
remcall Documentation

Release 0.1.0

luphord

Feb 20, 2019

Contents:

1	remcall	1
2	Installation	3
2.1	Stable release	3
2.2	From sources	3
3	Usage	5
4	remcall	7
4.1	remcall package	7
5	Contributing	17
5.1	Types of Contributions	17
5.2	Get Started!	18
5.3	Pull Request Guidelines	19
5.4	Tips	19
5.5	Deploying	19
6	Credits	21
6.1	Development Lead	21
6.2	Contributors	21
7	History	23
7.1	0.1.0 (not yet)	23
8	Indices and tables	25
	Python Module Index	27

remcall (short for remote method calls) is a protocol for inter process communication (IPC) between different programming languages using object proxying as its primary method for information exchange. Communication using remcall requires the upfront definition of a schema (comprised of record and enum types and more importantly interfaces with method signatures) which then depending on the programming language is compiled or interpreted. Both communication participants are then free to implement any or none of the interfaces and reference concrete objects to the other side which will be represented using proxy objects. There is a certain distinction between a server (waiting for connections, serving and entry point) and a client (initiating a connection, performing the first method call) in remcall, but the protocol allows for method calls and object proxying in both directions. Remcall employs a binary representation for both, its schema and its communication protocol. Communication can be layered on top of any bidirectional streams supporting binary data such as TCP sockets, stdin/out, websockets.

- Free software: MIT license
- Documentation: <https://remcall.readthedocs.io>.

2.1 Stable release

To install remcall, run this command in your terminal:

```
$ pip install remcall
```

This is the preferred method to install remcall, as it will always install the most recent stable release.

If you don't have [pip](#) installed, this [Python installation guide](#) can guide you through the process.

2.2 From sources

The sources for remcall can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/luphord/remcall
```

Or download the [tarball](#):

```
$ curl -OL https://github.com/luphord/remcall/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```


CHAPTER 3

Usage

To use remcall in a project:

```
import remcall
```


4.1 remcall package

4.1.1 Subpackages

remcall.codec package

Submodules

remcall.codec.base module

remcall.codec.read module

```
class remcall.codec.read.ReaderBase(stream)
    Bases: object

    read_constant (bytes_const: bytes)
    read_float32 ()
    read_float64 ()
    read_from_stream (bytes_count: int)
    read_int16 ()
    read_int32 ()
    read_int64 ()
    read_int8 ()
    read_name ()
    read_signed_integer (nbytes: int)
```

```
    read_string()
    read_struct_format(fmt)
    read_type_ref()
    read_uint16()
    read_uint32()
    read_uint64()
    read_uint8()
    read_unsigned_integer(nbytes: int)
class remcall.codec.read.SchemaReader(stream)
    Bases: remcall.codec.read.ReaderBase
    read_enum()
    read_interface()
    read_method()
    read_record()
    read_schema()
remcall.codec.read.read_schema(stream)
remcall.codec.read.schema_from_bytes(byt: bytes)
```

remcall.codec.write module

```
class remcall.codec.write.SchemaWriter(schema, stream)
    Bases: remcall.codec.write.WriterBase
    write_enum(enum)
    write_interface(interface)
    write_method(method)
    write_name(s)
    write_record(record)
    write_schema()
    write_to_stream(data: bytes)
class remcall.codec.write.WriterBase(schema, ostream)
    Bases: object
    write_bytes(b)
    write_float32(f: float)
    write_float64(f: float)
    write_int16(i: int)
    write_int32(i: int)
    write_int64(i: int)
    write_int8(i: int)
```

```

write_method_ref (method_idx)
write_signed_integer (i: int, nbytes: int)
write_string (s)
write_to_stream (data: bytes)
write_type_ref (typ: remcall.schema.core.Type)
write_uint16 (i: int)
write_uint32 (i: int)
write_uint64 (i: int)
write_uint8 (i: int)
write_unsigned_integer (i: int, nbytes: int)
remcall.codec.write.schema_to_bytes (schema)
remcall.codec.write.write_schema (schema, stream)

```

Module contents

remcall.communication package

Submodules

remcall.communication.base module

remcall.communication.bridge module

```

class remcall.communication.bridge.Bridge (schema, instream, outstream, main,
                                           enum_record_implementation: rem-
                                           call.implementation.EnumRecordImplementation)
    Bases: object
    acknowledge_disconnect ()
    call_method (method, this, args_dict)
    disconnect ()
    return_method (request_id: int, return_type: remcall.schema.core.Type, return_value)

```

remcall.communication.proxy module

```

class remcall.communication.proxy.MethodProxy (interface, method, bridge,
                                                  name_converter)
    Bases: object
class remcall.communication.proxy.ProxyFactory (schema, bridge, name_converter)
    Bases: object
class remcall.communication.proxy.ProxyType
    Bases: object
remcall.communication.proxy.create_proxy_class (interface, bridge, name_converter)

```

```
remcall.communication.proxy.create_proxy_classes(schema, bridge)
remcall.communication.proxy.create_proxy_classes_dict(schema, bridge,
                                                         name_converter)
```

remcall.communication.receive module

```
class remcall.communication.receive.Receiver(schema, instream, get_object, re-
                                             turn_method_result, acknowl-
                                             edge_disconnect, name_converter)

    Bases: remcall.codec.read.ReaderBase

    mainloop()
    process_method_call()
    process_method_return()
    process_next()
    read_enum_value(typ: remcall.schema.core.Type)
    read_from_stream(bytes_count: int)
    read_method_ref()
    read_object(typ: remcall.schema.core.Type)
    read_object_ref(typ: remcall.schema.core.Type)
    read_request_id()
    read_value(typ: remcall.schema.core.Type)
    receive_and_check_schema()
    wait_for_method_return(request_id, return_type)
```

remcall.communication.send module

```
class remcall.communication.send.Sender(schema, outstream, get_id_for_object)
    Bases: remcall.codec.write.WriterBase

    acknowledge_disconnect()
    call_method(method, this, args_dict)
    disconnect()
    noop()
    request_schema()
    return_method(request_id, return_type, return_value)
    send_schema()
    write_enum_value(enum_value)
    write_object_ref(obj)
    write_record_value(val)
    write_request_id(request_id=None)
```

```
write_to_stream(data: bytes)
```

```
write_value(typ, value)
```

remcall.communication.store module

```
class remcall.communication.store.IdStore
```

```
    Bases: object
```

```
    contains_object(obj)
```

```
    delete_object(obj)
```

```
    get_id_for_object(obj)
```

```
class remcall.communication.store.ReferenceStore(is_client, proxy_factory)
```

```
    Bases: object
```

```
    get_id_for_implementation_object(obj)
```

```
    get_id_for_object(obj)
```

```
    get_id_for_proxy_object(obj)
```

```
    get_implementation_object(key: int)
```

```
    get_object(key: int, typ: remcall.schema.core.Type)
```

```
    get_proxy_object(key: int, typ: remcall.schema.core.Type)
```

```
    next_object_id()
```

```
    object_id_sign
```

Module contents

remcall.schema package

Submodules

remcall.schema.base module

```
remcall.schema.base.assert_name(name: str)
```

remcall.schema.core module

```
class remcall.schema.core.Type(name: str)
```

```
    Bases: object
```

```
    is_declared
```

```
    resolve_type_references(type_ref_lookup)
```

```
    sort_key
```

```
    type_order = -1
```

```
class remcall.schema.core.Array(typ: remcall.schema.core.Type)
```

```
    Bases: remcall.schema.core.Type
```

```
    type_order = 3

class remcall.schema.core.Primitive(name: str)
    Bases: remcall.schema.core.Type

class remcall.schema.core.Enum(name: str, values: Iterable[str])
    Bases: remcall.schema.core.Type

    is_declared

    pretty_print() → str

    type_order = 0

class remcall.schema.core.Record(name: str, fields: Iterable[Tuple[Union[remcall.schema.core.Type, remcall.schema.typeref.TypeRef], str]])
    Bases: remcall.schema.core.Type

    is_declared

    pretty_print() → str

    resolve_type_references(type_ref_lookup: Mapping[remcall.schema.typeref.TypeRef, remcall.schema.core.Type]) → None

    type_order = 1

class remcall.schema.core.Method(name: str, arguments: Iterable[Tuple[Union[remcall.schema.core.Type, remcall.schema.typeref.TypeRef], str]], return_type: remcall.schema.core.Type)
    Bases: object

class remcall.schema.core.Interface(name: str, methods: Iterable[remcall.schema.core.Method])
    Bases: remcall.schema.core.Type

    is_declared

    methods_sorted

    pretty_print() → str

    resolve_type_references(type_ref_lookup: Mapping[remcall.schema.typeref.TypeRef, remcall.schema.core.Type]) → None

    type_order = 2

class remcall.schema.core.Schema(label, types, bytes_method_ref=2, bytes_object_ref=4, sha256_digest=None)
    Bases: object

    declared_types

    enums

    enums_sorted

    interfaces

    interfaces_sorted

    iter_declared_types

    method_lookup

    method_table
```



```
method_to_interface
pretty_print()
records
records_sorted
type_schemas
type_table
```

remcall.schema.typeref module

```
class remcall.schema.typeref.TypeRef (type_ref: int)
    Bases: object

    Represents temporary type references by an integer; to be resolved to actual types later
```

Module contents

4.1.2 Submodules

4.1.3 remcall.error module

```
exception remcall.error.DuplicateMethodReturnValue (request_id)
    Bases: remcall.error.RemcallError

exception remcall.error.DuplicateRegistrationForMethodReturn (request_id)
    Bases: remcall.error.RemcallError

exception remcall.error.MethodNotAvailable (method, impl_method_name, this)
    Bases: remcall.error.RemcallError

exception remcall.error.MissingMethodReturnValueEvent (request_id)
    Bases: remcall.error.RemcallError

exception remcall.error.RemcallError
    Bases: Exception

exception remcall.error.UnknownCommand (command)
    Bases: remcall.error.RemcallError

exception remcall.error.UnknownImplementationObjectReference (key)
    Bases: remcall.error.RemcallError

exception remcall.error.UnknownProxyObject (obj)
    Bases: remcall.error.RemcallError

exception remcall.error.UnknownType (typ)
    Bases: remcall.error.RemcallError

exception remcall.error.WrongNumberOfBytesRead (bytes_requested, bytes_read, offset)
    Bases: remcall.error.RemcallError
```

4.1.4 remcall.generate module

```
class remcall.generate.CSharpCodeGenerator(schema, namespace='Remcall.Generated',  
                                           name_converter=<remcall.naming.CSharpNameConverter  
                                           object>)  
  
    Bases: object  
  
    dedent()  
    indent()  
    indent_chars = '\t'  
    linebreak()  
    scalartypename(typ)  
    type_names = {Primitive("void"): 'void', Primitive("boolean"): 'bool', Primitive("in  
    typename(typ)  
    write_enum(enum)  
    write_interface(interface)  
    write_method(method)  
    write_record(record)  
    write_schema(fp)  
    writeline(s)
```

4.1.5 remcall.implementation module

```
class remcall.implementation.EnumRecordImplementation(schema, name_converter)  
    Bases: object  
  
class remcall.implementation.RecordType  
    Bases: object  
  
remcall.implementation.create_enum_implementation(enum, name_converter)  
remcall.implementation.create_record_implementation(record, name_converter)
```

4.1.6 remcall.naming module

```
class remcall.naming.CSharpNameConverter  
    Bases: remcall.naming.IdentityNameConverter  
  
    interface_name(name)  
    parameter_name(name)  
  
class remcall.naming.IdentityNameConverter  
    Bases: object  
  
    enum_field_name(name)  
    enum_name(name)  
    interface_name(name)  
    method_name(name)
```

```

    parameter_name (name)
    record_field_name (name)
    record_name (name)
    type_name (typ: remcall.schema.core.Type)
class remcall.naming.PythonNameConverter
    Bases: remcall.naming.IdentityNameConverter
    enum_field_name (name)
    method_name (name)
    parameter_name (name)
    record_field_name (name)
    type_name (typ: remcall.schema.core.Type)

```

4.1.7 remcall.util module

```

class remcall.util.QueueStream (name=None)
    Bases: object
    flush ()
    read (size: int)
    stream_counter = 0
    write (data: bytes)
class remcall.util.TypeWrapper (typ, name_converter)
    Bases: object
    Wraps a core.Type and provides a nice annotation for Signature instances
remcall.util.view_hex (b: bytes)

```

4.1.8 Module contents

Remcall (short for remote method calls) is a protocol for inter process communication (IPC) between different programming languages using object proxying as its primary method for information exchange. Communication using remcall requires the upfront definition of a schema (comprised of record and enum types and more importantly interfaces with method signatures) which then depending on the programming language is compiled or interpreted. Both communication participants are then free to implement any or none of the interfaces and reference concrete objects to the other side which will be represented using proxy objects. There is a certain distinction between a server (waiting for connections, serving and entry point) and a client (initiating a connection, performing the first method call) in remcall, but the protocol allows for method calls and object proxying in both directions. Remcall employs a binary representation for both, its schema and its communication protocol. Communication can be layered on top of any bidirectional streams supporting binary data such as TCP sockets, stdin/out, websockets.

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at <https://github.com/luphord/remcall/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

5.1.4 Write Documentation

remcall could always use more documentation, whether as part of the official remcall docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/luphord/remcall/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *remcall* for local development.

1. Fork the *remcall* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/remcall.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv remcall
$ cd remcall/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 remcall tests
$ python setup.py test or py.test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/luphord/remcall/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_remcall
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- luphord <luphord@protonmail.com>

6.2 Contributors

None yet. Why not be the first?

7.1 0.1.0 (not yet)

- First release on PyPI.

CHAPTER 8

Indices and tables

- `genindex`
- `modindex`
- `search`

r

- remcall, [15](#)
- remcall.codec, [9](#)
- remcall.codec.base, [7](#)
- remcall.codec.read, [7](#)
- remcall.codec.write, [8](#)
- remcall.communication, [11](#)
- remcall.communication.base, [9](#)
- remcall.communication.bridge, [9](#)
- remcall.communication.proxy, [9](#)
- remcall.communication.receive, [10](#)
- remcall.communication.send, [10](#)
- remcall.communication.store, [11](#)
- remcall.error, [13](#)
- remcall.generate, [14](#)
- remcall.implementation, [14](#)
- remcall.naming, [14](#)
- remcall.schema, [13](#)
- remcall.schema.base, [11](#)
- remcall.schema.core, [11](#)
- remcall.schema.typeref, [13](#)
- remcall.util, [15](#)

A

acknowledge_disconnect() (remcall.communication.bridge.Bridge method), 9

acknowledge_disconnect() (remcall.communication.send.Sender method), 10

Array (class in remcall.schema.core), 11

assert_name() (in module remcall.schema.base), 11

B

Bridge (class in remcall.communication.bridge), 9

C

call_method() (remcall.communication.bridge.Bridge method), 9

call_method() (remcall.communication.send.Sender method), 10

contains_object() (remcall.communication.store.IdStore method), 11

create_enum_implementation() (in module remcall.implementation), 14

create_proxy_class() (in module remcall.communication.proxy), 9

create_proxy_classes() (in module remcall.communication.proxy), 9

create_proxy_classes_dict() (in module remcall.communication.proxy), 10

create_record_implementation() (in module remcall.implementation), 14

CSharpCodeGenerator (class in remcall.generate), 14

CSharpNameConverter (class in remcall.naming), 14

D

declared_types (remcall.schema.core.Schema attribute), 12

dedent() (remcall.generate.CSharpCodeGenerator method), 14

delete_object() (remcall.communication.store.IdStore method), 11

disconnect() (remcall.communication.bridge.Bridge method), 9

disconnect() (remcall.communication.send.Sender method), 10

DuplicateMethodReturnValue, 13

DuplicateRegistrationForMethodReturn, 13

E

Enum (class in remcall.schema.core), 12

enum_field_name() (remcall.naming.IdentityNameConverter method), 14

enum_field_name() (remcall.naming.PythonNameConverter method), 15

enum_name() (remcall.naming.IdentityNameConverter method), 14

EnumRecordImplementation (class in remcall.implementation), 14

enums (remcall.schema.core.Schema attribute), 12

enums_sorted (remcall.schema.core.Schema attribute), 12

F

flush() (remcall.util.QueueStream method), 15

G

get_id_for_implementation_object() (remcall.communication.store.ReferenceStore method), 11

get_id_for_object() (remcall.communication.store.IdStore method), 11

get_id_for_object() (remcall.communication.store.ReferenceStore method), 11

`get_id_for_proxy_object()` (remcall.communication.store.ReferenceStore method), 11

`get_implementation_object()` (remcall.communication.store.ReferenceStore method), 11

`get_object()` (remcall.communication.store.ReferenceStore method), 11

`get_proxy_object()` (remcall.communication.store.ReferenceStore method), 11

I

`IdentityNameConverter` (class in remcall.naming), 14

`IdStore` (class in remcall.communication.store), 11

`indent()` (remcall.generate.CSharpCodeGenerator method), 14

`indent_chars` (remcall.generate.CSharpCodeGenerator attribute), 14

`Interface` (class in remcall.schema.core), 12

`interface_name()` (remcall.naming.CSharpNameConverter method), 14

`interface_name()` (remcall.naming.IdentityNameConverter method), 14

`interfaces` (remcall.schema.core.Schema attribute), 12

`interfaces_sorted` (remcall.schema.core.Schema attribute), 12

`is_declared` (remcall.schema.core.Enum attribute), 12

`is_declared` (remcall.schema.core.Interface attribute), 12

`is_declared` (remcall.schema.core.Record attribute), 12

`is_declared` (remcall.schema.core.Type attribute), 11

`iter_declared_types` (remcall.schema.core.Schema attribute), 12

L

`linebreak()` (remcall.generate.CSharpCodeGenerator method), 14

M

`mainloop()` (remcall.communication.receive.Receiver method), 10

`Method` (class in remcall.schema.core), 12

`method_lookup` (remcall.schema.core.Schema attribute), 12

`method_name()` (remcall.naming.IdentityNameConverter method), 14

`method_name()` (remcall.naming.PythonNameConverter method), 15

`method_table` (remcall.schema.core.Schema attribute), 12

`method_to_interface` (remcall.schema.core.Schema attribute), 12

`MethodNotAvailable`, 13

`MethodProxy` (class in remcall.communication.proxy), 9

`methods_sorted` (remcall.schema.core.Interface attribute), 12

`MissingMethodReturnValueEvent`, 13

N

`next_object_id()` (remcall.communication.store.ReferenceStore method), 11

`noop()` (remcall.communication.send.Sender method), 10

O

`object_id_sign` (remcall.communication.store.ReferenceStore attribute), 11

P

`parameter_name()` (remcall.naming.CSharpNameConverter method), 14

`parameter_name()` (remcall.naming.IdentityNameConverter method), 14

`parameter_name()` (remcall.naming.PythonNameConverter method), 15

`pretty_print()` (remcall.schema.core.Enum method), 12

`pretty_print()` (remcall.schema.core.Interface method), 12

`pretty_print()` (remcall.schema.core.Record method), 12

`pretty_print()` (remcall.schema.core.Schema method), 13

`Primitive` (class in remcall.schema.core), 12

`process_method_call()` (remcall.communication.receive.Receiver method), 10

`process_method_return()` (remcall.communication.receive.Receiver method), 10

`process_next()` (remcall.communication.receive.Receiver method), 10

`ProxyFactory` (class in remcall.communication.proxy), 9

`ProxyType` (class in remcall.communication.proxy), 9

`PythonNameConverter` (class in remcall.naming), 15

Q

`QueueStream` (class in remcall.util), 15

R

`read()` (remcall.util.QueueStream method), 15

`read_constant()` (remcall.codec.read.ReaderBase method), 7

`read_enum()` (remcall.codec.read.SchemaReader method), 8

`read_enum_value()` (remcall.communication.receive.Receiver method), 10

`read_float32()` (remcall.codec.read.ReaderBase method), 7

[read_float64\(\)](#) (remcall.codec.read.ReaderBase method), 7
[read_from_stream\(\)](#) (remcall.codec.read.ReaderBase method), 7
[read_from_stream\(\)](#) (remcall.communication.receive.Receiver method), 10
[read_int16\(\)](#) (remcall.codec.read.ReaderBase method), 7
[read_int32\(\)](#) (remcall.codec.read.ReaderBase method), 7
[read_int64\(\)](#) (remcall.codec.read.ReaderBase method), 7
[read_int8\(\)](#) (remcall.codec.read.ReaderBase method), 7
[read_interface\(\)](#) (remcall.codec.read.SchemaReader method), 8
[read_method\(\)](#) (remcall.codec.read.SchemaReader method), 8
[read_method_ref\(\)](#) (remcall.communication.receive.Receiver method), 10
[read_name\(\)](#) (remcall.codec.read.ReaderBase method), 7
[read_object\(\)](#) (remcall.communication.receive.Receiver method), 10
[read_object_ref\(\)](#) (remcall.communication.receive.Receiver method), 10
[read_record\(\)](#) (remcall.codec.read.SchemaReader method), 8
[read_request_id\(\)](#) (remcall.communication.receive.Receiver method), 10
[read_schema\(\)](#) (in module remcall.codec.read), 8
[read_schema\(\)](#) (remcall.codec.read.SchemaReader method), 8
[read_signed_integer\(\)](#) (remcall.codec.read.ReaderBase method), 7
[read_string\(\)](#) (remcall.codec.read.ReaderBase method), 7
[read_struct_format\(\)](#) (remcall.codec.read.ReaderBase method), 8
[read_type_ref\(\)](#) (remcall.codec.read.ReaderBase method), 8
[read_uint16\(\)](#) (remcall.codec.read.ReaderBase method), 8
[read_uint32\(\)](#) (remcall.codec.read.ReaderBase method), 8
[read_uint64\(\)](#) (remcall.codec.read.ReaderBase method), 8
[read_uint8\(\)](#) (remcall.codec.read.ReaderBase method), 8
[read_unsigned_integer\(\)](#) (remcall.codec.read.ReaderBase method), 8
[read_value\(\)](#) (remcall.communication.receive.Receiver method), 10
[ReaderBase](#) (class in remcall.codec.read), 7
[receive_and_check_schema\(\)](#) (remcall.communication.receive.Receiver method), 10
[Receiver](#) (class in remcall.communication.receive), 10
[Record](#) (class in remcall.schema.core), 12
[record_field_name\(\)](#) (remcall.naming.IdentityNameConverter method), 15
[record_field_name\(\)](#) (remcall.naming.PythonNameConverter method), 15
[record_name\(\)](#) (remcall.naming.IdentityNameConverter method), 15
[records](#) (remcall.schema.core.Schema attribute), 13
[records_sorted](#) (remcall.schema.core.Schema attribute), 13
[RecordType](#) (class in remcall.implementation), 14
[ReferenceStore](#) (class in remcall.communication.store), 11
[remcall](#) (module), 15
[remcall.codec](#) (module), 9
[remcall.codec.base](#) (module), 7
[remcall.codec.read](#) (module), 7
[remcall.codec.write](#) (module), 8
[remcall.communication](#) (module), 11
[remcall.communication.base](#) (module), 9
[remcall.communication.bridge](#) (module), 9
[remcall.communication.proxy](#) (module), 9
[remcall.communication.receive](#) (module), 10
[remcall.communication.send](#) (module), 10
[remcall.communication.store](#) (module), 11
[remcall.error](#) (module), 13
[remcall.generate](#) (module), 14
[remcall.implementation](#) (module), 14
[remcall.naming](#) (module), 14
[remcall.schema](#) (module), 13
[remcall.schema.base](#) (module), 11
[remcall.schema.core](#) (module), 11
[remcall.schema.typeref](#) (module), 13
[remcall.util](#) (module), 15
[RemcallError](#), 13
[request_schema\(\)](#) (remcall.communication.send.Sender method), 10
[resolve_type_references\(\)](#) (remcall.schema.core.Interface method), 12
[resolve_type_references\(\)](#) (remcall.schema.core.Record method), 12
[resolve_type_references\(\)](#) (remcall.schema.core.Type method), 11
[return_method\(\)](#) (remcall.communication.bridge.Bridge method), 9
[return_method\(\)](#) (remcall.communication.send.Sender method), 10

S

[scalartypename\(\)](#) (remcall.generate.CSharpCodeGenerator method),

14
Schema (class in remcall.schema.core), 12
schema_from_bytes() (in module remcall.codec.read), 8
schema_to_bytes() (in module remcall.codec.write), 9
SchemaReader (class in remcall.codec.read), 8
SchemaWriter (class in remcall.codec.write), 8
send_schema() (remcall.communication.send.Sender method), 10
Sender (class in remcall.communication.send), 10
sort_key (remcall.schema.core.Type attribute), 11
stream_counter (remcall.util.QueueStream attribute), 15

T

Type (class in remcall.schema.core), 11
type_name() (remcall.naming.IdentityNameConverter method), 15
type_name() (remcall.naming.PythonNameConverter method), 15
type_names (remcall.generate.CSharpCodeGenerator attribute), 14
type_order (remcall.schema.core.Array attribute), 11
type_order (remcall.schema.core.Enum attribute), 12
type_order (remcall.schema.core.Interface attribute), 12
type_order (remcall.schema.core.Record attribute), 12
type_order (remcall.schema.core.Type attribute), 11
type_schemas (remcall.schema.core.Schema attribute), 13
type_table (remcall.schema.core.Schema attribute), 13
typename() (remcall.generate.CSharpCodeGenerator method), 14
TypeRef (class in remcall.schema.typeref), 13
TypeWrapper (class in remcall.util), 15

U

UnknownCommand, 13
UnknownImplementationObjectReference, 13
UnknownProxyObject, 13
UnknownType, 13

V

view_hex() (in module remcall.util), 15

W

wait_for_method_return() (remcall.communication.receive.Receiver method), 10
write() (remcall.util.QueueStream method), 15
write_bytes() (remcall.codec.write.WriterBase method), 8
write_enum() (remcall.codec.write.SchemaWriter method), 8
write_enum() (remcall.generate.CSharpCodeGenerator method), 14

write_enum_value() (remcall.communication.send.Sender method), 10
write_float32() (remcall.codec.write.WriterBase method), 8
write_float64() (remcall.codec.write.WriterBase method), 8
write_int16() (remcall.codec.write.WriterBase method), 8
write_int32() (remcall.codec.write.WriterBase method), 8
write_int64() (remcall.codec.write.WriterBase method), 8
write_int8() (remcall.codec.write.WriterBase method), 8
write_interface() (remcall.codec.write.SchemaWriter method), 8
write_interface() (remcall.generate.CSharpCodeGenerator method), 14
write_method() (remcall.codec.write.SchemaWriter method), 8
write_method() (remcall.generate.CSharpCodeGenerator method), 14
write_method_ref() (remcall.codec.write.WriterBase method), 8
write_name() (remcall.codec.write.SchemaWriter method), 8
write_object_ref() (remcall.communication.send.Sender method), 10
write_record() (remcall.codec.write.SchemaWriter method), 8
write_record() (remcall.generate.CSharpCodeGenerator method), 14
write_record_value() (remcall.communication.send.Sender method), 10
write_request_id() (remcall.communication.send.Sender method), 10
write_schema() (in module remcall.codec.write), 9
write_schema() (remcall.codec.write.SchemaWriter method), 8
write_schema() (remcall.generate.CSharpCodeGenerator method), 14
write_signed_integer() (remcall.codec.write.WriterBase method), 9
write_string() (remcall.codec.write.WriterBase method), 9
write_to_stream() (remcall.codec.write.SchemaWriter method), 8
write_to_stream() (remcall.codec.write.WriterBase method), 9
write_to_stream() (remcall.communication.send.Sender method), 10
write_type_ref() (remcall.codec.write.WriterBase method), 9
write_uint16() (remcall.codec.write.WriterBase method), 9
write_uint32() (remcall.codec.write.WriterBase method),

[9](#)
write_uint64() (remcall.codec.write.WriterBase method),
[9](#)
write_uint8() (remcall.codec.write.WriterBase method), [9](#)
write_unsigned_integer() (rem-
call.codec.write.WriterBase method), [9](#)
write_value() (remcall.communication.send.Sender
method), [11](#)
writeline() (remcall.generate.CSharpCodeGenerator
method), [14](#)
WriterBase (class in remcall.codec.write), [8](#)
WrongNumberOfBytesRead, [13](#)