Stack Exchange API Version 2.0 Spec
This is an “air-quotes spec” for the next version of the Stack Exchange API.

This is speaking in much broader strokes than formal documentation, describing intent and philosophy more than interface. It is not intended to guide application development, but instead to solicit feedback about direction of the API. Think “RFC, but softer”.

Actual development of the API will follow the same “deploy, iterate, use, repeat” cycle as all other Stack Exchange features have; with a proper beta period as with version 1.0. Developers should wait for proper documentation of new methods before attempting development against the 2.0 API.

Breaking Changes
These changes will require changes to existing code written against the v1.1 API.

Additions to existing methods are included in this category, as depending on how the returns were deserialized they may in fact be breaking.

All String Fields Will Be HTML “Safe”
With 1.x some fields require HTML encoding (like question titles) while others are safe to inline (like question bodies).

With 2.0, all fields will be safe to inline by default. This will require any existing encoding calls to be removed.

The goal here is to have a consistent string safety, and given that we think it’s better to default to safe rather than unsafe. We believe it is reasonable to expect all API consumers to have some way of dealing with HTML, whether that is “HTML aware” controls, HTML strippers/decoders, or what have you.

There has been some debate as to whether this approach is correct. The counter argument is that the API should be returning data, not concerning itself with presentation.

The penalty for failure is worth considering. In the “default to safe” case, a failure results in over encoding. It’s ugly, and will generate bug reports, but that’s all. In the “default to unsafe” case, a failure results in a script injection attack or worse.

To accommodate those cases where unsafe data is desired, it will be possible to create filters (which are discussed under New Features) which return unsafe data. This will effectively revert the API back to v1.x encoding rules.

There will not, however, be any default filters that are unsafe. The extra “I know what I’m doing” friction a developer must endure to get to unsafe data is by design. It may be illustrative to compare this to C#’s `unsafe` keyword.
**Meta Wrapper Object Changes**

Currently, a return is keyed by the type.

/questions returns are found under the “questions” key in a wrapper object (that has count, page, pagesize, etc.).

Example:

```json
{
  ...,  
  "questions": [
    {  
      ...
    },
    ...
  ]
}
```

With 2.0 we will be breaking this data out into two distinct fields: “type” and “items”.

Like:

```json
{
  ...,  
  "type": "question",
  "items": [
    {
      ...
    },
    ...
  ]
}
```

This will make it easier to write de-serialization code, and make the relationship between distinct methods that return data of the same type clearer.

**Dropping All Sorts Based On Views**

These are very rarely used sorts (bordering on never), and have become increasingly expensive to support (as view data has been moved “further away” in the database). View counts are also very “noisy” and thus of quite limited utility.
**Shallow User’s Without IDs**
Currently, any place we return a user outside of the `/users/*` methods returns a common “shallow user”.

Example on /questions:

```
"owner": {
   "user_id": 147915,
   "user_type": "registered",
   "display_name": "redconservatory",
   "reputation": 815,
   "email_hash": "be528e6ffa1cab6c5e0f7ab9157e6f2d"
}
```

This causes a problem whenever we have to deal with users who do not exist (which is rare, but happens), since v1.x requires the “full user” to be returned and we don’t have one anymore.

v2.0 will allow for just the `display_name` field to be returned, with the `user_type` of “does_not_exist”.

If the above user were deleted, we would return the following:

```
"owner": {
   "user_type": "does_not_exist",
   "display_name": "redconservatory"
}
```

This will require code that assumes a user_id is present to change to first check user_type, which is guaranteed to always be present. Be aware that display_name is **not** guaranteed to be present.

In an earlier draft, we called these users “deleted”. This has been changed, because there are cases where a user never existed (mostly due to post migrations between sites); “deleted” implied the wrong relationship. It is, unfortunately, not possible to distinguish between users who once existed and no longer do so and those who were never created in the first place.

**Changes To Association Data**
We’re phasing out association_id in favor of a numerical equivalent called account_id. This field will be added to all users returned, and be the key of interest to `/users/{id}/associated` in version 2.0.

The motivation for this change is many fold.

First, upcoming changes to account association to remove the explicit “associate” action that has become mostly vestigial (and continues to confuse users). Second, GUIDs are ugly and confusing to users (as seen on stackexchange.com user page links). Finally, recent changes to credential management internally have both introduced the notion of an account id and made it so that in the future using GUIDs for identification will impose some small performance penalty.

Thus, we feel it’s important to move to account_ids sooner rather than later, given the expected life time of version 2.0.
Be aware that while most users will have an account_id, not all will for historical reasons. It is reasonable to assume that all users who authenticate to an application will have an account_id, however.

/users/{id}/associated will also resume returning reputation, as it has become much quicker to gather that data in the interval between versions 1.1 and 2.0.

For apps with existing databases of association_ids, a service outside of the proper API will be created to map those to account_ids. This service will not be bound by the lifetime of API, probably living for only a month or two.

Be aware that association_ids have not been historically as stable as account_ids, so is there is no guarantee that a cached association_id will map to an account_id. This mapping service will be a best effort affair accordingly.

**Quotas Will No Longer Be Returned In Headers**
Right now, we report remaining requests per day, and maximum requests per day in the headers of an API response. Historically, this is because Twitter does it that way.

There are some problems with this:

1. Discoverability is awful, this is the only thing we stash in headers
2. Other v2.0 changes w.r.t. filtering returned data would need to be special cased to deal with headers
3. We always return these, and 99.9% of the time nobody checks them; terrible waste of bandwidth.

Instead they will be returned as fields on the wrapping object, a la `total` and `type`, named `quota_current` and `quota_max`.

Code that checks for the existing headers will need to be changed to check for these fields.

**Badges Method Changes**
Badges kind of grew out of control, adding basic sorting and paging over the `/badges/*` routes. Sort options of name, type [named, tag_based], and rank will be added.

Changing `tag_based: bool` to `badge_type: (named|tag_based)`, as sorting on a bool is sort of... weird.

All existing badge code will need to be changed accordingly.

**Parameters Used For Optionally Returned Fields Will Be Dropped**
`body`, `answers`, and `comments` will be dropped.

This functionality will be overtaken by filters, code using them will need to be changed.
URL Hooks will be dropped
Returned values of the form `*_url` were an... idea. They help in the very short term, because new developers see them as pointers to methods they may not have known existed.

In the longer term, they turned out to just be a waste of bandwidth so they’re being dropped. Better documentation (which really started with 1.1) takes over any utility they had.

New User Fields
Adding the fields
- `se_employee` – to distinguish proper moderators and users from employees
- `rep_change_day/week/month` – interesting data we have, might as well expose it
- `last_modified` – when the user’s profile was last changed
- `account_id` – see “Changes To Association Data” for more discussion

Returning Unregistered Users
Currently `/users` will only return registered users. This was a bug that snuck into v1.0, and we’ve been stuck with it ever since.

It’s going to be fixed in 2.0.

Any code that makes assumptions that `user_type` is equals to “registered” when returned from `/users` will need to be changed.

Adding Link Fields
Something that is oddly difficult with the API is generating a proper url to the same resource on a Stack Exchange site. Attribute rule changes since 1.0 was released make this oversight even more egregious.

For instance, my user link is “http://stackoverflow.com/users/80572/kevin-montrose”; one of my posts is “http://stackoverflow.com/questions/5724064/nearly-all-openid-relying-parties-reject-assertions-from-my-dotnetopenauth-backed”. To render either of those links you need to know internal link rendering rules: which characters to strip, how many characters to leave in, what to do with spaces and entities, and so on. Answers are even worse.

To address this the question, answer, user, and comment types will gain `link` fields which will be populated with a link to the same resource on the site.

Beta And Launch Dates Added To /Sites
`/sites` will return private beta, public beta, and launch dates for each site. Sites without formal betas (like the Trilogy, or SE 2.0s that skipped private beta) will not return those fields.

What we’ve found, both internally and externally, is that there are lots of different valid definitions of a “creation date” for a site depending on what use is intended. We feel it’s better to be explicit about exactly what event in a site’s life-cycle a date corresponds to rather than advice people “discovery creation date” via some semi-hacky method (like querying for the community user’s creation date).
Reworking Errors
Currently there are a number of error codes which are just minor variations on “bad parameter”. We’ll collapse those all into one.

Furthermore, using multiple HTTP status codes makes dealing with errors from browsers pointlessly complicated. We’re just going to 400 everything.

This will require any error handling code to be changed. In theory, most code will have a single point where this can be checked.

Renaming /stats To /info
The statistics route has come to carry more general information about a site than the statistics names would suggest. Things like styles, related sites, and so on. The name /info better suits what it has become. A simple rename should be sufficient for existing calls to /stats.

Changing The JSONP Callback Parameter To “callback”
Versions 1.x of the API used `jsonp` for JSONP callback function names. This choice was purely accidental, it just so happened that we had existing code for handling JSONP responses that expected that parameter.

This does make it a bit more difficult to use JQuery against our API than is really necessary. Version 2.0 will begin expecting the JSONP callback function name in `callback`.

For backwards compatibility purposes, `jsonp` will continue to be honored. We feel there’s no reason to force this change on old code, it’s really just to ease future development. We do consider `callback` the preferred parameter name going forward, and may drop `jsonp` in a future release.
New Features
These changes will require entirely new code is written to take advantage of them.

Authentication
We will be implementing OAuth 2.0 for user authentication.

The draft specification can be found at:

We’re taking Facebook’s implementation as a model for ours; given its widespread usage it should make acclimating easier for developers. You can read more about their flow at:
https://developers.facebook.com/docs/authentication/

We will include two scopes in version 2.0, `read_inbox` and `no_expiry`. By default, access tokens will expire after some period to be determined; most likely 1 day. Default permissions simply allow a user to be identified, which should be sufficient for most purposes “pre-write”.

Authentication will be at an account level, a user will grant access to all of their user accounts across the Stack Exchange network with a single authorization flow. In other words, if an app user with accounts on Stack Overflow and Math Stack Exchange authorizes with default permissions that single access code will allow their users on each site to be discovered. Per-site authorizations introduce too much end-user frustration, although they may still be introduced as part of write permissions in a subsequent release.

UI for authorization will reside under stackexchange.com (in the redirect case), and a popup will be available for the client-side case. The exact method of obtaining a popup remains to be determined. Although the user will interact with permissions, dialogs, etc. hosted under stackexchange.com; the actual starting URI may be a specific Stack Exchange site that the user indicates having an account on. This depends on the ultimate implementation, as well as a measure of users’ “understanding they’re on a network”.

Desktop application authorization will conform to the flow for Desktop Apps Facebook provides. In short, a well-known return URI will be authorized for all applications and a desktop app can use redirection of an embedded web browser control to that URI as completion of authorization (and read an appended access_token parameter at that time). Authorizing a desktop application against a web site has always been cumbersome, but we think this flow isn’t needlessly complicated or confusing to end users.

“Me” Methods
Hand in hand with authentication is a way to refer to the currently authenticated user.

Every method of the form `/users/{ids}` will have an equivalent starting with `/me` that conceptually runs the same method where `{ids}` is set to the id of the authenticated user. `/users/{ids}/comments` becomes `/me/comments` for example.
Naturally, these methods will require an auth token.

**Filters**

One of the pain points with the API is the difficulty in tuning performance, from a client’s perspective.

A related problem is the amount of bandwidth wasted: a client has to pull down basically everything about a user to get just their reputation, for example.

To address this, we’re adding “filters” to let clients specify which fields are cared about. Any fields not requested will not be returned (saving bandwidth), and where possible not queried for (generally at the JOIN level).

Creation of a filter will involve passing a set of field names to `/filter/create` in `include` and `exclude` (and optionally, a `base_filter`), which will return an opaque value. This value can be then passed as `filter` to any api method.

`/filter/create` will also accept a `safe` field, which defaults to true. If set to false, string returned to requests made with the created filter are **not guaranteed to be safe to inline**. See the discussion above under “All String Fields Will Be HTML Safe” for why this defaults to safe. It is recommended that all applications try not to make unsafe requests, though we acknowledge that it may be necessary under some limited conditions.

Field names will take the form of `type.field`. For example: question.answers, question.title, question.owner.reputation, and so on. Top level (wrapper object) fields will be referred to with a leading `.`, like `.total`. Excluding an entire type (like `question`) will be possible.

A typical application is envisioned as having only one or two filters, conceptually cutting down the API into just the data they care about. More complicated applications may build filters on the fly, and we support that, but it should be very uncommon.

Filters will not be tied to an app or user context, and will be immutable.

By default, filters will exclude `.total` (doubling the speed of a query in common cases), `.type`, question.body, answer.body, user.about_me, question.comments, answer.comments, and quota fields.

In the interest of ease of development, all routes will also take the `include` and `exclude` fields but only when called without an app key. This should make it easy to dummy up filters while developing an app, but prevent developers from completely missing the point and making absurdly long requests in live apps.

Some default filters will be provided: `all` and `total` which will return everything, and just the total field respectively. Additionally, default filters that represent all combinations of `body`, `answers`, and `comments` as exposed by v1.x will be provided to ease transition from those parameters.
Inbox Access
A new route, /users/{ids}/inbox, will return items in inboxes. It will be just enough information for display purposes, a la stackexchange.com’s inbox tab.

This method will require an auth code with `read_inbox` permissions granted.

New “last modified” User Sort
Currently is something of a pain to get a diff of users’ profiles; this will solve that.

The actual sort parameter value will be “modified”, joining reputation, creation, and name. Reputation will remain the default sort.

Suggested Edits
Suggested edits were introduced immediately before version 1.1 was released, and were in too much flux to be included at that time. This is no longer the case, and they will be added in 2.0.

New methods of `/suggested-edits` and `/suggested-edits/{ids}` with sorts of approval, rejection, & creation. A `/posts/{ids}/suggested-edits` method with the same options will also be added.

Returned data of the edit text, related post, rejection, approval, and creation dates, and the proposing user if applicable.

Elected Moderators
Adding a `/users/moderators/elected` method to distinguish between “is a moderator (potentially SE employee)” and “was elected a moderator”.

Per User/IP Pair Quotas
Our current throttling is strictly IP based. This causes... issues, especially if using something like Google App Engine.

With authentication support, we have another option. A user will get a request quota per IP address authenticated a day. Requests per second will still be IP based, as there’s not really any other option there.

There will be a max of 10 authorized application/ip pairs per user per day, after which quota will begin being shared again. This is necessary to prevent abuse.

Applications serving multiple users, but not requiring authentication, on shared IPs (like Google App Engine) should authorize as the application author with `no_expiry` permissions. As application authors have historically been Stack Overflow users in good standing, we will raise per-user quotas as needed for those applications. This will lock up one “slot” for the application author, but we believe this is acceptable.
**Event Stream**

We will be adding an `/events` method, which takes a parameter of `since` which must be within the last 15 minutes.

This method will return objects describing questions, answers, edits, comments, and user creations that have occurred after the point in time passed in `since`. This method will require an auth token to use.

This method will return the fields `event_type`, `event_id`, and `creation_date`. An additional query will be necessary to get the data necessary to display these events to an end user, and developers are encouraged to delay slightly and batch those requests for events of the same type.

We considered and discarded a Twitter-style COMET form of `/events`. The performance and reliability implications are unacceptable. Unfortunately, technologies more suited to such a use case (like WebSockets) are not yet mature enough to adopt. We will revisit this method in a subsequent version, as we acknowledge that repeatedly polling is less than ideal.

**Post Type Agnostic Lookup**

We’re adding a `/posts/{ids}` method, which will accept both question and answer post ids and return a minimal common object.

Fields returned:

- post_type
- body
- owner
- community_owned
- creation, activity, and edit dates
- up and down vote counts
- view count

Developers should still be using the appropriate `/questions/*` or `/answers/*` methods when they know the post type. But there are cases where this is unknown (the most common one being user entered data) and a proper method for handling this case is desirable.

**Alternate Date Format For Requests**

When doing ad-hoc queries and preliminary development the unix epoch format for timestamps is a bit painful when constructing query strings. An alternate syntax of “yyyyy-mm-dd hh:mm:ss” will be accepted.

Like ad-hoc filters, this will only work when not using an app key; so proper production applications won’t be wasting any precious bytes with wasteful date/time encodings.
End of Life For The 1.x API Family
We have no termination date for the v1.x APIs, but once v2.0 is out of beta and we’re reasonably sure of its correctness and completeness we intend to do the following.

1. Stop registration for new v1.x app keys
2. Contact reasonably high usage app authors who haven’t transitioned to v2.0
3. After some delay, disable unregistered requests to v1.1

We’ll monitor usage of v1.1, and once we’re satisfied that its “low enough” we’ll pull the plug. There’s no point in maintaining an older API version any longer than necessary.

If we discover any legitimate use cases where version 1.1 of the API provides a better end-user or development experience than 2.0, there will be a point release to address these short-comings before shutting down version 1.1.

In short, we’re committed to supporting developers on our existing API but fully expect version 2.0 to replace it in the long term.

Version 2.0 Development Timeline
If you’re reading this and not receiving a paycheck from Stack Exchange Inc. it’s sometime on or after September 9th 2011. Active development in accordance with this “specification” will begin as soon as adequate feedback has been received.

We’re anticipating a private beta period a la version 1.0 in roughly a month, followed by a public beta 2 to 4 weeks after that. As with version 1.0, a series of point releases with increasing stability are expected. By the end of the calendar year version 2.0 should be frozen.

These times are estimates and subject to change. Although we anticipate no major difficulty in hitting the “ship it this year” target the start and end times for the beta periods are decidedly less stable.