# Kin Web Client Interface

Client interface ideas and definitions, targeted at standard web architecture.

## Basic definitions

Client interface is based in the form concept: user interacts with the system through queries and results, interfaced by a series of “form windows”.

A form should be defined by a platform and geometry independent language, we are thinking about something like XML, so one unique definition may be used to create web forms, wap forms, or binary data for the standard proprietary kin client interface. Of course, client interface may require also some (or a lot of) code, to do things on the client side. This code may then be specific to the client interface we are using (like javascript for web interface, C dll for a windows binary interface, etc.) It may also be written in a wider supported language, like Java, to hold several platforms. Here we target on web interfaces, but we will try to make definitions reusable for other kind of possible interfaces.

## Server Ideas

Server may be completely proprietary, or we may use an extensible commercial http server (like apache) patched with one or more modules to serve the form pages and items. Proprietary server may be a little easier as we control everything, but usually apache modules may have almost the same degree of control over the process (persistent state machines, etc.). Using a standard web server allows for standard extensions with no extra work, like using SSL for encrypted communication, deflation for compression (not very portable though), etc.

At the first development, though, Web Client Interface will be integrated into the CAP (Client Access Point) as a proprietary http server, to simplify cache management, user validation, and other client management processes). An external client interface may also be used, provided it issues standard queries to the CAPs.

A session may be identified by a single cookie, related to that kin session (but encoded and signed to avoid session intrusions).

Generation of a form window shouldn't be difficult: a main html page with form structure, maybe some other parts like help or selection pop-ups may be downloaded at separated javascript or html documents (also generated). This allows the browser to catch frequently used selection boxes, etc. (like a user selection list) if server signals them with a cache time and they have a reproducible URL (like in GET requests).

Small common javascript code may send some client information to server, like browser screen size, supported colors, etc. so forms may be adapted on the fly without using much javascript code (i.e. we can send pre-formatted html pages instead of javascript to generate every page on the client side).

As one Client Access Module will be serving several remote clients, it may also catch some usual database lists (like users list) to speed up several client lists set up. It may also have pre-calculated pages, like the ones containing those usual lists, so when a client requests that URL-referenced list box it is already available (although most browsers will just send a HEAD request to just ensure it's still the same, and never request it again in one day session)

Using these and some other tricks, a web based client interface may be almost as fast as a proprietary binary client-side one, with the big difference of running on any web browser capable platform. Bandwidth should be small, once the main list boxes and help pages are cached at the browser, and only a quite small latency is required for the user to get instant feedback to its requests: response time when filling a form is much better than a remote terminal service (like Citrix Metaframe®) as it is a local job, and downloading the screen should be a no-pain matter in most cases as html and javascript documents may be kept pretty small and often cached on the client side.

An important point to keep in mind with this web page generator is to use the simplest html and javascript code possible to get a good result, so it may be more portable between different browsers.

Resource consumption at server side is not too big, as there's actually only one copy of everything, irrespective of how many sessions are open, so a single server may handle hundreds of clients (depending on how interactive is the application, generation of result pages requires some memory and time each, but usually it is millisecond units time, or even less).

## Handicaps

Web interface has many advantages, but also some drawbacks. A big one is a looser control from the client side, as code to check or create data there, can prove a little short, and many things you would implement on a binary client, are better run on the server side. This is not a real shame, as server application support is pretty good, and code doesn't have to be distributed on every client. So applications developed with this kind of interface in mind, will have most of their code into TM modules, and very simple “data displaying and retrieving” client interfaces.

## Conclusions

Web client interface makes server side a little busier, and a little more complex, but frees developers and, mainly, support departments, of the troubles associated with a proprietary model that runs differently in every platform, even in every computer depending on its environment. It allows instant access to applications, with zero-time installation and null setup. And absolute ubiquity and freedom in platform choosing at every moment, even in future, is a good point to take into account.

On the other side, more resources are needed on the server cluster (or in its borders), but it is easily scalable. For a mean application, several tens, even hundred or more clients may be managed from a single Client Access Manager (with not a big memory), and servers are easier to manage and care than complex distributed clients.