stringkey*)

This method will create a new Private Key using the specified string data.

Parameters **stringkey** – The key in string format

Returns A new PrivateKey

classmethod from_wif(*klass*, *wifkey*)

This method will create a new PrivateKey from a WIF format string.

Parameters **wifkey** – The private key in WIF format

Returns A new PrivateKey

get_public_key()

This method will create a new PublicKey based on this PrivateKey.

Returns A new PublicKey

sign(*data*)

Digest and then sign the data.

to_address()

Convert to public key and then get the public address for that key.

to_hex()

This method will convert the Private Key to a hex string representation.

Returns Hex string representation of this PrivateKey

to_wif()

This method will export the Private Key to WIF (Wallet Import Format).

Returns The PrivateKey in WIF format.

wif_prefix = '\x80'

class coinbits.txns.keys.**PublicKey**(*hexkey*)

Bases: object

This is a representation for Bitcoin public keys. In this class you'll find methods to import/export keys from multiple formats. Use a hex string representation to construct a new public key or use the clas methods to import from another format.

Initialize a public key object. Requires an existing version of this key in hex.

Parameters **hexkey** – The key in hex string format

__str__()
This method will convert the public key to a string representation.

Returns A string representation of the public key

classmethod from_private_key(*klass*, *private_key*)
This class method will create a new PublicKey based on a PrivateKey.

Parameters **private_key** – The PrivateKey

Returns A new PublicKey

key_prefix = '\x04'

to_address()
This method will convert the public key to a bitcoin address.

Returns A bitcoin address for the public key

to_hex()
This method will convert the public key to a hex string representation.

Returns A hex string representation of the public key

verify(*signature*, *message*)
Verify the given signature of the message. Returns True if verification is successful, False otherwise.

coinbits.txns.scripts module

coinbits.txns.scripts.**pay_to_pubkey_hash**(*key*)
76 A9 14 OP_DUP OP_HASH160 Bytes to push

89 AB CD EF AB BA AB BA AB BA AB BA AB BA AB BA AB BA 88 AC Data to push OP_EQUALVERIFY OP_CHECKSIG

coinbits.txns.wallet module

class coinbits.txns.wallet.**Teller**(*private_key*)
Bases: object

A Teller can be used to create transactions.

Args: private_key: a PrivateKey

make_standard_tx(*output*, *destination*, *amount*, *fee=10000*)
Create a standard transaction.

Parameters

- **output** – The previous output transaction reference, as an OutPoint structure
- **destination** – The address to transfer to
- **amount** – The amount to transfer (in Satoshis)
- **fee** – The amount to reserve for the miners. Default is 10K Satoshi's.

Returns A Tx object suitable for serialization / transfer on the wire.

Module contents

Submodules

coinbits.client module

class coinbits.client.**BitcoinClient**(*peerip*, *port=8333*)
Bases: object

The base class for a Bitcoin network client. This class will handle the initial handshake and responding to pings.

coin = 'bitcoin'

connected()
Called once we've exchanged version information and can make calls on the network.

handle_ping(*message_header*, *message*)
This method will handle the Ping message and then will answer every Ping message with a Pong message using the nonce received.

Parameters

- **message_header** – The header of the Ping message
- **message** – The Ping message

handle_version(*message_header*, *message*)
This method will handle the Version message and will send a VerAck message when it receives the Version message.

Parameters

- **message_header** – The Version message header
- **message** – The Version message

loop()
This is the main method of the client, it will enter in a receive/send loop.

message_received(*message_header*, *message*)
This method will be called for every message, and then will delegate to the appropriate handle_* function for the given message (if it exists).

Parameters

- **message_header** – The message header
- **message** – The message object

send_message(*message*)
This method will serialize the message using the appropriate serializer based on the message command and then it will send it to the socket stream.

Parameters **message** – The message object to send

coinbits.encoding module

coinbits.encoding.**b256encode**(*n*)

coinbits.encoding.**b58_check_decode**(*s*)

coinbits.encoding.**b58decode**(*s*)

coinbits.encoding.**b58encode**(*n*)

Module contents

Coinbits is a Python library for bitcoin peer to peer communication.

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

C

Symbols

A

B

C

D

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W