



## Mollom client API 1.0

September 15, 2008

Mollom is a quality-assessment and spam-filtering service for user-submitted website content, including comments, contact form messages, and so on. Registered websites send submitted content to be checked by mollom.com. The Mollom service responds with a rating of either 'spam' or 'ham' (i.e. 'not spam') and a quality score. If Mollom is not fully confident of its spam rating, it will return the third possible answer: 'unsure'. In this case, a website can then ask the submitting user to respond to a CAPTCHA challenge supplied by the Mollom servers. Thanks to the 'unsure' reply and the CAPTCHAs, Mollom avoids legitimate content being incorrectly classified as spam. Combining text classification with an occasional CAPTCHA has two important benefits: (i) it effectively eliminates the need to moderate messages that Mollom decides to block, reducing the moderation workload, and (ii) because the CAPTCHAs are rare (currently only approximately 4% of human users ever see a CAPTCHA), it makes websites a lot more accessible.

A key Mollom feature is that all participating websites can send feedback to Mollom, explicitly marking comments as spam, profanity or low-quality. Mollom combines the information sent in by all participating websites and learns from it, preventing future abuse.

Mollom provides an Application Programming Interface (API) for the development of Mollom clients and for the use of its services. This is the documentation of Mollom's client API, but a reseller API is available separately. Please also read and agree to the additional information on the usage guidelines (<http://mollom.com/usage-guidelines>), the terms of service (<http://mollom.com/terms-of-service>) and the privacy policy (<http://mollom.com/web-service-privacy-policy>).

## 1 Mollom basics

Figure 1 illustrates how a client could interact with the Mollom service. If a message is received, it is sent to Mollom for checking, after which the client can decide to accept or block the content.

The Mollom API currently uses an XML-RPC interface (<http://xmlrpc.com>). XML-RPC is a simple remote procedure call protocol which uses XML to encode its calls and HTTP as its transport mechanism. XML-RPC libraries for various operating systems and programming languages are available at <http://www.xmlrpc.com/directory/1568/implementations>. Generic XML-RPC tutorials are available at <http://www.xmlrpc.com/directory/1568/tutorialspress>. The XML-RPC requests to the Mollom service should follow the HTTP/1.0 standard.

In order to use Mollom, a client needs to request a list of valid servers (see Section 9 for more information) to which it then can make Mollom service requests. Each website using Mollom also needs public and private keys, which can be obtained at <http://mollom.com>.

Each Mollom API request is an HTTP request that specifies one of the following remote procedure calls:

- `mollom.getServerList` – request a list of Mollom servers that can handle a site's calls.
- `mollom.checkContent` – asks Mollom whether the specified message is legitimate.

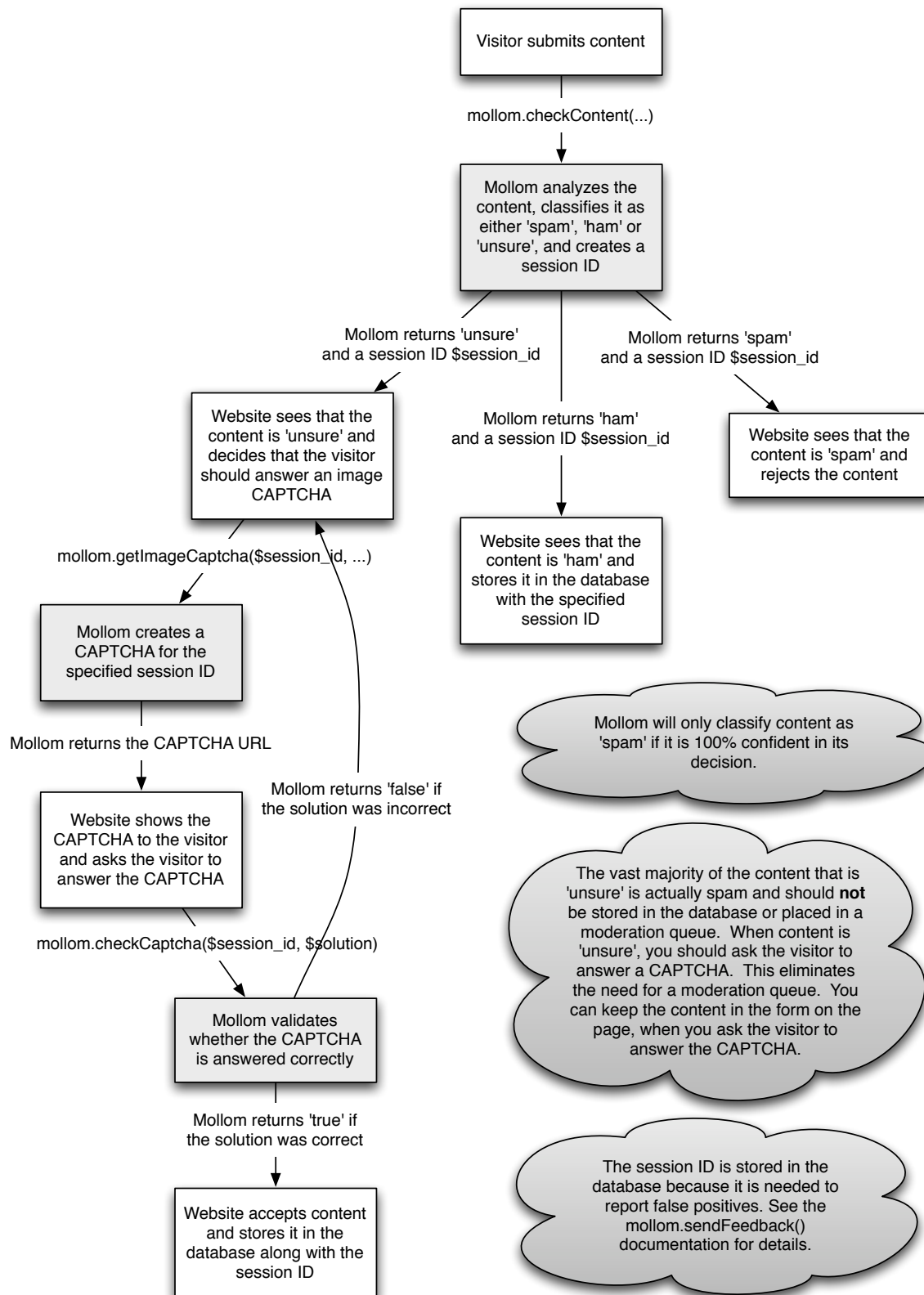


Figure 1: A detailed overview of how a client could use Mollom to check if a message is spam or not. The chart illustrates how the Drupal module for Mollom blocks comment spam.

- `mollom.sendFeedback` – tells Mollom that the specified message was spam or otherwise abusive.
- `mollom.getImageCaptcha` – requests Mollom to generate an image CAPTCHA.
- `mollom.getAudioCaptcha` – requests Mollom to generate an audio CAPTCHA.
- `mollom.checkCaptcha` – requests Mollom to verify the result of a CAPTCHA.
- `mollom.getStatistics` – retrieves usage statistics from Mollom.
- `mollom.verifyKey` – return a status value.

## 2 API versions

In order to accommodate multiple API versions in future, a version string needs to be added to the server URL. For example, a request made to `http://xmlrpc.mollom.com` using Mollom API version 1.0 needs to contact the following URL `http://xmlrpc.mollom.com/1.0`.

## 3 HTTP versions

The Mollom servers currently accept HTTP 1.0 connections only. HTTP 1.1 is not yet supported. If you are using CURL, make sure to set the HTTP version to 1.0 by setting `CURLOPT_HTTP_VERSION` to `CURL_HTTP_VERSION_1_0`.

## 4 Client-side load balancing and fail-over

Mollom provides clients (i.e. Drupal, Joomla!, Wordpress) with a list of available Mollom servers. Mollom provides different users with different server lists; paying customers can use more and/or different servers than free Mollom users.

First, this lets Mollom deal with a larger overall request load by distributing requests among multiple servers, aka *load balancing*, and to provide different users different service and performance guarantees. Second, Mollom can have have ‘hot spare’ servers on call to take over immediately if another machine becomes unavailable (*high availability / fail-over*). It also means that Mollom can offer dedicated servers with maximum privacy and quality of service to enterprise users, while still offering a fall-back strategy: other servers take over seamlessly until the dedicated machine is available again.

Clients need to request a server list before further communication with Mollom can be initiated. An initial server list can be obtained from `http://xmlrpc.mollom.com/1.0` using the `mollom.getServerList` call discussed in Section 9. Once a list of Mollom servers has been retrieved, it should be cached persistently, and clients can start making calls to the first server in the list. If this server fails, the client should fall back to the next server in the list and try again. The order of the servers in the list is important and should be respected; always start with the first server in the list. Figure 2 illustrates in more detail how a Mollom client should implement the client-side load balancing and fail-over functionality.

## 5 Authentication

All requests made to Mollom need to contain authentication information to establish the identify of the site making the request. To control the authenticity of a message and integrity of transmitted information, public and private Mollom keys are necessary. These keys can be obtained from `http://mollom.com`.

The mechanism that Mollom uses to provide this integrity is based on *message authentication codes* (MAC). Specifically, Mollom uses such a MAC mechanism based on a cryptographic hash function called *keyed-hash message authentication code* (HMAC), documented at `http://www.ietf.org/rfc/rfc2104.txt`.

Both the client and the server share a secret key that is used to create an authentication hash based on the combination of a time, a cryptographic nonce and the secret key. Both client and Mollom server hash the string concatenation of `time + ':' + nonce + ':' + secret key` with the secret key, and if their hashes match, the transmission is validated and authentic. The secret key is explicitly concatenated to the string that is hashed by the HMAC to prevent a possible known plaintext attack on the HMAC.

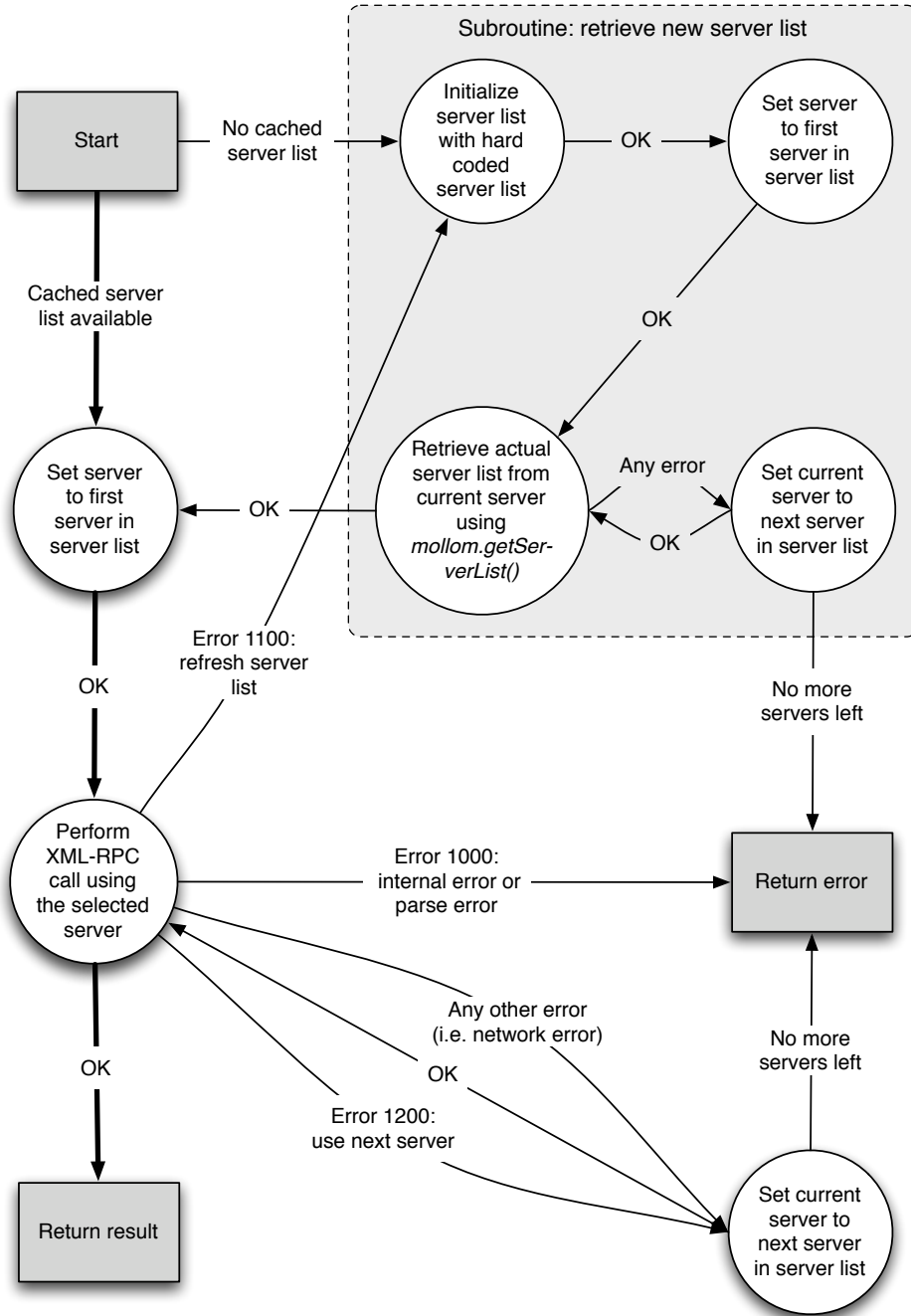


Figure 2: A detailed overview of how a client should execute XML-RPC calls, how a client should perform error handling and how a client should take advantage of Mollom’s load balancing and fail-over functionality.

Replay attacks are prevented by calculating a delta (the difference) between the site's pass time and the time on the Mollom server. Over subsequent calls, this delta is only allowed to change by one minute (but the delta is periodically reset to allow for clock changes). Additionally, a cryptographic nonce (a *one-use-only* randomly created number) is added. It is advisable to use a large random number or string as the nonce (recommended minimum: a 32-bit random integer). Both the time stamp and the nonce are protected by the HMAC hash.

The HMAC is only used for determining the authenticity of the Mollom user making the calls. The content itself is not protected. There are three reasons for not including the message content in the HMAC: (i) Many spam comments are very large, and busy sites receive thousands of them a day. Calculating the HMACs of all these messages' contents would become prohibitively computationally intensive. (ii) Due to varying handling methods for new lines and internationalized content on different platforms, this could lead to a significant source of coding errors. (iii) If spammers are out to inject information into Mollom, there are potentially easier ways to go about it. For this reason, the system has several security layers built in to prevent spammers compromising the Mollom database. Protecting actual message content in the HMAC would not improve this security.

To implement this authentication mechanism, **all** requests to Mollom must include at least these four parameters:

**public\_key** The individual public access key used to identify a site.

**time** A valid `dateTime` (<http://www.w3.org/TR/xmlschema-2/#dateTime> or <http://www.w3.org/TR/OTE-datETIME>). The format usually takes the following form `1978-11-19T04:20:10.000+02000` (`yyyy-MM-dd'T'HH:mm:ss.SSSZ`) where text between single quotes should not be interpreted.

**hash** The HMAC-SHA1 digest (<http://www.ietf.org/rfc/rfc2104.txt>) of the string concatenation of the specified `time` + `':'` + `nonce` + `':'` + `secret key`. This is used to validate a site's authenticity.

**nonce** The randomly created nonce.

To see PHP authentication code, please consult the Mollom module for Drupal, available for download on the Mollom website. Open the file `mollom.module` and search for the functions `mollom()` and `mollom_authentication()`. Examples in other languages are available as well.

If additional security is required, we can consider to setup a SSL certificate and to handle requests using a secure transport layer. This would guarantee Mollom users that they are communicating with legitimate Mollom servers and that content traffic is authentic.

## 6 Error handling

As shown in Figure 2, Mollom can return several XML-RPC error codes. If a parse error or an internal problem occurs, error code 1000 will be returned. Mollom uses a client-side load-balancing scheme (see Section 4), which requires some additional error codes. The error code 1100 indicates that a refresh of the server list (see Section 4 and Section 9) is required. The error code 1200 signifies that the current server is too busy, and the client's request should be resent to the next server in the current server list.

Note that if the Mollom server is unavailable due to network problems or server downtime, the XML-RPC client library will probably return network errors.

See Section 18.5 for the XML-RPC error message format.

## 7 Mollom sessions

A session ID is a unique token that the Mollom server assigns to a session and specific operation that a site visitor tries to perform. The first in a series of requests to the Mollom server requires no session ID. Mollom generates and assigns a new session ID and returns it to the client. Any further operations performed on this message must include its unique session ID. For example, if a user edits a message previously checked by Mollom, the client needs to send the new message data to Mollom together with its previously assigned session ID.

## 8 Author IP

The content author's IP address (not the hostname) needs to be included with many of Mollom's API calls. This parameter is optional, but it can drastically improve Mollom's results. When someone moderates or edits a piece of content, the client should send Mollom the original poster's IP address, not the moderator's. If the the original author's IP address isn't known, the author IP address should be left unspecified.

Sending Mollom the right IP address is key, and often non-trivial. Care needs to be taken to do it right. In most situations, the visitor's IP is available in the `REMOTE_ADDR` HTTP variable. However, when running behind a reverse proxy (such as Squid), the `REMOTE_ADDR` HTTP variable will contain the reverse proxy's IP address and the visitor's IP address will be in the `HTTP_X_FORWARDED_FOR` HTTP header instead. It is important to send the visitor's IP address and not that of the reverse proxy. Similarly, when running on a cluster setup, you might have to use the `HTTP_X_CLUSTER_CLIENT_IP` HTTP header to get the right IP address.

Note that both `HTTP_X_FORWARDED_FOR` and `HTTP_X_CLUSTER_CLIENT_IP` can be spoofed by spammers; you should not send these headers unless they are guaranteed to come from a reverse proxy. To see example PHP code, please consult the `ip_address()` function in `includes/bootstrap.inc` of Drupal 6 and beyond. You'll see that Drupal does not use the `HTTP_X_FORWARDED_FOR` or `HTTP_X_CLUSTER_CLIENT_IP` HTTP headers unless a Drupal site administrator has explicitly enabled Drupal's `reverse_proxy` configuration setting. We believe that an explicit configuration setting is the best way to prevent spoofing attacks.

## 9 mollom.getServerList

mollom.getServerList			
Required	Name	Type	Description
required	public_key	string	Site public key
required	time	string	Site server time in this format:yyy-MM-dd'T'HH:mm:ss.SSSZ
required	hash	string	HMAC-SHA1 digest
required	nonce	string	One time nonce
		array of strings	Array of servers that can be used

The remote procedure call to obtain the list of valid Mollom servers – needed to initiate traffic – is `mollom.getServerList`. The order of the servers in the list is important. Always use the first server in the list, unless that server is unavailable. A server is unavailable on the return of a network error or an XML-RPC error *other than* 1000, 1100 or 1200 (see Section 6 and Figure 2).

It would be a waste of overhead to call `mollom.getServerList` before every Mollom API call or HTTP request. Clients should include code to cache the server list for an extended time using either a database- or file-cache. Clients can be designed to refresh the server list once every few days or weeks, but this is not necessary. Mollom will return the 1100 error code if it needs a client to update its cached server list.

Clients should also request a new server list if they don't have one or if all of the listed servers fail. A new server list can be retrieved from any Mollom server, but when no servers are currently assigned or when they have all failed, the `mollom.getServerList` call should be directed to either `http://xmlrpc1.mollom.com`, `http://xmlrpc2.mollom.com` or `http://xmlrpc3.mollom.com`. Figure 2 illustrates in more detail how this mechanism needs to be implemented.

In other words, before every remote procedure call, the client should execute something along the lines of the following code, taken from the Drupal 6 Mollom module available at `http://mollom.com/download`:

```
// Retrieve the list of Mollom servers from the database:
$servers = variable_get('mollom_servers', NULL);

if ($servers == NULL) {
  // Initialize server list:
  $servers = _mollom_retrieve_server_list();

  // Store the list of servers in the database:
```

```
variable_set('mollom_servers', $servers);
}
```

If the server list is not yet initialized, the client needs to retrieve a server list from mollom.com as shown in the subroutine in Figure 2:

```
function _mollom_retrieve_server_list() {
  // Start from a hard coded list of servers:
  $servers = array('http://xmlrpc1.mollom.com', 'http://xmlrpc2.mollom.com', 'http://xmlrpc3.mollom.com');

  // Retrieve the actual server list from mollom.com:
  foreach ($servers as $server) {
    $result = xmlrpc($server . '/' . MOLLOM_API_VERSION, 'mollom.getServerList', _mollom_authentication());
    if (!xmlrpc_errno()) {
      return $result;
    }
  }

  return array();
}
```

Once the client has a valid list of Mollom servers, it can execute the actual remote procedure call as follows:

```
if (is_array($servers)) {
  // Send the request to the first server, if that fails, try the other servers in the list:
  foreach ($servers as $server) {
    $result = xmlrpc($server . '/1.0', $method, $data + _mollom_authentication());

    if ($errno = xmlrpc_errno()) {
      if ($errno == MOLLOM_REFRESH) {
        // Retrieve new server list from current server:
        $servers = _mollom_retrieve_server_list();

        // Store the updated list of servers in the database:
        variable_set('mollom_servers', $servers);
      }
      else if ($errno == MOLLOM_ERROR) {
        return $result;
      }
      else if ($errno == MOLLOM_REDIRECT) {
        // Do nothing, we select the next server automatically.
      }
    }
    else {
      return $result;
    }
  }
}

// If everything fails, we reset the server list to force loading of a new list:
variable_set('mollom_servers', array());
```

## 10 mollom.checkContent

mollom.checkContent			
Required	Name	Type	Description
required	public_key	string	Site public key
required	time	string	Site server time in this format:yyyy-MM-dd'T'HH:mm:ss-SSSZ
required	hash	string	HMAC-SHA1 digest
required	nonce	string	One time nonce
optional	session_id	string	Current session ID
optional	post_title	string	Title of submitted post
optional	post_body	string	Body of submitted post
optional	author_name	string	Submitting user's name or nick
optional	author_url	string	Submitting user's URL
optional	author_mail	string	Submitting user's email address
optional	author_openid	string	Submitting user's openID
optional	author_ip	string	Submitting user's current IP
optional	author_id	string	Submitting user's unique site ID
	spam	integer	Returns 1 if ham, 2 if spam and 3 if unsure
	quality	double	An assessment of the content's quality, between 0 and 1
	session_id	string	Session ID

The `mollom.checkContent` call is probably the most frequently used Mollom call. It can be used to check if a comment is spam or not, and to get an assessment of its quality. The `mollom.checkContent` call will return 'ham', 'spam' or 'unsure' (encoded as 1, 2 and 3, respectively) together with a session ID and a quality assessment. If Mollom returns 'ham' or 'spam', the content can be safely accepted or rejected, as the case may be. But if Mollom returns 'unsure', an additional check is needed to decide if the content can be accepted or not. Mollom provides CAPTCHA challenges for this check, but other mechanisms could be used. Mollom is designed so that only a small fraction of human-submitted content will be flagged as unsure.

Note that if Mollom returns 'spam', no CAPTCHA should be shown to the user. Mollom will only return 'spam' if it is 100% sure that the content is spam. It is essential that these attempts are blocked without presenting any CAPTCHA. This allows Mollom to block both spambots trying to hack the CAPTCHAs and human users sending spam.

The quality score returned by `mollom.checkContent` is a real value between 0 and 1, where 0 denotes very bad quality, and 1 very high quality. Mollom only returns a score, clients must define for themselves the quality level cutoff between content acceptance and rejection. The quality scores could also be used to present the content sorted in a way that makes moderation easier.

Remarks:

- Apart from the authentication fields, which are compulsory, all other fields are optional. This means that they can either be left out altogether or be empty strings. However, the more information Mollom receives, the more accurate its classification will be.
- If multiple OpenIDs are given for a user, they can all be passed into the OpenID field by separating them with white spaces (spaces, tabs or new lines).
- If a site has content types that do not map well onto the specified fields (for example, a 'survey' content type), content type fields or data can be concatenated and passed into the `post_body` field.
- A unique user ID (user name or numeric ID) can be passed to Mollom. If no user ID is known (for an anonymous user, for example) no value should be passed.

## 11 mollom.sendFeedback

mollom.sendFeedback			
Required	Name	Type	Description
required	public_key	string	Site public key
required	time	string	Site server time in this format:yyyy-MM-dd'T'HH:mm:ss-SSSZ
required	hash	string	HMAC-SHA1 digest
required	nonce	string	One time nonce
required	session_id	string	Session ID associated with the feedback
required	feedback	string	Feedback: "spam", "profanity", "low-quality" or "unwanted"
		boolean	Always returns true

The `mollom.sendFeedback` call can be used to provide feedback to Mollom. It tells Mollom that the specified message was spam or otherwise inappropriate for the reporting website.

This mechanism allows Mollom to learn from its mistakes. For example, if a spam message was erroneously classified as ham, Mollom will use that information to fine-tune its spam detection mechanisms. The more feedback provided to Mollom, the more effective it becomes. If possible, this feedback mechanism should be built into all Mollom applications.

Mollom will not blindly trust `mollom.sendFeedback` requests. It has a built-in reputation-management system that separates valid `mollom.sendFeedback` requests from invalid `mollom.sendFeedback` requests. Developers should be careful when testing this functionality. It might negatively affect the reputation associated with a particular set of credentials. See Section 17 for more information about testing your Mollom client.

The remote procedure call `mollom.sendFeedback` takes the parameters required for authentication, the session ID of the message that is being reported on, plus the actual `feedback` being provided.

There are four possible string values for `feedback`:

Value	Description
<code>spam</code>	Spam or unsolicited advertising
<code>profanity</code>	Obscene, violent or profane content
<code>low-quality</code>	Low-quality content or writing
<code>unwanted</code>	Unwanted, flaming, trolling or off-topic content

These options not only help Mollom to learn spam patterns, but also get metrics on obscene and poor-quality content. This is used for improving the assessment of the `quality` score returned by `mollom.checkContent`.

The method `mollom.sendFeedback` always returns the Boolean value 'true' if the function call succeeded. If not, an XML-RPC error is returned.

## 12 mollom.getImageCaptcha

mollom.getImageCaptcha			
Required	Name	Type	Description
required	public_key	string	Site public key
required	time	string	Site server time in this format:yyyy-MM-dd'T'HH:mm:ss-SSSZ
required	hash	string	HMAC-SHA1 digest
required	nonce	string	One time nonce
optional	session_id	string	Current session ID
optional	author_ip	string	Submitting user's current IP address
	session_id	string	Session ID associated with this CAPTCHA
	url	string	URL of the CAPTCHA

An *image CAPTCHA* is an image that can be used to tell whether the poster of a comment or message is a human or a computer. To do so, the image displays a distorted text message that is hard to read for computer programs but still readable for humans. CAPTCHAs aim to stop spambots or automated

scripts from posting comments or messages. The `mollom.getImageCaptcha` call instructs Mollom to generate an image CAPTCHA on the server-side.

The CAPTCHA APIs can be used in combination with `mollom.checkContent` or as a stand-alone API – i.e. it can be used to protect a registration form without having to use Mollom’s content analysis features.

If there is an active session ID for this operation because `mollom.checkContent` was already called or CAPTCHAs have already been called for a visitor’s previous operations, it should be specified as the `session_id` parameter. A session ID is a unique token that the Mollom server assigns to a session and specific operation that a site visitor tries to perform. Note that the Mollom sessions are in no way related to HTTP sessions. The first in a series of requests to the Mollom server requires no session ID. Mollom generates and assigns a new session ID and returns it to the client. Any further operations performed on this message must include its unique session ID. For example, if a user edits a message previously checked by Mollom, the client needs to send the new message data to Mollom together with its previously assigned session ID.

For example, if a website needs to request three CAPTCHAs in a row for a particular visitor, it is important that the website keeps track of the `session_id` assigned by the Mollom server. For each of the subsequent requests, that session ID must be specified as part of the `mollom.getImageCaptcha` call. This is important because it helps the Mollom server identify visitors trying to break a CAPTCHA through brute force attempts.

Furthermore, session IDs can and should be reused across different call types. For example, if `mollom.checkContent` returns `unsure` and a visitor is asked to solve a CAPTCHA, the session ID returned by `mollom.checkContent` should be included in the call to `mollom.getImageCaptcha`. Note that Mollom cannot guarantee that the same `session_id` passed into Mollom will be returned.

Mollom will respond to a successful request with an XML-RPC response message that containing a `session_id` and the URL of the CAPTCHA.

Generating CAPTCHAs on a central server helps make them more effective. As soon as Mollom’s servers notice that spammers are able to break the CAPTCHAs with automated scripts or programs, Mollom can alter the way in which these messages are distorted and presented.

Mollom removes image CAPTCHAs from its servers that are more than 30 minutes old. Also note that every time a site visitor reloads a CAPTCHA image from Mollom’s servers on any given URL, a different CAPTCHA will be presented. Only the text of the most recently loaded CAPTCHA can be used to solve the CAPTCHA. Developers please note: CAPTCHAs or CAPTCHA URLs should never be cached or stored locally.

When you display the image CAPTCHA, please do not assume that the CAPTCHAs will always have the same size because the Mollom servers can return variable sized CAPTCHAs. For example, never hard code the width and height attributes of the HTML image tag.

## 13 mollom.getAudioCaptcha

mollom.getAudioCaptcha			
Required	Name	Type	Description
required	<code>public_key</code>	string	Site public key
required	<code>time</code>	string	Site server time in this format:yyyy-MM-dd'T'HH:mm:ss-SSSZ
required	<code>hash</code>	string	HMAC-SHA1 digest
required	<code>nonce</code>	string	One time nonce
optional	<code>session_id</code>	string	Current session ID
optional	<code>author_ip</code>	string	Submitting user's current IP address
	<code>session_id</code>	string	Session ID associated with this CAPTCHA
	<code>url</code>	string	URL of the CAPTCHA

An *audio* CAPTCHA is an mp3 file in which a series of digits and letters is spoken by a distorted human voice. Just like an image CAPTCHA, it can be used to tell humans and computers apart. Audio CAPTCHAs are accessible for visually impaired users.

The interface for requesting audio CAPTCHAs is identical to the `mollom.getImageCaptcha` call.

## 14 mollom.checkCaptcha

mollom.checkCaptcha			
Required	Name	Type	Description
required	public_key	string	Site public key
required	time	string	Site server time in this format:yyyy-MM-dd'T'HH:mm:ss-.SSSZ
required	hash	string	HMAC-SHA1 digest
required	nonce	string	One time nonce
required	session_id	string	Session ID associated with the CAPTCHA
required	solution	string	Submitted CAPTCHA solution
		boolean	True if correct, false if incorrect

To validate a visitor's CAPTCHA answer, it has to be sent with the appropriate session ID to Mollom to be checked. Mollom will return a Boolean where `true` means that the CAPTCHA was filled out correctly, and `false` means that the answer was incorrect.

When the result is incorrect, a new CAPTCHA can be requested from Mollom by calling `mollom.getImageCaptcha` or `mollom.getAudioCaptcha` again. When making a request following a failed response, the active `session_id` must be specified in the call. This enables Mollom to do proper accounting and identify spambots trying to solve CAPTCHAs by brute force.

## 15 mollom.getStatistics

mollom.getStatistics			
Required	Name	Type	Description
required	public_key	string	Site public key
required	time	string	Site server time in this format:yyyy-MM-dd'T'HH:mm:ss-.SSSZ
required	hash	string	HMAC-SHA1 digest
required	nonce	string	One time nonce
required	type	string	Type of statistics requested
		integer	Requested statistics value

The `mollom.getStatistics` call retrieves usage statistics from Mollom. In addition to the authentication part, an extra `type` field needs to be passed in. There are several possible string values for `type`:

Value	Description
<code>total_days</code>	Number of days Mollom has been used.
<code>total_accepted</code>	Total accepted posts.
<code>total_rejected</code>	Total rejected spam posts.
<code>yesterday_accepted</code>	Number of posts accepted yesterday.
<code>yesterday_rejected</code>	Number of spam posts blocked yesterday.
<code>today_accepted</code>	Number of posts accepted today.
<code>today_rejected</code>	Number of spam posts rejected today.

The call returns an integer denoting the desired statistics type.

## 16 mollom.verifyKey

mollom.verifyKey			
Required	Name	Type	Description
required	public_key	string	Site public key
required	time	string	Site server time in this format:yyyy-MM-dd'T'HH:mm:ss-.SSSZ
required	hash	string	HMAC-SHA1 digest
required	nonce	string	One time nonce
		boolean	Always returns true

The `mollom.verifyKey` call returns information about the status of a key. It requires only the authentication sections in the call. Mollom will return a Boolean value 'true' if the key is valid. If there is a problem with the key, and XML-RPC error with fault code 1000 will be returned with a detailed error message.

## 17 Testing

To test a Mollom plug-in during development, Mollom allows a key to be put into 'developer mode'. This can be done on the Mollom website by creating a key associated with a test site. After the key is created, follow the 'edit site' link for the site you just created, and enable the 'developer mode' checkbox. By doing this, API calls using this key will be treated as test calls by Mollom. If an already existing site is put into developer mode, it can take as long as 60 minutes for the changes to become effective.

When in 'developer mode', the `checkContent` method can be forced to return certain values. A specific string passed in the `post_body` field defines what is returned. If the string `spam` is passed, the call will return 'spam' with quality 0.0; if `unsure` is passed, it returns 'unsure' with quality 0.5; and if `ham` is passed, it will return 'ham' with quality 1. Mollom will also return valid session IDs that can be used for testing.

The `checkCaptcha` method can also be forced to return a specific response by setting the `solution` field. If the field is set to the string `correct`, the call will return `true`; when it is set to `incorrect`, `false` will be returned.

The calls for requesting a CAPTCHA `getImageCaptcha` and `getAudioCaptcha` will return valid CAPTCHA sessions and URLs. The correct/incorrect responses can be faked using the `checkCaptcha` test mode described above.

In developer mode, the `sendFeedback` method will test for correct use, but will simply return 'true' with no further effect.

The `verifyKey` call can be tested using your own created developer key (this will return true), a specifically created disabled key (which has a public key value of 'disabled-key' and private key value of 'disabled-key'), and a specific unknown key (which has a public key value of 'unknown-key' and a private key value of 'unknown-key'). The last two cases will return exceptions.

## 18 XML-RPC basics

What follows is a brief overview of the important XML-RPC structures needed when calling Mollom. First, the request and response structure is given, next, the construction of a single value, a struct and an error is shown.

### 18.1 Request

A request to Mollom has the following form, where the exact parameters are given as a structure.

```
<?xml version="1.0"?>
<methodCall>
  <methodName>mollom.getServerList</methodName>

  -request parameters-
</methodCall>
```

## 18.2 Response

A response from Mollom is structured as follows, where the parameters can be a struct, a single value or a fault.

```
<?xml version="1.0"?>
<methodResponse>

-response parameters-

</methodResponse>
```

## 18.3 Single value

A single value is passed or received from Mollom in this form.

```
<params>
  <param>
    <value>
      <boolean>1</boolean>
    </value>
  </param>
</params>
```

Note that Mollom does not currently support CDATA sections in the XML.

## 18.4 Structure

A set of parameters is passed or received via a struct. This is structured as follows.

```
<params>
  <param>
    <value>
      <struct>
        <member>
          <name>field1</name>
          <value><string>some string</string></value>
        </member>
        <member>
          <name>field2</name>
          <value><integer>42</integer></value>
        </member>
      </struct>
    </value>
  </param>
</params>
```

## 18.5 Error code

If an XML-RPC error is received, this is encoded in the following specific form.

```
<fault>
  <value>
    <struct>
      <member>
        <name>faultCode</name>
        <value><int>1000</int></value>
      </member>
      <member>
        <name>faultString</name>
        <value><string>Invalid public key.</string></value></member>
    </struct>
  </value>
</fault>
```