Skybot

Steve Landers and Mike Doyle

Iomas Research LLC

Abstract

The Skype API provides facilities for external applications to be able to communicate with and control various aspects of the Skype VOIP and IM application. Originally developed in early 2005, Skybot uses the power and flexibility of Tcl to dramatically expand the usefulness of the Skype system. Using Skybot, one can create a variety of programmable responses to incoming Skype calls and chat messages. The system incorporates a Tcl-based state machine which can be configured to automatically trigger Tcl code in response to various Skype events. This provides the user with a surprisingly-broad and powerful set of capabilities for automating Skype-based communication operations.

Introduction

Iomas Research [1] is an Internet technology research and development group based in Wheaton, Illinois. Iomas team members use the Skype [2] product extensively to facilitate collaboration across locations and timezones.

Skype is a software system that allows users to make audio and video calls over the Internet. It supports conference calls, instant messaging, file transfer and video conferencing. Skype uses a proprietary Internet telephony (VoIP) network that works on a peer-to-peer model.

One of Skype's distinguishing features is that it can operate behind firewalls and NAT routers. It achieves this by using super-nodes to relay calls between clients not directly connected to the Internet. Any Skype client can become a super-node if it connects to the Internet without NAT, which means Skype can and does “borrow” bandwidth and computing power.

Skybot was born out of the desire to minimize this relaying by automatically taking Skype offline when the user's computer becoming idle. Subsequently more advanced features were added to make it an electronic secretary by monitoring Skype calls and chats, and automatically responding to events using pre-defined actions.

Skybot – the first generation

The original Skybot concept was a tool that monitored the state of the computer and terminated Skype when the user was inactive, restarting it when the user became active again.

The main issue with this approach was how to detect if the computer was idle.

At the time that Skybot originated (early 2005) there was no way of achieving this directly via Tcl/Tk (TIP 245 - “Discover User Inactivity Time” had not yet been proposed [3]). Given that the initial users were on Windows, a Tcl program was written using the TWAPI [4] extension that detected when the screensaver was active.

TWAPI (“The Tcl Windows API”) provides access to Win32 API's on Windows platforms. It provides a number

of commands covering a wide variety of features, including system functions, user management, security and access control, process management and window management.

An alternative to TWAPI would have been Ffidl [5]– an extension that allows a Tcl program to invoke shared library functions without C glue. It could have been used to invoke functionality in the the win32.dll directly, but TWAPI was chosen because it provided higher-level abstractions in addition to the low-level interfaces provided by Ffidl.

For example, the following command checks if the screensaver is running:

```
if {[twapi::screensaver_running] } { .... }
```

There are also commands that can return a handle to a process, and to kill a process

```
set id [twapi::get_process_ids -glob -name Skype.exe]
twapi::endprocess $id -force
```

While effective, the approach of killing and restarting Skype wasn't particularly elegant nor efficient, so it was decided to interface to Skype via the Skype API [6] that had recently been published.

The Skype API supports three types of messages

- commands to control Skype
- directives – i.e. “out of band” commands that control the API itself
- information messages that Skype sends about its status

The Skype API provides all of the features needed by Skybot – including changing the user's status and taking Skype offline.

During the early days of Skybot the Skype API was only available for Windows, so a Windows specific Tcl interface was produced using Critcl [7] and the Windows Messaging API. This extension (called “winmsg”) is 140 lines of code, and includes commands to register callbacks, send messages and retrieve data from incoming messages using the Windows COPYDATA facility.

The Winico [8] extension was used to interface with the Windows taskbar, allowing Skybot to display an icon and associated menu in the system tray.

A bug in the Tk 8.4 Windows code that caused menu entry images to be incorrectly displayed was worked around by temporarily disabling menu animations when pop-up menus are posted:

```
proc PopupMenu {menu x y} {
    set mem [twapi::malloc 4]
    set SPI_GETMENUANIMATION 0x1002
    set SPI_SETMENUANIMATION 0x1003
    twapi::SystemParametersInfo $SPI_GETMENUANIMATION 0 $mem 0
    binary scan [twapi::Twapi_ReadMemoryBinary $mem 0 4] i enabled
    if {$enabled} {
        twapi::SystemParametersInfo $SPI_SETMENUANIMATION 0 NULL 0
    }
    tk_popup $menu $x $y
    if {$enabled} {
        twapi::SystemParametersInfo $SPI_SETMENUANIMATION 0 $mem 0
    }
    twapi::free $mem
}
```

Skybot is deployed as a single file, self-contained Starpack [9] executable that is approximately 2.5 Mbyte in size. It doesn't require any special installation - it can be dropped onto the desktop or copied to any directory and run from there.

When Skybot starts it adds an icon to the Windows system tray then checks the Windows registry to confirm that Skype is installed and running (if necessary it will start Skype).

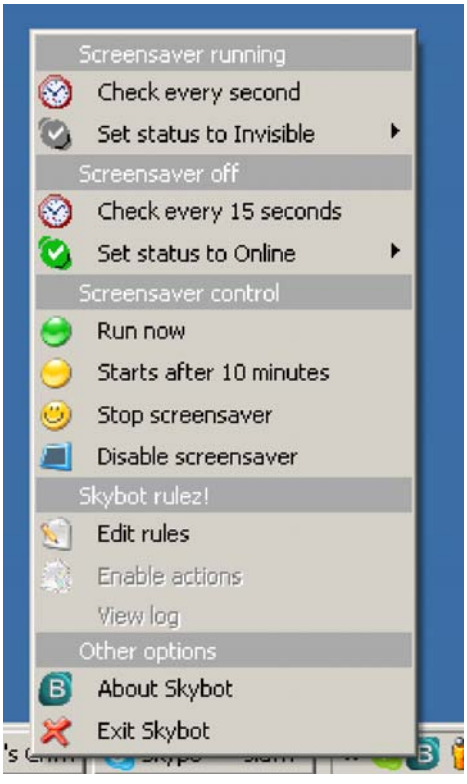
The first time Skybot is run Skype will ask the user to confirm that Skybot is allowed to access Skype. This will be repeated whenever the Skybot executable changes (for security Skype maintains a hash of any authorized programs and will detect if they change).

Skybot then starts monitoring the Windows screensaver at a specified interval and requests Skype change the user state as appropriate:

- when the user is active (i.e. the screensaver is inactive) Skybot checks the screensaver state every 15 seconds
- when the user is idle (i.e. the screensaver is active) Skybot checks the screensaver state once per second the default active state for Skype is “Online” and the inactive state is “Invisible”.

This means that Skype is put into the “Invisible” state within 15 seconds of the screensaver starting and brought back “online” within a second of the user becoming active again. These values are configurable from the Skybot menu.

The “Invisible” state is a reasonable default during work hours. It doesn't prevent relaying (at least, as far as we can tell) but saves the overhead of connecting and disconnecting.



The user can click on the Skybot icon in the System Tray and see the menu that is used for configuring Skybot.

The menu also contains options to

- run the screensaver immediately (useful when leaving the computer for any reason)
- set the idle time before the screensaver starts
- disable the screensaver (useful in presentations)

A right click on the Skybot icon presents a menu of currently connected Skype buddies, allowing quick access for initiating a chat or call. Also, the Skybot icon changes color to inform the user if there is a problem with Skype or it isn't running.

This first version of Skybot proved to be very useful, but thoughts soon turned to extending the functionality further.

It was recognized that the current Skybot features were just specific instances of a general rule/action action approach.

So some questions were asked:

- what if Skybot had an extension language that could be used to specify rules and actions?
- what if Skybot had a generalized state engine to evaluate the rules?
- what if the actions could access state and configuration information?

- what if these rules could be stored in a database for easy management

... then you'd have "Skybot Rulz!"

Skybot Rulz!

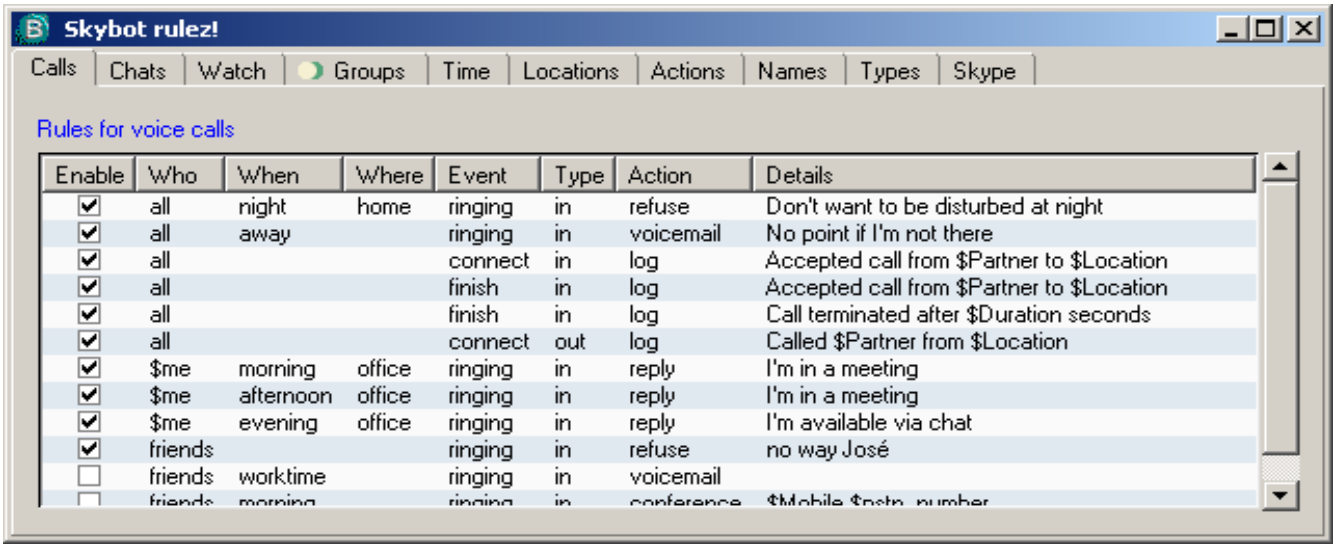
Skybot was extended with a rules-based state engine that supports event recognition and automated responses specified in Tcl.

The rules in the first version are relatively simple – they allow the detection of a set of pre-defined events and conditions. The design allows for the definition of complex interactions using a full state-machine approach.

The rules are stored in a normalized database, and presented to users as tables. There is a single maintenance screen with multiple tabs – one for each table in the database.

There are three classes of rules
- call rules
- chat rules
- watch rules (watch for someone changing to a particular status)

Defining call rules



Enable	Who	When	Where	Event	Type	Action	Details
☑	all	night	home	ringing	in	refuse	Don't want to be disturbed at night
☑	all	away		ringing	in	voicemail	No point if I'm not there
☑	all			connect	in	log	Accepted call from $Partner to $Location
☑	all			finish	in	log	Accepted call from $Partner to $Location
☑	all			finish	in	log	Call terminated after $Duration seconds
☑	all			connect	out	log	Called $Partner from $Location
☑	$me	morning	office	ringing	in	reply	I'm in a meeting
☑	$me	afternoon	office	ringing	in	reply	I'm in a meeting
☑	$me	evening	office	ringing	in	reply	I'm available via chat
☑	friends			ringing	in	refuse	no way José
☐	friends	worktime		ringing	in	voicemail	
☐	friends	morning		ringing	in	conference	$Mobile $pstn_number

The first thing to note is that rules may be defined but not enabled. This makes it easier to create rules without affecting the Skybot behavior.

Now take a look at the first rule, which specifies that Skype shouldn't accept calls at night when the user is at home. Note the following:
- Who (i.e. “all”) is defined in the “Groups” tab, where users and groups are defined – the icon next to “all” signifies a group rather than an individual
- When (i.e. “night”) occurs in the “Time” tab
- Where (i.e. “home”) occurs in the “Location” tab
- Event (i.e. “ringing”) corresponds to the Skype event of the same name
- Type (i.e. “in”) refers to the direction of the call – either in or out.
- Action (i.e. “refuse”) is defined in the Action tab
- the Details column is a descriptive comment that can be used in the rule definition

This emphasizes a key point about the Skybot design – all rules and actions are stored as data in tables, and are abstracted and normalized.

Defining users and groups

Now we can look at some of the definitions – first the Who value as defined in the Groups tab:

There are two groups defined by default:

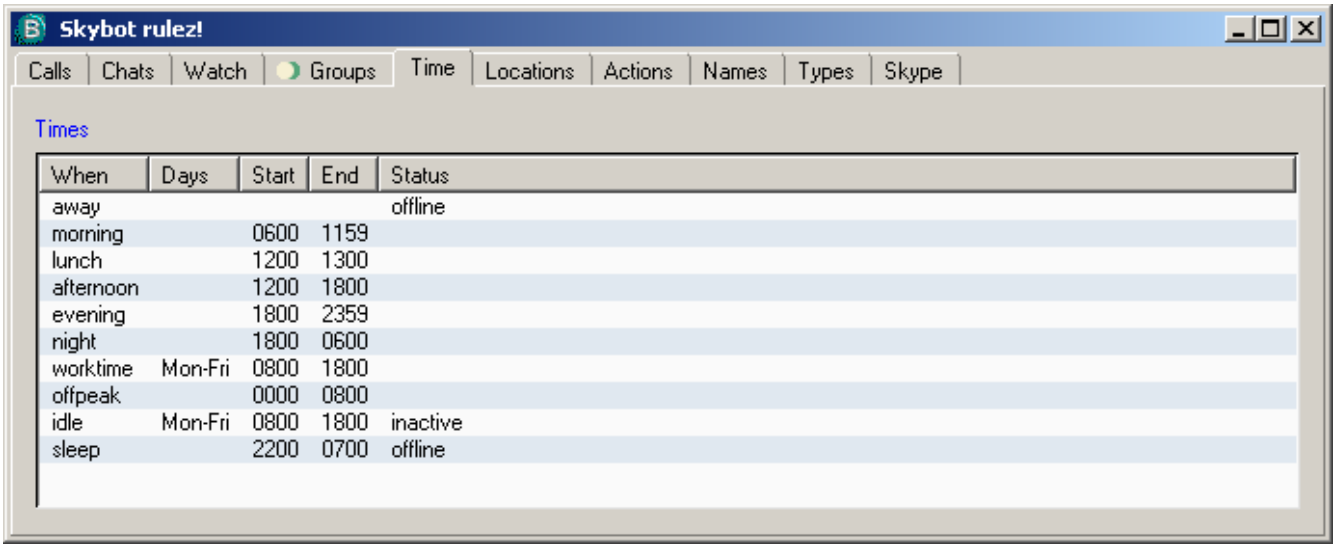
- all – which is a wild card that matches all Skype users
- friends – which is the list of people on the user's Skype contact list (as returned by the Skype API call "SEARCH FRIENDS").

Alternatively, you can enter a Skype user name or a pattern (e.g."iomas-*").or specify groupings – for example "wheaton" might expand to “cyndy maury mike”..

Groups can be recursive, as in
wheaton = “cyndy maury mike”
iomas = “wheaton steveh stevel”

This scheme also allows convenient aliases to map to real Skype user names – for example, the above names might be aliased to iomas-cyndy, iomas-maury and so on. During data entry, a check is performed to ensure a group isn't defined with the same name as an alias or a Skype name on the users contact list.

Defining times



When specifying time, you can pick from a pre-defined list of time definitions. The Time tab allows you to add your own time definitions or edit the existing ones. Time definitions can be based on time ranges, on the current status or both.

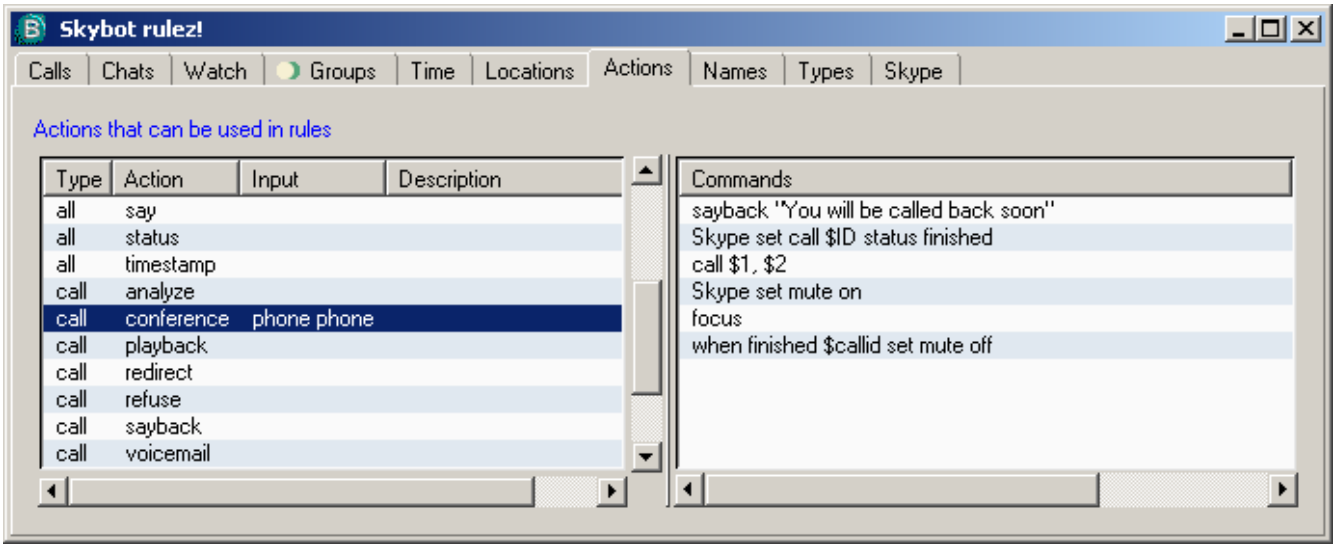
Days can be a list (space or comma separated), or a range "Mon-Fri" or "Monday-Friday”.

Defining Locations

Locations are based on IP address, and are defined in the Locations tab. Skybot recognizes new a location and pops up a dialog so the user can give it a name (e.g. Home, Office, etc)

Defining Actions

The available actions depend on the type of rule, and are defined in the Action tab.



Note that there are rules that apply to calls, chats, watches and to all events. In the above example, you can see the Tcl commands that are run when the event is detected. These make use of some pre-defined Tcl commands but the intention is to eventually allow definition of arbitrary Tcl procedures in the database as well.

The "when" command creates a Tcl trace on the call status variable, allowing event driven actions within rules.

The Details variable is passed in from the call/chat/watch rule, and will be substituted, so that any variables it contains will be expanded. This will allow rules to be parameterized, for example:

Who	When	Action	Details
iomas-*	night	reply	"$firstname, I'm still sleeping"

Useful in an organization that spans timezones, or where team-members are semi-nocturnal.

Because rules and actions are just data stored in database tables, Skybot is both flexible and extensible – and can be extended by adding new rows to the tables in the database, or by modifying existing rows.

Call actions could include the following:

accept	accept the call
refuse	don't answer call
message	send Instant Message to either the user, the caller or both via an IM gateway
reply	reply to caller with pre-defined chat message
mail	send mail to me/caller/whoever
play	play sound clip to audio output device or caller
say	"speech to text" to audio output device or caller
voice mail	send call to voice mail
log	log call details (start time, then end time)
conference	hang up call then initiate conference call with pre-defined numbers
redirect	hangs up call, then initiates conference with
mute	mute speakers

For example, to refuse all incoming calls when in a meeting, but let Mike know that the user is available via chat, use something like the following Calls definitions (abbreviated for clarity)

Who	When	Action	Details
all	morning	refuse	
mike	morning	reply	I'm available via chat
default	morning	reply	I'm unavailable

The "default" rule will only match if there is no other matching response – in this case Mike would see "I'm available via chat" but everyone else would see "I'm unavailable"

Note that there are a set of pre-defined variables available, corresponding to the call properties obtained from Skype (e.g. the above $pstn_number).

The derivation of the commands to execute involves a join of the calls/chat/watch table with the Groups, Time Location and Action table, then a substitution of variables in the Commands field of the action table.

Chat actions (when a new chat session starts)

reply	reply with pre-defined chat message
call	forward chat message to call, via text to speech program like t2s
mail	forward message via mail
message	send the user an Instant Message via an IM gateway
proxy	proxy the entire chat session to another IM system
play	play sound clip to audio output device
say	"speech to text" to audio output device
confirm	display a confirm dialog that allows the user to accept or reject the chat
log	log call details (start time, then end time)

Watch actions (when someone's status changes to a particular state) might include:

mail	send mail to me/caller/whoever
play	play sound clip to speakers and/or caller
say	“speech to text” to audio output device
call	call the user and play a text to speech file or sound clip
message	send an IM via the IM gateway
status	change the user's status

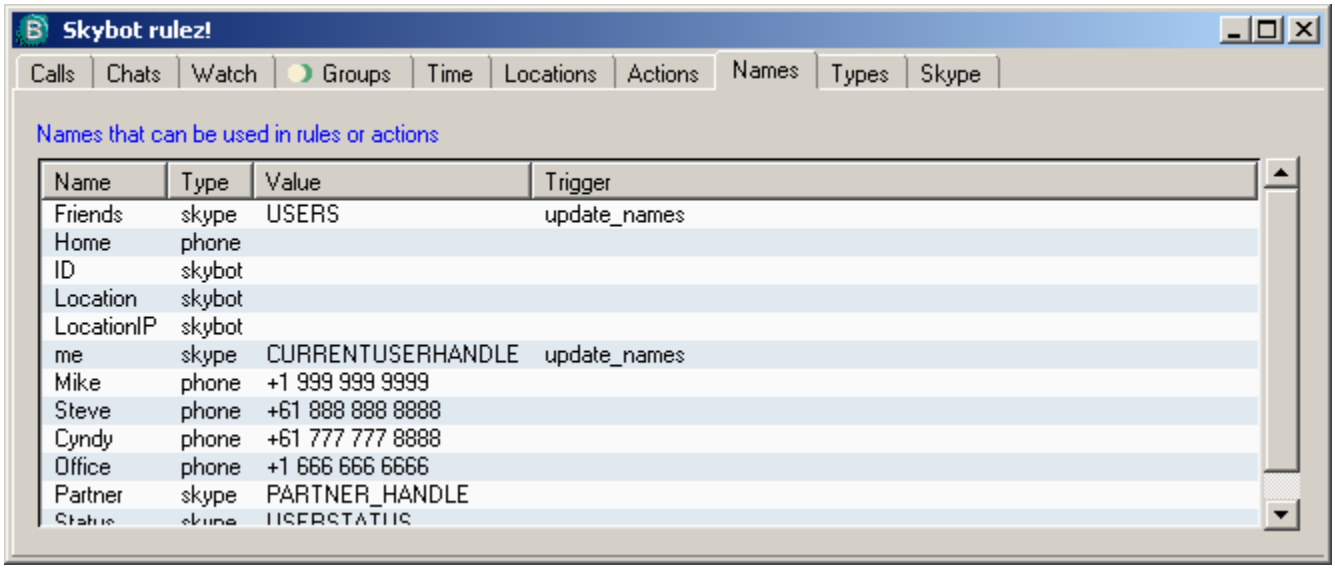
For example, to set the user's status to Offline when idle during prime time and to Invisible (to allow Skype proxying) when idle during non-prime time use the following (this would override the default screensaver-based idle behavior)

Who	When	Status	Action
$me	prime	screensaver	offline
$me	offpeak	screensaver	invisible

Note the “me” variable will always expand to the current username.

Names and types

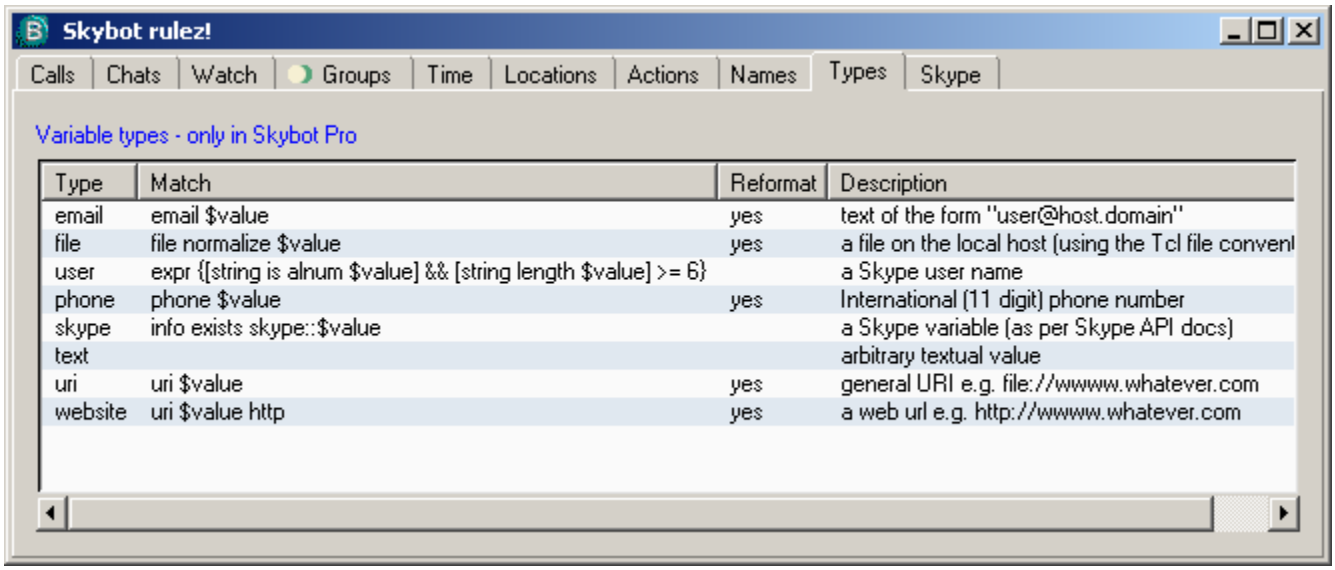
The Names tab allows the specification of variables that can be referred to in actions or rules.



There are a number of different types of variables supported:

Type	Description
text	arbitrary textual value
phone	an international phone number - 11 digits of the of the form "+nn nn nnn nnnn" or "+n nnn nnn nnnn"
email	text of the form "user@host.domain"
Skype	A Skype variable (as per Skype API docs)
website	A web url e.g. http://www.whatever.com
uri	a general uri
file	a file on the local host, using the Tcl file conventions (e.g. C:/wherever/whatever)

The variable types are defined in the Types tab using Tcl expressions. The reformat column specifies if the Tcl command returns a value that should replace the original value (useful, for example, when forcing phone numbers to a standard format).



Skype variables

The Skype tab exposes the internal mapping from Skype messages to the Skybot Tcl commands that parse them and invoke the state machine.

There are two types of data available from within Skype

- data that is "pushed" from Skype to Skybot when certain events occur (e.g. incoming call/chat)
- data that may be "pulled" when it needs to be used (e.g. user status)

Pushed data includes the incoming call or chat notification, the status of the call/chat, members in a chat, the chat topic, call duration on completion, call failure reason if there was a problem.

Note that all data may potentially be pushed by Skype, and so Skybot has to recognize it.

Other data needs to be pulled by Skybot from Skype - either when referred to by a rule or when a push occurs by Skype. For example, when an incoming call is detected Skybot needs to pull information about the call, including the partner_handle, partner_dispname, conf_id, type and pstn_number.

To define a Skype variable we specify the following

type	The type of variable
property	the Skype property / variable name
pattern	pattern to match on
postprocess	Tcl commands to post-process the value after retrieving it from Skype
pull	list of other data to pull when push is detected

Example – incoming call

When there is an incoming call Skype sends Skybot a message like the following

```
CALL $id STATUS RINGING
```

where $id is the Skype message identifier.

The pattern matching recognizes the message and calls the command

```
call $id $value
```

The call procedure sees that there isn't an existing call $id, and so creates a new one – creating a separate interpreter to hold its state. Skybot then pulls the Skype PARTNER_HANDLE property (which contains the Skype usename of the incoming call). Skybot also pulls any other values it needs including userstatus, etc.

Skybot then looks through the enabled rules for those whose “Who” value either matches the username exactly, or refers to a group that contains the username.

It then looks for those rules where the "When" value matches, likewise with “Where”.- and substitutes then evaluates any associated actions.

Current status

The Skybot Rulz! proof of concept was demonstrated in 2005 – the state engine is functional and many of the rules have been defined. But the productization of Skybot was deferred and it was stored in the half-bakery due to other priorities and pending the availability of the Skype API on Linux and MacOS X.

The intention is to complete Skybot and produce a free “lite” version that just manages Skype presence and a low-cost standard version with additional functionality, including rules to:

- call back when current call finishes
- send chat message when current call finishes
- change Skype icon when on a call
- initiate conference call
- send call to voice mail
- refuse call
- accept a call
- reply via chat
- send SMS via smsgateway service

Conclusion

Although starting life as a tool for the convenience of Iomas team members, Skybot demonstrates:

- a fully functional quality UI for Windows in Tcl/Tk
- the flexibility of Tcl as an embedded extension language
- the power of dynamic languages in all stages of the product cycle – from prototyping to user interface design
- the flexibility and extensibility of a data-centric design

There are a number of potential uses for Skybot to route incoming calls in specific ways, based on the identity of the caller and/or the time of day or day of the week.

From simple, practical solutions like during off hours, routing calls from a client to a work voice mail system, calls from your spouse to go to your cell phone, and calls from your boss to trigger an email alert to be sent to your PDA.

But the potential exists for some creative automation too. Want to play hookey from work one afternoon? Skybot could keep your online status looking like you're at your machine, periodically switching it between "online" and "away". If your boss sends you a chat message, it could initiate a conversation between him/her and a customized Eliza engine that simulates your presence by sending plausible yet non-committal chat responses, feigning some crisis that you have to drop off for, then sending the whole conversation to your Blackberry, or iPhome so you'll know to finish the hole you're golfing on and head back to the office.

References

1 Iomas Research - http://iomas.com
2 Skype - http://skype.com
3 TIP 245 - http://tip.tcl.tk/245
4 TWAPI - http://wiki.tcl.tk/twapi
5 Ffidl - http://wiki.tcl.tk/ffidl
6 Skype API - http://developer.skype.com/
7 Critcl - http://wiki.tcl.tk/critcl
8 Winico - http://wiki.tcl.tk/winico
9 Starpacks - http://wiki.tcl.tk/starpack