DOI:10.1145/1866739.1866765

**Building on the OpenSocial API suite, developers can create applications that are interoperable within the context of different social networks.**

BY MATTHIAS HÄSEL

# OpenSocial: An Enabler for Social Applications on the Web

SOCIAL NETWORKING AND interfaces can be seen as representative of two characteristic trends to have emerged in the Web 2.0 era, both of which have evolved in recent years largely independently of each other. A significant portion of our social interaction now takes place on social networks, and URL addressable APIs have become an integral part of the Web.[14] The arrival of OpenSocial[7,8,13] now heralds a new standard uniting these two trends by defining a set of programming interfaces for developing social applications that are interoperable on different social network sites.

Social network sites such as MySpace, Facebook, or XING are all examples of online communities that are technically accommodated through networking software that maps a social graph.[1] This enables individual members to create and maintain personal profiles and to manage their connections to other members within a network community (for example, to friends, colleagues, or business contacts). The networking software often permits the sending and receiving of messages via the respective site as well as supporting so-called update feeds that let users know about their contacts' activity within a given network. In the context of social applications, networking sites are referred to as containers. By means of the OpenSocial API, the container grants its applications access to its social graph (such as profile and contact data), as well as to any messaging systems or update feeds. Used by collaborating people, these applications then create a far richer user experience than software that exists outside a social graph context.[14]

In contrast to applications developed for proprietary environments such as the Facebook Platform,[3,6] OpenSocial applications are interoperable within the context of multiple networks and build on standard technologies such as HTML and JavaScript. The advent of OpenSocial increases a developer's scope and productivity

## » key insights

- **Using the OpenSocial standard, social networks can enable third-party to access their social graph. The ability to develop social applications without having to build a new social graph from scratch opens up a wealth of different business opportunities.**
- **OpenSocial applications are based on the Gadget architecture originally developed by Google. They build on standard Web technologies such as HTML and JavaScript, and prevalent principles such as Ajax and REST.**
- **Key services of the OpenSocial API include People and Relationships, as well as Activities and Notifications. To provide these services, operators of social networks can refer back to Apache Shindig.**


PHOTOGRAPH BY ANDREW MAGER

considerably, as it means that applications need only be developed once, and can then be implemented within the context of any given container that supports the standard. Meanwhile, operators of social network sites are presented with the opportunity to expand on their own existing functionalities with a host of additional third-party applications, without having to relinquish control over their user data in the process.

Until it was made public in November 2007, the OpenSocial standard was driven primarily by Google. The standard was not suited to productive use at that time however, as there were several shortcomings with respect to the user interface and security.[15] The specification is now managed by the non-profit OpenSocial Foundation[13] and, with its 0.8 version, a stable state suitable for commercial use has been reached. By now, more than 20 large social network sites (including MySpace, hi5, XING, and orkut) are using the OpenSocial standard to provide their users with social applications.[10]

## Social Applications

OpenSocial applications are based on the gadget architecture originally developed by Google, which has been expanded on by interfaces that enable access to the social data found in the context of any given container. Gadgets are XML documents containing HTML and JavaScript code along with metadata. The XML specification of a gadget is rendered by the container and integrated into its own Web site. Communication between the gadget and the container operates in such instances via standardized Ajax requests,[5] which are defined in the *opensocial.** and *gadgets.** namespaces of the OpenSocial Javascript API.[11]

From a user's perspective, first-time usage of a social application usually involves some kind of installation process whereby the user explicitly gives permission for the application to access the data in their personal user profile. The developer can enter optional URLs in the gadget specification, to which the container reports user-related events, such as the adding or deleting of the gadget to or from the user account. Once the gadget has been added to the user account, the user can then interact with this gadget. Containers can support several different locations where gadgets may be rendered. These locations are formally known as views. A gadget can request through the API which views are sup-

ported by the container and in which view it is currently being rendered. Four views are specified in the class *gadgets.views.ViewType*:

*Profile View.* The gadget is rendered with other gadgets as part of the user profile. This is particularly useful when users have generated content within the application that they wish to present to other users in their own profile.

*Canvas View.* The gadget is rendered here by itself in a full-screen view. The canvas view is therefore the main place where users interact with the application and create content.

*Home View.* The gadget is rendered on the start page of the container here. Several gadgets are usually rendered alongside one another, giving users an overview of new developments in their own information space.

*Preview View.* This view allows a user to preview the gadget's functionality. It is therefore used particularly often if the user has not yet installed the gadget (for example, on a purely informative page).

Data is often created during the course of user interaction that must be saved on a persistent basis. OpenSocial therefore specifies a persistence layer, allowing developers to store simple name/value pairs for each application or for each application and user. The container assumes responsibility for the actual implementation of this persistence layer, which remains hidden from the developer. Requests to the container are made using the class *opensocial.DataRequest*, which besides methods for the provision of the persistence layer also provides methods for accessing data about people, relationships, activities, and messages.

**OpenSocial applications are based on the gadget architecture originally developed by Google, which has been expanded on by interfaces which enable access to the social data found in the context of any given container.**

### People and Relationships

The OpenSocial API enables direct access to the social graphs of the container. Individual person objects within the graph are represented by the class *opensocial.Person* here. This class supports more than 50 fields used within the context of user profiles on social networks. The majority of these fields are optional, meaning the container ultimately decides which profile data they make available to their applications. Two instances of the class *opensocial.Person* are directly available in the form of the VIEWER and the OWNER. The first object here involves the current user or viewer, while the latter is the user in whose context the application is executed. This implies the VIEWER and OWNER are the same person if a gadget is rendered in the home or canvas view. They can be two different people though in the profile view if the VIEWER views the application in the context of the OWNER's user profile.

The central characteristic of networking software is its support in the creation of targeted connections, and in viewing and traversing their own connections and those made by others within the network.[1] Social applications have to access precisely these connections to be able to support the exchange of information and the interaction between users. OpenSocial enables this with the requests VIEWER_FRIENDS and OWNER_FRIENDS. These differ from one another if a person views another user's profile, but are often not disjoint, meaning the VIEWER can often find their own contacts within the OWNER's contact list.

### Activities and Notifications

Most current networking platforms offer update feeds, which let the user know about the activities of their contacts (for example, "What's new in your network" on XING or "News Feed" on Facebook). Users can find out here, for instance, who has changed companies or has been promoted, who has connected with whom and who has joined which group. Within an activity, the subject or the object(s) of this activity are linked, aimed at increasing user activity on the platform. OpenSocial displays activities using the class *opensocial.Activity* and gives social applications the opportunity to promote themselves in the container's update feed, hence growing organically as a result. On the one hand, an interaction between the user and the container can form an activity (for example, the installation of a gadget or the adding of a gadget to a user's own profile page). On the other hand, an activity can also be an interaction between the user and the gadget (for example, the upload of a file in a collaboration tool or the achieving of a new high score in a game). Containers usually implement the activity concept here in such a way that users can decide for themselves at

any time whether or not they want to allow a gadget to send activities to their contact network.

In order to avoid users having a large number of the same or similar activities in their update feeds, they can be aggregated with the help of activity templates in accordance with the OpenSocial standard, in which developers define general events with placeholders for data specific to applications and users. This separation of content and presentation means containers can display consolidated bundles of activities, thus maintaining a clearly structured update feed. The container reserves the right to determine the ultimate display and execution of each and every activity, meaning the use of activity templates from a developer's perspective can be a mandatory requirement for a container to process activities reported by a gadget in the first place.

The sending of notifications is another method in addition to the activity concept that provides other users within the networking site with user-specific information. Displayed using the class *opensocial.Message*, social applications can feed different types of messages to the container. Once again, the container has control here over whether and how these messages are presented to the recipient (for example, as an email or site message). To protect users' privacy, containers usually give them the freedom to choose for themselves whether a gadget may send notifications to other users at all.

### Internationalization

Social applications can define language modules (so-called message bundles) in their XML specification, enabling them to support any given number of different languages. The gadget is structured in such a way here that all text resources for the user interface, notifications and activities are stored in different message bundles for different languages. Specifications are also in place here to determine which language should be used if a language module is not available for the language required by the container. Language modules are separate XML documents, with names specifying both the user interface language and the target market (for example, *en_UK.xml, en_US.xml, de_ALL.xml*).

### Backend Servers

Social applications frequently communicate with backend servers, which in turn fulfill tasks such as making external content available, carrying out more complex calculations or performing data management by means of a persistence layer that transcends the container. Such backend servers can be implemented by utilizing established Web frameworks, thus overcoming limitations when the gadget operates in JavaScript and HTML. Developers can use *gadgets.io.makeRequest()* to call up HTML, JSON, XML and Atom data from remote servers. The gadget can also transfer data securely to the backend server. In such cases, the container always acts as a proxy, which according to the specifications should always first validate the transferred parameters and then contact the backend server. To ensure that parameters truly originate with the container, OpenSocial makes use of the OAuth API authorization protocol.[9]

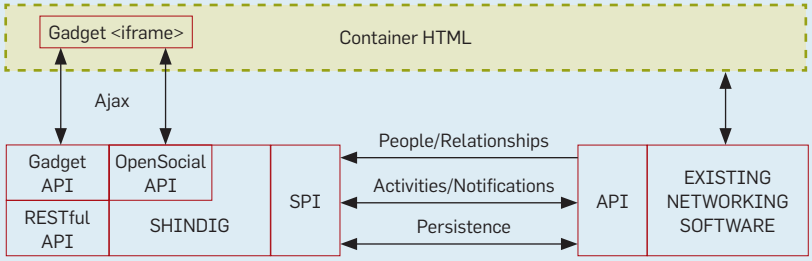
In order to enable data access to servers, mobile end devices, and desktop computers that aren't dependent on Ajax and direct user interaction, the OpenSocial specification—in addition to the JavaScript API—also includes a RESTful API[4] that supports the formats AtomPub, XML, and JSON. Backend servers can make use of this API analogously to the JavaScript API for accessing people, relationships, activities, and notifications within the context of the container. Use of OAuth also guarantees that access to the respective data is only granted to authorized users. The container determines whether and in which form the RESTful API is made publicly available.

### A Reference Implementation

Containers must implement all of the OpenSocial JavaScript API's methods and meet the Gadgets specification to be able to support the OpenSocial standard. Apache Shindig[16] is a reference implementation of the entire OpenSocial stack that operators of social network sites can refer back to. Shindig is open-sourced and developed in parallel in both Java and PHP. The data exchange between Shindig and the existing networking software takes place via the so-called OpenSocial Service Provider Interface (SPI), whose classes are extended in such a way that they access the backend of the networking software. The backend does not necessarily have to be in Java or PHP, provided a suitable API is available that is URL addressable, for instance.

Besides implementation of the gadget container JavaScript (for security, communication, and user interface) and of the OpenSocial container JavaScript (for persistence, people, relationships, activities, and notifications), Shindig includes a gadget server that transforms the XML specification of a social application into JavaScript and HTML code and makes this available via HTTP. The resulting code is usually integrated into the page descriptions of the container as an *<iframe>* element. Shindig also enables the integration of caching mechanisms such as memcached[2] to ensure the gadget server performs sufficiently. A further component of Shindig is the OpenSocial gateway server, which implements the RESTful API amongst other things. The resulting architecture is depicted in the accompanying figure.

**OpenSocial reference architecture.**



### An Example Application

As a real-world example, consider an application that allows the user to create polls, letting their friends within the container vote for different alternatives. Before the user would be asked to enter a question in the Canvas View, the application would identify the user by requesting the OWNER object via Shindig's OpenSocial API. In the background, Shindig's SPI would request the user data from the container's networking software. Subsequently, the application would request the OWNER_FRIENDS, letting the user choose which of their contacts should participate in the poll. The application would then send the poll data and participants list to its backend server using a *makeRequest()* call to the Gadget API, with Shindig acting as a proxy between the gadget and its backend. The application would then create an *opensocial.Message* object in order to send out a notification to each participant, which could be translated by the container into site messages containing a link to the Canvas View, with a unique identifier for the poll. Participants clicking on that link would be identified by requesting the OWNER object in the Canvas View, and would then be asked to vote. The application would pass the vote to its backend using another *makeRequest()* call. The user who created the poll might be allowed to check the participation status and results in the Home and/or Canvas View of the gadget, with the current poll data being requested from the application's backend.

In addition to polls among the user's contacts, the application could also allow the user to create open polls. Using the Profile View, all users that visit the OWNER's profile could be asked to vote. To ensure none of the visitors participate in the poll more than once, the application might identify each participant by asking for the current VIEWER. To obtain participants for the poll, the application could publish the creator and poll name by sending an *opensocial.Activity* object to the container's update feeds.

### OpenSocial vs. the Facebook Platform

As a major player in the field, Facebook offers its developer community a proprietary platform to create social applications.[3,6] More than 660,000 developers and entrepreneurs have used it so far. Basic features in common with OpenSocial include getting people and relationships data, as well as sending notifications and publishing user activities.

Despite its similarity in terms of basic features, the Facebook Platform is quite different from OpenSocial from a technical perspective. OpenSocial (except for its REST capabilities) is rather client-side JavaScript oriented, whereas the Facebook Platform is based on a URL addressable, REST-like server API. OpenSocial gadgets are rendered by the surrounding container and can communicate with their backend servers via JavaScript calls only, whereas Facebook applications rest fully on their developer's Web server. To ensure that applications do not misbehave, Facebook sanitizes the HTML and sandboxes all JavaScript. Facebook therefore requires proprietary languages, including FBML (an evolved subset of HTML), FQL (an SQL-style interface for querying social data) and FBJS (an Ajax library proxied by Facebook).

From a business perspective, the main drawback of Facebook applications is they lack the ability to scale to other networks. OpenSocial applications, in contrast, can scale well—at least in theory. In practice, though, they must be generic in order to do so. In particular, if containers require their applications to adhere to a certain look-and-feel, scaling might no longer work as easily as expected—even if containers do not diverge from each other in their specific implementations of the OpenSocial standard. Within a specific container, respectively, applications are likely to have a diverging look-and-feel as containers do not provide default UI elements that can be reused by application developers. Each application within a container will therefore look different, even for common functionalities. Facebook, in contrast, provides UI elements to make application development easier and more consistent with the Facebook look-and-feel, including headings, error and dialog boxes, as well as a person multi-select widget.

### The Future of Social Applications

From a developer's perspective, the question is not whether to build an application on OpenSocial or for Facebook—but rather what platform to begin with. The great success of the Facebook Platform makes it unlikely that Facebook will join the OpenSocial initiative; yet, the number of containers supporting the OpenSocial standard is constantly increasing. Consequently, application providers will need to consider working with both platforms if they wish to reach a maximum number of users.

With its 0.9 version,[12] OpenSocial has resolved some of its major drawbacks by introducing features such as data pipelining, templates and OSML tags. Data pipelining is a declarative syntax that allows developers to access social graph and backend server data without having to rely on asynchronous JavaScript calls. The container makes this data available in different contexts, such as the JavaScript API and as named variables in OpenSocial templates. Templates allow developers to separate UI markup from programming logic and thus produce code that is more reusable and easier to maintain. OSML tags, finally, provide reusable UI elements such as *<os:PersonSelector>* or *<os:Tabs>* and thus guarantee a common user experience for common functionalities across all applications within a container.

Although the technological basis for developing social applications is advancing fast, some aspects, especially regarding privacy, are still underspecified in the OpenSocial standard. Whether a given application can export data about people and their relationships to its backend server, send out notifications, or publish activities is completely left to the container. Future versions need to address these issues to become real game-changers for the social Web.

### Conclusion

OpenSocial can be seen as a precursor to a paradigm shift that will ultimately see social network sites evolve from enclosed software systems to runtime environments that provide the basic networking functionality and standardized infrastructure that social applications need to operate. In that

sense, OpenSocial is a key enabler for moving from a Web of content to a Web of loosely coupled social applications.[14] At the same time, the idea of being able to develop social software without having to build a new social graph from scratch opens up a wealth of different business opportunities. Internet companies can exploit this, for instance, to boost their outreach and profile immensely—by positioning their existing product on other networking sites as a social application. This is particularly appealing to start-up companies that have to gain a broad user base as quickly as possible with a flexible product, despite the scarce resources they have available to them.

For social network sites, OpenSocial represents a secure infrastructure that enables exclusively authorized parties access to carefully specified parts of the OpenSocial APIs. However, it is users who must always retain control over which parties are granted access to their personal data. C



**References**

1. Boyd, D. and Ellison, N. Social network sites: Definition, history, and scholarship. *J. Computer-Mediated Communication 13*, 1 (2007), 210–230.
2. Danga Interactive. *memcached: distributed memory object caching system.* 2009; http://www.danga.com/memcached/
3. Facebook. *Facebook developers*; http://developers.facebook.com/
4. Fielding, R.T. and Taylor, R.N. Principled design of the modern Web architecture. *ACM Trans. Internet Technology 2*, 2 (2002), 115–150.
5. Garrett, J.J. *Ajax: A New Approach to Web Applications.* White paper, Adaptive Path Inc., 2005.
6. Graham, W. *Facebook API Developers Guide.* firstPress, 2008.
7. Grewe, L. *OpenSocial Network Programming.* Wrox, 2009.
8. Häsel, M. and Rieke, K. OpenSocial. *Informatik Spektrum 32*, 3 (2009), 255–259.
9. Hueniverse. *Beginner's Guide to OAuth.* 2007; http://www.hueniverse.com/hueniverse/2007/10/beginners-guide.html
10. OpenSocial Foundation. *Containers – OpenSocial.* 2009; http://wiki.opensocial.org/index.php?title=Containers
11. OpenSocial Foundation. *JavaScript API Reference (v0.8) – OpenSocial.* 2009; http://wiki.opensocial.org/index.php?title=JavaScript_API_Reference
12. OpenSocial Foundation. *Spec Changes — OpenSocial.* 2009; http://wiki.opensocial.org/index.php?title=Spec_Changes
13. OpenSocial Foundation. *OpenSocial— It's Open. It's Social. It's up to you.* 2009; http://www.opensocial.org
14. Raman, T.V. Toward 2W, Beyond Web 2.0. *Commun. 52*, 2 (2009), 52–59.
15. Schonfeld, E. *OpenSocial Still "Not Open for Business,"* 2007; http://www.techcrunch.com/2007/12/06/opensocial-still-not-open-for-business/
16. The Apache Software Foundation. *Shindig – Welcome to Shindig!* 2009; http://incubator.apache.org/shindig/

**Matthias Häsel** (matthias.haesel@ottogroup.com) is Team Manager Innovation & Synergy Management E-commerce at Otto Group, Hamburg, Germany. He was previously senior product manager at XING, where he led the development of the OpenSocial-based XING Applications Platform.

**Fonte: Communications of the ACM, v. 54, n. 1, p. 139-144, 2011. [Base de Dados]. Disponível em: <http://web.ebscohost.com>. Acesso em: 14 mar. 2011.**