



# Prototyping and writing Skype extras using Python

PyCon Due 2008, Arkadiusz Wahlig



## Table of contents

- Low-level Skype API
- **Skype4Py**
- Problems
- Future
- Software using Skype4Py
- Reference



## Low-level Skype API

- Text commands based
- Commands are same for all supported desktop platforms
- Commands are asynchronous



## Example API session

1. Application → Skype  
"GET USER jsmith ONLINESTATUS"
2. Application sleeps waiting for reply
3. Skype → Application  
"USER jsmith ONLINESTATUS NA"
4. Some time passes
5. Skype → Application  
"USER jsmith ONLINESTATUS ONLINE"



## Command identifiers

1. Application → Skype  
“#1 GET USER jsmith ONLINESTATUS”
2. Application sleeps waiting for reply
3. Skype → Application  
“#1 USER jsmith ONLINESTATUS NA”
4. Some time passes
5. Skype → Application  
“USER jsmith ONLINESTATUS ONLINE”



## Command types

- Single commands:  
“FOCUS”, “MINIMIZE”, ...  
Response is the command itself.
- Variables:  
“CONNSTATUS”, “USERSTATUS”, ...  
Response is the command itself.
- Object properties (get):  
“GET USER jsmith FULLNAME”, ...  
Response is the command without “GET”.



## Command types

- Object properties (set):  
“SET SMS 123 BODY Hello!”, ...  
Response is the command without “SET”.
- Altering objects (analogous to calling methods):  
“ALTER SMS 123 SEND”, ...  
Response is the command itself.



## Command types

- In all command types, the Skype can also reply with an error:  
“ERROR <errno> <errstr>”
- Error numbers are documented in the low-level API reference.





## API transports

- Platform dependant
- Use native IPC mechanisms
- Responsible for attaching procedure



## API transports

- Linux
  - X11 messaging
  - DBus
- Windows
  - WinAPI window messaging (WM\_COPYDATA)
- Mac OS X
  - Distributed Notification Center



## API versioning

- Client version
- Protocol version
- To ensure compatibility, application can set the Skype client to use particular protocol version
- Selected using command:  
"PROTOCOL 5"



## API wrappers

- Skype4COM
- Skype4Java
- **Skype4Py**
- Wrappers comparison



## Skype4COM

- Official Skype wrapper
- Closed (with plans to open)
- Windows only
- Usable in all COM-capable languages
  - including Python using win32 extensions and IronPython
- Object oriented interface



## Skype4Java

- Community wrapper officially supported by Skype
- Open source (Apache 2.0 License)
- Multiplatform (Linux, Windows, Mac OS X)
- Custom OO interface
- Unupdated for quite a long time



## Skype4Py

- Community wrapper officially supported by Skype
- Open source (BSD license)
- Multiplatform (Linux, Windows, Mac OS X)
- Skype4COM's OO interface
- As the only one, supports the newest additions to the low-level API



## API wrappers

	Skype4COM	Skype4Java	Skype4Py
<b>Platforms</b>	Windows	Windows, Linux, Mac OS X	Windows, Linux, Mac OS X
<b>License</b>	Proprietary	Apache 2.0	BSD
<b>Maintainer</b>	Skype	Community with Skype support	Community with Skype support
<b>Language</b>	COM	Java	Python
<b>API version</b>	3.2	2.5	<b>3.6</b>
<b>Latest release</b>	1.0.29.0 (2.04.2008)	1.0 (30.09.2006)	1.0.29.0 (10.05.2008)
<b>Notes</b>	Installed with the client		





## Skype4Py

- Installing
- Enums
- Classes
- Errors
- First steps
- Example I
- Event handlers
- Example II
- Advanced usage
- Distribution



## Installing

- Requires Python 2.4 or newer
  - On 2.4, `ctypes` has to be installed separately
- Standard Python installation  
`python setup.py install`
- A setup executable for Windows users
- Package name: `Skype4Py`



## Enums

- Various Skype4Py properties return or can be set to one value from a set.
- In Skype4Py, enum is a group of variables with common prefix in their name.
- Their names are inherited from Skype4COM.
- All enum variables are defined in `Skype4Py.enums` module.
- Variables are autoimported into the main `Skype4Py` package namespace.



## Classes overview

- Classes are defined in various modules inside the `Skype4Py` package.
- All classes inherited from `Skype4COM` are prefixed with a capital letter "I".



## Classes overview

- `Skype4Py.skype.ISkype`
- `Skype4Py.client.IClient`
- `Skype4Py.profile.IProfile`
- `Skype4Py.settings.ISettings`
- `Skype4Py.conversion.IConversion`
- `Skype4Py.user.IUser`
- `Skype4Py.chat.IChat`
- `Skype4Py.chat.IChatMessage`
- `Skype4Py.call.ICall`
- `Skype4Py.voicemail.IVoicemail`
- `Skype4Py.sms.ISmsMessage`
- `Skype4Py.filetransfer.IFileTransfer`



## Skype4Py.skype.ISkype

- The main Skype4Py class
- The only class instantiated directly
- Creates (in-)directly all other objects
- Manages the connection to the client
- Manages the event handlers



## Skype4Py.skype.ISkype

- Gives access to main functionalities
  - Contact list
  - Groups
  - Calls
  - Chats
  - Voicemails
  - User profile
  - Settings
  - SMSs



## Skype4Py.client.IClient

- A singleton always available in `Client` property of the `ISkype` class.
- Controls the client UI
  - Start or shutdown the client
  - Add menu items and events
  - Simulate key presses
  - Open various dialogs and wizards





## Skype4Py.profile.IProfile

- A singleton always available in `CurrentUserProfile` property of the `ISkype` class.
- Provides read/write access to user profile
  - Full name
  - Mood text
  - City / country
  - Homepage / About
- Gives access to account balance information



## Skype4Py.settings.ISettings

- A singleton always available in `Settings` property of the `ISkype` class.
- API related client settings
  - Audio input/output devices names
  - Ringer device name
  - Video device name
  - Client language
  - Avatars
  - Ringtones



## Skype4Py.conversion.IConversion

- A singleton always available in `Convert` property of the `ISkype` class.
- Provides functions to convert enum values to human readable text in a selected language
- Provides functions to convert text constants into enums



## Skype4Py.user.IUser

- Not a singleton
- Represents any Skype user
- Provides read-only access to user information
  - Online status
  - User profile
  - Avatars
  - Features support (calls, video calls, voicemail)
  - Is authorized, is blocked
  - Last online datetime
- For non-authorized users, not all properties may be available



## Skype4Py.chat.IChat

- Represents a Skype chat (dialogs, multichats, public chats)
- Controls a chat
  - Members
  - Messages
  - Statistics
  - Topic
  - Options
  - Public chats extensions



## Skype4Py.chat.IChatMessage

- Represents a single message posted on a chat
- Message details
  - Body
  - Datetime
  - Sender
  - Status
  - Seen status
- Message editing including editor details



## Skype4Py.call.ICall

- Represents a Skype call, both video and audio only
- Controls a call
  - Holding, resuming, finishing
  - Capturing/feeding audio streams
  - Controlling video transmission
  - Transferring
  - Statistics
- Supports conferences



## Skype4Py.voicemail.IVoicemail

- Represents a voicemail
  - Downloaded from server (a voicemail for us)
  - Recorded locally (to be sent to another user)
- Provided functions
  - Upload / download
  - Record / playback
  - Statistics
  - Upload





## Skype4Py.sms.ISmsMessage

- Represents an SMS message
- Currently only outgoing messages supported
- Provided functionalities
  - Aid in message creation
  - Price calculation
  - Multiple targets
  - Status
  - Send / delete



## Skype4Py.filetransfer.IFileTransfer

- Represents a single file transfer, both incoming and outgoing
- Provided functionalities
  - Transfer statistics
  - File information
  - Partner
- Currently there is no way to start a file transfer without user interaction



## Errors

- `Skype4Py.errors.ISkypeError`  
Errors reported by Skype over the low-level API (“ERROR” reply)
- `Skype4Py.errors.ISkypeAPIError`  
Errors caused by a failure in the transport layer



## First steps

`Skype4Py . Skype`  
is an alias for  
`Skype4Py . skype . ISkype`  
class.



## First steps

```
import Skype4Py
```

```
skype = Skype4Py.Skype()
```



## Selecting a transport

```
import Skype4Py
```

```
skype = Skype4Py.Skype(Transport="dbus")
```



## Enabling API debug

```
import Skype4Py
```

```
skype = Skype4Py.Skype(ApiDebugLevel=1)
```



## Attaching to the client

```
import Skype4Py

skype = Skype4Py.Skype()
skype.FriendlyName = "PyCon2"
skype.Attach()
```





## Example I

- Attach to the client
- Enumerate the contact list
- Send a chat message to online/away Skype users saying hello using their full name
- Send the same as an SMS message to SkypeOut users
- Print the usernames and send datetimes.



## Example I

```
import Skype4Py

if __name__ == "__main__":
    skype = Skype4Py.Skype()
    skype.FriendlyName = "Example I"
    skype.Attach()
```



## Example I

```
for user in skype.Friends:
    if user.IsSkypeOutContact:
        m = send_sms_message(skype,
                             user)
    else:
        m = send_chat_message(skype,
                              user)
    print user.Handle, m.Datetime
```



## Example I

```
def send_chat_message(skype, user):  
    if user.OnlineStatus in \  
        (Skype4Py.olsOnline,  
         Skype4Py.olsAway):  
    return skype.SendMessage( \  
        user.Handle,  
        "Hello %s!" % user.FullName)
```



## Example I

```
def send_sms_message(skype, user):  
    return skype.SendSms(user.Handle,  
        Body="Hello %s!" % \  
        user.FullName)
```



## Things to notice

- Skype4Py uses `property()` to implement properties in all classes, this means that simply-looking operations can cause a lot of internal processing.
- Since we needed the `Skype` object in all functions, it would be wise to build the example as a class with `Skype` object being a property.



## Ideas for extending the example

- The `Sex` property of an `IUser` object could be used to improve the sent message (“Hello Mr/Ms ...”).
- We could use the `Timezone` property to calculate the users local time and send him messages every morning.
- Finally we could call the contacts and stream a voice welcome message to them.
- ...



## Event handlers

- Allow reacting on certain events occurring in Skype
- Handlers are callables asynchronously triggered by Skype4Py on separate threads (using the `threading.Thread` class)
- Handlers of the same event type run serially on a single thread





## Event handlers

- There are three ways to assign a callable to an event
  - Using a class encapsulating many handlers of different events
  - Using the `On...` properties
  - Using the `RegisterEventHandler` method



## Handlers in a class

```
import Skype4Py

class MySkypeEvents:
    def UserStatus(self, Status):
        print "Status changed to %s" % Status

skype = Skype4Py.Skype( \
    Events=MySkypeEvents())
```



## Handlers using the On... properties

```
import Skype4Py

def user_status(Status):
    print "Status changed to %s" % Status

skype = Skype4Py.Skype()

skype.OnUserStatus = user_status
```



## Handlers using the RegisterEventHandler method

```
import Skype4Py

def user_status(Status):
    print "Status changed to %s" % Status

skype = Skype4Py.Skype()

skype.RegisterEventHandler('UserStatus', \
    user_status)
```



## Event handlers

- Common to all three ways are
  - The name of the event
  - The list of arguments accepted by the handler
- For reference: the names and arguments lists can be found in `Skype4Py.skype.SkypeEvents` class



## Example II

- Attach to the client and stay attached
- Wait for incoming chat messages, detect if they are known commands, execute them and send a reply
- The commands:
  - `@userstatus <status>`  
Changes the online status of the bot
  - `@credit`  
Returns the current Skype account balance of the bot



## Example II

```
import Skype4Py
import time
import re

(...)

if __name__ == "__main__":
    bot = SkypeBot()

    while True:
        time.sleep(1.0)
```



## Example II

```
class SkypeBot(object):  
  
    def __init__(self):  
        self.skype = Skype4Py.Skype( \  
            Events=self)  
        self.skype.FriendlyName = \  
            "Skype Bot"  
        self.skype.Attach()
```





## Example II

```
def AttachmentStatus(self, status):  
    if status == \  
        Skype4Py.apiAttachAvailable:  
        self.skype.Attach()
```



## Example II

```
def cmd_userstatus(self, status):  
    if status:  
        try:  
            self.skype.ChangeUserStatus(\  
                status)  
        except Skype4Py.SkypeError, err:  
            return str(err)  
    return 'Current status: %s' % \  
        self.skype.CurrentUserStatus
```



## Example II

```
def cmd_credit(self):  
    return self.skype.          (...)  
        CurrentUserProfile.BalanceToText
```



## Example II

```
commands = {  
    "@userstatus *(.*)": \  
        cmd_userstatus,  
    "@credit$": cmd_credit  
}
```



## Example II

```
def MessageStatus(self, msg, status):  
    if status == Skype4Py.cmsReceived:  
        if msg.Chat.Type in (Skype4Py.chatTypeDialog,  
                               Skype4Py.chatTypeLegacyDialog):  
            for regexp, target in self.commands.items():  
                match = re.match(regexp, msg.Body,  
                                re.IGNORECASE)  
                if match:  
                    msg.MarkAsSeen()  
                    reply = target(self, *match.groups())  
                    if reply: msg.Chat.SendMessage(reply)  
                    break
```



## Advanced usage

- Bypassing the object interface
- Application-to-Application



## Bypassing the object interface

- When to do it
  - If new commands are added to the low-level API
  - For some reason we can't use a newer version of Skype4Py
  - There are bugs or limitations in Skype4Py (in this case it should also be reported to the developers)



## Bypassing the object interface

```
import Skype4Py

skype = Skype4Py.Skype()

skype.SendCommand( skype.Command( \
    "FOCUS" ) )
```





## Bypassing the object interface

```
skype.SendCommand( skype.Command( \
    "FOCUS",
    Block=True,
    Timeout=10000))
```



## Bypassing the object interface

```
cmd = skype.Command("FOCUS",  
    Block=True,  
    Timeout=10000)
```

```
skype.SendCommand(cmd)
```

```
print cmd.Reply
```



## Application-to-Application

- Creates a data connection between two Skype users
- No user interface controls involved
- Supports reliable and unreliable (but faster) data transfers



## Skype4Py.application.IApplication

- Called an application object for short
- Represents one end of the connection
- Defined by a well-known name
- To create a connection, both users have to create an application with the same name
- More than one application can exist at the same time (with different names)



## Skype4Py.application.IApplication

- An application context exists in the client. Deallocation of the Python application object does not destroy it. It must be done explicitly.
- A connection between users is represented by an `IApplicationStream` object
- Single application can have many streams but only one per remote user



## Skype4Py.application.IApplicationStream

- Represents a connection to a single user
- Provides methods to exchange data between the two users



## Application events

- There are five event types triggered by the App2App protocol
  - `ApplicationConnecting(App, Users)`
  - `ApplicationDatagram(App, Stream, Text)`
  - `ApplicationReceiving(App, Streams)`
  - `ApplicationSending(App, Streams)`
  - `ApplicationStreams(App, Streams)`



## Example III

```
import Skype4Py, time

if __name__ == "__main__":
    skype = Skype4Py.Skype()
    skype.OnApplicationReceiving = receiving
    app = skype.Application("PyConApp")
    app.Create()
    try:
        while True: time.sleep(1.0)
    finally:
        app.Delete()
```





## Example III

```
def receiving(app, streams):  
    for stream in streams:  
        print stream.Read()
```



## Example III

```
import Skype4Py, sys

if __name__ == "__main__":
    skype = Skype4Py.Skype()
    app = skype.Application("PyConApp")
    app.Create()
    try:
        stream = app.Connect("arkadiusz.wahlig", True)
        stream.Write(" ".join(sys.argv[1:]))
    finally:
        app.Delete()
```



## Distribution

- As any other Python script
- Prototyping with Skype4Py, final version with Skype4COM
- Cython (in future)
- py2exe on Windows



## Problems

- Skype makes the API available only when the user has logged into the client
- There is no way to log in as a different user
- Skype lets the user choose which programs can access it, in case of Skype for Windows programs are distinguished by the executable module name (“python.exe” for Python)



## Future

- Keep the compatibility with earlier versions
- Keep the library up to date with Skype and Skype4COM development
- Compatibility with Cython
- Add additional, task oriented APIs; for example a gaming API
- Add common dialogs (for example a user picker) in a multiplatform way
- Write more examples



## Software using Skype4Py

- Some projects using Skype4Py
  - **SkySentials** (Phillip Kolmann)  
<http://www.kolmann.at/philipp/linux/>
  - **Skype plugin for BitlBee** (Miklos Vajna)  
<http://vmiklos.hu/project/bitlbee-skype/>
  - **Command-line tools for Skype** (Vincent Oberle)  
[http://www.oberle.org/skype\\_linux\\_tools](http://www.oberle.org/skype_linux_tools)
  - **SkypeStatus docklet**
  - **SkypeTunnel**



## Reference

- **Skype4Py wiki home**  
<https://developer.skype.com/wiki/Skype4Py>
- **Skype4Py Reference Manual**  
<http://skype4py.sourceforge.net/doc/html/>
- **Skype Developer Zone**  
<https://developer.skype.com/>
- **Low-level API Reference**  
<https://developer.skype.com/Docs/ApiDoc>



## Python Magazine



### Extending Skype using Python

by Arkadiusz Wahlig

Skype is one of the most popular VoIP clients and instant messengers, with features like voice and video calls, and chats. What isn't as well known is the fact that Skype's functionality can be extended using external programs. This article explains how to create extensions using Python thanks to the Skype4Py library.

**PyMag; Feb. 2008 (Volume 2 Issue 2)**

<http://pymag.phparch.com/c/issue/view/68>





Thank you!

D?

R!