Planet SoC

Week 2 Updates: Improved Infrastructure

Mon, 06/11/2007 - 04:53 — jeff

Last week I concentrated on improving the infrastructure and architecture of the MySpaceIM Protocol Plugin for Pidgin (msimprpl). These improvements do not immediately result in any major user-visible changes (such as additional features) but will greatly ease the ability to add such features in the future. Besides enhancing extensibility, having a good architecture will also add stability since the program will be simpler and easier to understand from a programmer perspective. And more people will be able to contribute since the barrier to entry will be lower.

MySpaceIM internally uses numeric uids to refer to users, while externally presenting a corresponding username to the user. This adds some complications, expressed here in detail, so this post is quite long.

The first set of changes I committed were numerous enhancements to MsimMessage, which represents a MySpaceIM protocol message. MsimMessage, as defined and implemented in message.h and message.c, is now used for both sending (msim_msg_send())) and receiving messages (via msim_parse(), which handles the raw protocol data from msim_input_cb())), so MsimMessage is getting quite powerful. I added msim_msg_clone() to deeply copy a message, so that it can be kept around if the callee requires it but the caller destroys the message after calling the callee (replacing a strategy that used the return value to determine whether the message should be freed or not).

I changed msim_msg_new() to accept variadic arguments, creating a new message with the given elements. This makes creating new messages easier. msim_send() accepts the same variadic arguments, but sends the message immediately. msim_msg_new() is useful when a message is to be created but further manipulated before sending. Both functions call msim_msg_new_v() with a va_list to create the new message. msim_send() also has a GCC sentinel attribute, if compiled with GCC, so the compiler will warn if the terminating NULL of the variadic arguments is missing: __attribute__((__sentinel__(0))).

I changed msim_msg_pack(), which is used by msim_msg_send() to serialize an MsimMessage for transfer on the wire, to ignore elements with names beginning with an underscore. The main msimprpl module uses underscore fields for tagging certain information onto an MsimMessage, not intended for sending to the server, but useful to keep around on the message itself.

Protocol messages must have their fields in a certain order to be interpreted correctly (this was one of the reasons for developing MsimMessage, over using an (unordered) GHashTable) so I added msim_msg_insert_before() that allows for inserting message elements into a given position (complementing msim_msg_append). Lastly I added msim_msg_dump sends a message to the debug log using purple_debug_info.

Now that I had a powerful MsimMessage, I changed msim_status_cb(), msim_incoming_im_cb(), and msim_send_im_by_userid_cb() to receive an MsimMessage in their argument instead of a struct specific to that function. All these functions were callbacks for functions that required uid-to-username or username-to-uid name resolution. msim_incoming_im(), for example, would receive incoming instant messages addressed as coming from a user, by numeric uid. Numeric uid is not that useful when real people refer to users by their username, so msim_incoming_im()

would send a query to lookup the username by the uid, and set up the msim_incoming_im_cb() to be called with the username response, along with the instant message data.

Needless to say, this method of having a callback for each incoming protocol message that needs to have a uid resolved is cumbersome. A better approach would be to push the uid-to-username resolving down below the message-specific processing functions. I added msim_preprocess_incoming() to perform the resolving on incoming messages, before the message ever reaches msim_process() (which dispatches the message to the appropriate handler, such as msim_incoming_im(), depending on its contents). If a message needs to be "preprocessed", msim_preprocess_incoming() will send a uid-to-username query, and once the query arrives a username field will be appended to the message, which will then be sent to msim_process() for handling. This means that msim_status() and msim_incoming_im() no longer have the associated callback functions--they just look at the username field that has been added by preprocessing. The handlers know nothing of uid-to-username resolving, nor do they longer need to (decoupling).

msim_preprocess_incoming() still sets up a callback function to tag a username (using the aforementioned underscore fields) on to incoming messages containing uids, so callbacks are not entirely removed. Sean Egan suggested keeping a data structure (such as GList) in my proto_data struct (MsimSession) containing the outstanding messages, which would be a more elegant solution removing the need for callback functions, and would also make timeouts easier to handle (if the uid-to-username request does not provoke a reply). Although this would clean up the code, I'm postponing this change at this time because the preprocessing is transparent to msim_process() and can be changed at any time without changing the processing functions such as msim_incoming_im(). Another improvement would beto lookup the uid-to-username mapping from the buddy list, if possible, and only send a query if necessary. I wrote the code to (attempt) to do this, msim_uid2username_from_blist(), but it is not well-tested nor functional at the moment, so it will also temporarily postponed from use. But it is just an optimization (albiet a large one) -- receiving messages still works fine for now. The time will come.

So now that msim_preprocess_incoming() handles uid-to-username resolving on incoming messages in a transparent, layered way, I also found the need to translate username-to-uid on outgoing messages. The function to do this I called msim_postprocess_outgoing(). Called explicitly instead of msim_msg_send(), msim_postprocess_outgoing() will take a given MsimMessage, and add a new field in a given location with the uid of the given username. (Now you can see why I needed msim_msg_insert_before().) msim_postprocess_outgoing(), unlike its counterpart, will first try to lookup the username-to-uid mapping with purple_blist_node_get_int() (receiving the uid stored in a buddy on a buddy list) if possible, and fall back on sending a username-to-uid query, whose response will insert the uid field and actually send the message. msim_send_im() uses msim_postprocess_outgoing(), and now no longer requires msim_send_im_by_userid() and msim_send_im_by_userid_cb().

As you can see, MySpaceIM's feature of identifying users by numeric userid internally, but by usernames to the user, creates some complexity to overcome. But msim_preprocess_incoming() and msim_postprocess_outgoing() simplifies the complexity enormously. I'm just about done with architectural improvements at the moment, and now will probably concentrate more on using this brand new preprocessing/postprocessing infrastructure to add many useful features.

Tags: gsoc myspace msimprpl pidgin Pidgin
Source: jeff's blog
Login or register to post comments

Comments

looks like your coming along

Fri, 06/15/2007 - 12:26 — Anonymous

looks like your coming along way in the protocol plugin.

keep up the good work.

Login or register to post comments