

Sugar Human Interface Guidelines

This document was originally written in 2006 as a goal. It was then mostly, but incompletely, implemented and shipped. Some of the goals have shifted based on user experience; these new goals are mostly visible at [Designs](#). These new goals have also now been partially implemented. There is no single location that documents what has and has not been implemented from these two sets of goals. Work is proceeding to incorporate the new goals from [Designs](#) back into this document.

Introduction

Who Should Read This Document

These guidelines are targeted primarily at developers who are building tools for the OLPC laptop. They provide an in-depth view of the various features of [Sugar](#), the laptop user interface, and focus closely on the parts of the UI that pertain directly to software development and the ways in which applications, presented as "[activities](#)," interact with the operating system.

However, as these guidelines are intended to provide a comprehensive overview of the user interface, these pages should also be of general interest. Hopefully, the descriptions of the various UI elements, particularly in the [Laptop Experience](#) section, will quench the thirst of all who want to better understand the project and its goals.

How to Read This Document

Although many who make it to this page will have read at least one set of human-interface guideline, we strongly request that you read the content of this document in full. While many of the terms contained within will be quite familiar to you, we urge you to review them anyway, since our approach to the user experience shifts away from some traditional models. This document may introduce some unfamiliar ideas around such otherwise familiar terms that you should consider throughout development.

API Reference
Inline references to related APIs appear throughout.

We urge you to read this document once from start to finish, but extensive use of both internal and external hyperlinks will also allow you to peruse its contents at will. Hopefully, this will make revisiting particular parts of the guidelines quick and easy, and will allow you to move naturally through the most pertinent details. Additionally, the document has been laid out in a 3-tier structure — document, chapters, and pages. Feel free to view the document in full to get a broad picture or to print a hardcopy, or use the integrated navigation to move through one chapter or page at a time.

We have included relevant links to the APIs in order to make the relationship between design and implementation clearer. Please take advantage of this as you develop for the laptops.

Providing Feedback

This document remains in constant flux as the project moves forward. We value any feedback that you might have, and would ask that you share any thoughts and suggestions via the [wiki](#) discussion pages. Discussions surround each tier of the document; if you have specific comments, please post them in the discussion for the corresponding page. For more general comments, feel free to use the talk pages at the chapter level or for the HIG as a whole. Links to the talk pages reside next to each section header.

Core Ideas

Activities, Not Applications

There are no [software applications](#) in the traditional sense on the laptop. The laptop focuses children around "[activities](#)." This is more than a new naming convention; it represents an intrinsic quality of the learning experience we hope the children will have when using the laptop. Activities are distinct from applications in their focus—collaboration and expression—and their implementation—journaling and iteration.

API Reference
Sugar Activity APIs

Presence is Always Present

Everyone has the potential for being both a learner and a teacher. We have chosen to put collaboration at the core of the user experience in order to realize this potential. The presence of other members of the learning community will encourage children to take responsibility for others' learning as well as their own. The exchange of ideas amongst peers can both make the learning process more engaging and stimulate critical thinking skills. We hope to encourage these types of social interaction with the laptops.

API Reference
Package: sugar.presence

In order to facilitate a collaborative learning environment, the laptops employ a mesh network that interconnects all laptops within range. By exploiting this connectivity, every activity has the potential to be a networked activity. We aspire that all activities take advantage of the mesh; any activity that is not mesh-aware should perhaps be rethought in light of connectivity. As an example, consider the web-browsing activity bundled with the laptop distribution. Normally one browses in isolation, perhaps on occasion sending a friend a favorite link. On the laptop, however, a link-sharing feature integrated into the browser activity transforms the solitary act of web-surfing into a group collaboration. Where possible, all activities should embrace the mesh and place strong focus on facilitating such collaborative processes.

Tools of Expression

Starting from the premise that we want to make use of what people already know in order to make connections to new knowledge, our approach focuses on thinking, expressing, and communicating with technology. The laptop is a "thing to think with"; we hope to make the primary activity of the children one of creative expression, in whatever form that might take. Thus, most activities will focus on the creation of some type of object, be it a drawing, a song, a story, a game, or a program. In another shift in the language used to describe the user experience, we refer to objects rather than files as the primary stuff of creative expression.

As most software developers would agree, the best way to learn how to write a program is to write one, or perhaps teach someone else how to do so; studying the syntax of the language might be useful, but it doesn't teach one how to code. We hope to apply this principle of "learning through doing" to all types of creation, e.g., we emphasise composing music over downloading music. We also encourage the children to engage in the process of collaborative critique of their expressions and to iterate upon this expression as well.

The objectification of the traditional filesystem speaks more directly to real-world metaphors: instead of a sound file, we have an actual sound; instead of a text file, a story. In order to support this concept, activity developers may define object types and associated icons to represent them.

Journaling

The concept of the [Journal](#), a written documentation of everyday events, is generally understood, albeit in various forms across cultures. A journal typically chronicles the activities one has done throughout the day. We have chosen to adopt a journal metaphor for the filesystem as our basic approach to file organization. While the underlying implementation of such a filesystem does not differ significantly from some of those in contemporary operating systems, it also holds less importance than the journal abstraction itself.

At its core, our journal concept embodies the idea that the filesystem records a history of the things a child has done, or, more specifically, the activities a child has participated in. Its function as the store of the objects created while performing those activities is secondary, although also important. The Journal naturally lends itself to a chronological organization (although it can be tagged, searched, and sorted by a variety of means). As a record of things a child has *done*—not just the things a child has *saved*—the Journal will read much like a portfolio or scrapbook history of the child's interactions with the machine and also with peers. The Journal combines entries explicitly created by the children with those which are implicitly created through participation in activities; developers must think carefully about how an activity integrates with the Journal more so than with a traditional filesystem that functions independently of an application. The activities, the objects, and the means of recording all tightly integrate to create a different kind of computer experience.

Design Fundamentals

Know Your Audience

Inexperienced

The goal of OLPC is to provide children with new opportunities to explore, experiment, and express themselves. Many children in need of such opportunities have previously had little or no access to computing, and so will be unfamiliar with the laptop and how to interact with it. This will undoubtedly have effects on some aspects of activity development. On the one hand, it means that developers must focus energy into making interfaces discoverable, wholly intuitive, and building metaphors that strengthen and clarify the interface. On the other hand, since the laptop will be the first experience of computing for many children, activities do not have to be overly true to legacy behaviors or expectations. This frees developers to innovate.

Young

Many of the children receiving laptops will be as young as five or six; others will be in their mid teenage years. Additionally, those that receive them at a young age will continue to use them throughout their education. Therefore, it is important to develop activities in ways that scale well across age levels.

International

The OLPC initiative, by its nature, requires international involvement and participation. Developers must keep in mind the broad range of cultures and languages that the laptops must transcend. In particular, activities should not depend on western icons and modes of thinking, but should abstract ideas to a level that would be familiar to humankind in general, where possible. For instance, consider the camera button on the keyboard. Though one might be inclined to label this key with a small image of a camera and lens, the eye graphic speaks directly to our human capacity for vision, providing a cross-cultural icon that represents the computer's ability to capture what it sees.

Key Design Principles

Low floor, no ceiling: this mantra should guide your development efforts for OLPC. All activities and interfaces should be designed in such a way as to be simple and intuitive to users of all age groups, nationalities, and levels of computer experience. At the same time, we don't wish to impose unnecessary limitations on the software either. Instead, we hope to create a platform suitable for all kinds of creative expression which provides a low floor to the inexperienced, but doesn't impose a ceiling upon those who are. This is a worthy goal, but will require a genuine effort on the part of developers, who must take many aspects of design into account. The following list, while certainly not comprehensive, provides a starting point for such considerations.

Performance

The OLPC laptop bucks the trend of "more, faster, fatter"; we aim to provide a computer tailored to the needs of children in the context of their learning, not to the needs of frantic video games or office applications. We are, however, working within constraints of component cost, robustness, and power consumption. To satisfy these constraints, we have opted for NAND flash rather than a hard disk and a modest 256MB of memory (Please see [hardware specifications](#)). Thus, developers must make every effort to write efficient code while minimizing memory usage.

Since there is no swap space on the laptop, only a limited number of activities can run concurrently; the Sugar UI exposes these details directly to the children. The [Home](#) screen features an activity ring that contains icons representing each instance of an open [activity](#). The size of the ring segment that a given activity occupies represents its overall memory usage; when the ring fills up, no additional activities may be launched until some resources have been freed. Take these limitations into account as you develop activities, since they will have a greater impact on the performance of your software on the laptop than on other platforms.

Usability

OLPC places an emphasis on [discoverability](#) and usability due to our [target audience](#). Usability has everything to do with the actual behavior of the activities, the layout of the buttons and tools, and the feedback that the interface provides to the children when they interact with it. Ultimately, the design decisions that make your activities usable will depend greatly on the type of activity you are developing, and it will be up to you to consider carefully the kinds of interactions that the children will expect when presented with it. As a general rule, if the interface provided does what the child expects it to, you are off to a good start. However, since it is quite difficult to know what they will expect—and in practice not all children will expect the same things—there is no substitute for user testing.

Simplicity

We designed the entire laptop interface with a goal of simplicity. It can be tempting—and also quite easy—to add an overabundance of features to software: the abundance of MIPS and memory often exacerbate the software-bloat phenomenon. The laptop hardware "limitations" lead toward a more concise direction and aid in designing for simplicity.

Keep in mind that simple doesn't necessarily mean limited. OLPC hopes to demonstrate to the world that simple—even minimal—controls can have great expressive power. Avoid bloated interfaces that do too much, and limit the controls to those immediately relevant to the task at hand. Rather than creating a "Swiss Army knife" of an activity, think of the laptop itself as the knife, and instead develop a particular tool for that knife that does one thing, and does it very well. When all the activities on the laptop embrace this idea, the true power of the laptop will emerge.

Reliability

Of course we want to avoid instances where things go wrong; this should be a goal for every piece of software. We are committed to ensuring that the UI framework prevents activities from causing

system crashes. Developers should consider a "fail-soft" approach to their designs, such as incorporating a suitable behavior for the spontaneous termination of an activity.

Security

Security is detailed elsewhere in this wiki (See [OLPC:Bitfrost](#)). Our goal is to protect against five categories of "bad things" software can do:

- damaging the laptop;
- compromising privacy;
- damaging the children's data;
- doing bad things to other people; and
- impersonating the child.

It is important to include anti-theft measures and common mechanisms for objectionable-content filtering.

From the perspective of the user interface, it is important that these goals be achieved without the use of menus, pop-up boxes, passwords, etc., as these approaches are meaningless to most people.

Adaptability

Several use conditions should be taken into consideration in designing activities: the laptop has both a grayscale (sunlight) mode and a color (backlight) mode; the mesh—while always available—may or may not be connected to the Internet at the time the activity is active; the laptop may be configured in either laptop mode (keyboard and touchpad exposed) or [handheld mode](#) (game controller, camera, microphone and speakers only). Signal strengths, and therefore bandwidth, may fluctuate, and at times activity participants may even drop off temporarily. Activities should handle all of these cases with care. E.g., temporary loss of connectivity should be handled silently, and reconnection of an individual to an activity they were previously participating in should happen with no noticeable side-effects as outlined in the guidelines for [activity robustness](#).

Recoverability

Recoverability is fundamental to encouraging exploration. With creative exploration among OLPC's main goals, it therefore becomes an issue of high importance on the laptops. When children know they have a fallback plan—a way back to the current state of things—they will much more frequently go beyond their comfortable boundaries and experiment with new tools and new creative means of expression.

The [journal](#) provides a partial notion of recoverability, since its auto-journaling amounts to maintaining an automatic incremental backup. The ability, for children, to choose to "keep" anything they're working on in its current state furthers this idea.

However, the primary and essential means of recoverability remains the ability to undo one's actions. Of course, the notion of undo/redo becomes complicated in the realm of collaborative editing, which imposes a limitation on the extent to which undos are possible, since collisions could often occur between the things one child wants to undo and the things another child has already changed since those were done. Nonetheless, we are dedicated to providing this functionality to every extent possible, and [activities](#) should strive to support this to the best of our ability.

(Future revisions of the [keyboard](#) may even have an undo/redo key to further strengthen this idea.)

Interoperability

Coming soon...

Mobility

Of course, as with all portable computers, a general notion of mobility is intrinsic to the laptops. However, in the hands of children, this mobility rises to a new level, since we can expect that they will carry them not only to and from school, but on a hike, onto the playground, or to any number of other locations where they can learn and experience the world. Their physical form has been designed with ruggedness in mind. The important thing to consider is the effect such mobility can, and should, have on the activities themselves. The lens of the built-in camera looks a lot different when it's treated not as a simple webcam, but as a way to capture the world around them, both indoors and out.

Transparency

OLPC also hopes to encourage the children using the laptops to explore the technology under the surface. Towards this end, a [view source](#) key has been added to the laptop keyboards, providing them with instant access to the code that enables the activities that they use from day to day. This key will allow those interested to peel away layers of abstraction, digging deeper into the codebase as they learn.

To enable such layered exploration, OLPC has written much of what can be in [Python](#), a scripting language, to enable children to view the source code. This means, aside from general good practice, code should be both readable and well commented. The [PEP 8](#) style guidelines for Python provide an excellent resource, and OLPC recommends that developers follow the practices laid out therein unless a compelling reason exists not to do so.

Accessibility

There are lots of things to think about that relate to accessibility in a set of human interface guidelines. We've just started hashing out general accessibility issues at the [OLPC:Accessibility](#) page.

Broadly speaking, the user interface of the GUI shell and of activities must address the following accessibility issues:

- Using the interface only from the keyboard (without a mouse or trackpad)
- Using the interface without requiring the ability to distinguish color (a significant portion of the population has some level of color blindness)
- Providing an enlarged print/icon option for folks whose vision is less than 20/20 (but who still can see things that are somewhat enlarged - e.g. 18 point fonts)
- Using the keyboard without needing to press more than one key at a time (all modifiers must work with AccessX functionality)
- Supporting programmatic access to the GUI (which for us will mean supporting [ATK](#) in Sugar and all activities)
- Either shipping with some number of assistive technology applications (is a screen reader an "activity"?), or making them easy to download
- Providing some way for a user to discover accessibility support and enable what they need (Windows XP & Vista offer an "accessibility wizard" for this purpose; we don't have good upstream technology from GNOME we can take for this unfortunately; the Ubuntu accessibility folks are perhaps furthest along in thinking about this)

The Laptop Experience

Introduction

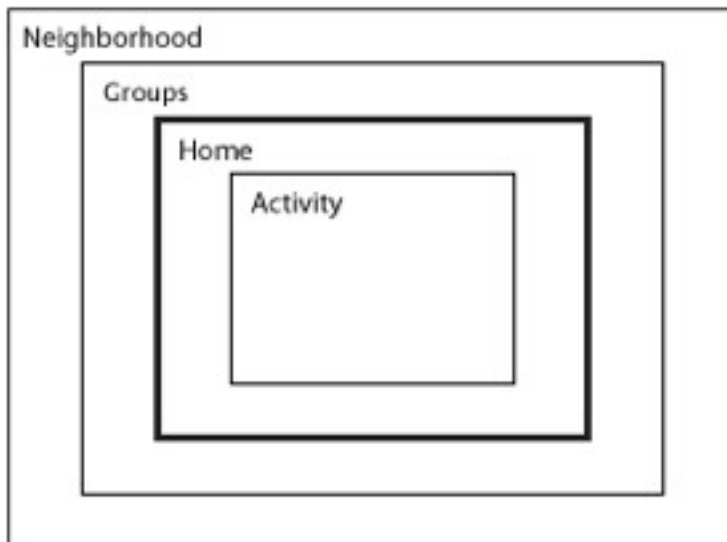
Most developers are familiar with the [desktop metaphor](#) that dominates the modern-day computer experience. This metaphor has evolved over the past 30 years, giving rise to distinct classes of interface elements that we expect to find in every OS: desktop, icons, files, folders, windows, etc.

While this metaphor makes sense at the office—and perhaps even at home—it does not translate well into a collaborative environment such as the one that the OLPC laptops will embody. Therefore, we have adopted a new set of metaphors that emphasize community. While there are some correlations between the Sugar UI and those of traditional desktops, there are also clear distinctions. It is these distinctions that are the subject of the remainder of this section. We highlight the reasoning behind our shift in perspective and detail functionality with respect to the overall laptop experience.

- Desktop : [Neighborhood](#)
- Menubar : [The Frame](#)
- Hierarchical Filesystem : [Journal](#)
- Applications : [Activities](#)
- Files : [Objects](#)
-

API Reference
Package sugar
Package sugar.shell

Zoom Metaphor



Four distinct zoom levels define the laptop: Neighborhood, Groups, Home, and Activity

The mesh network is a permanent fixture of the laptop environment and is represented explicitly in the interface. A zoom is used to relate four discrete views, each of which caters to a particular set of goals: Home, Groups, Neighborhood, and Activity. Using keyboard shortcuts or controls in the [Frame](#), children may zoom in and out on the mesh community.

API Reference
Package sugar.shell.view.home

Home



The Home view: Each child chooses a dual-tone color scheme for her XO character that is used throughout the interface. Activity icons are color-coded by the child who launched the activity.

Of all the zoom levels, the Home view relates most closely to the traditional desktop. As the first screen presented to the child at startup, it serves as a starting point for the exploration of both the mesh network and also of personal activities and objects. From this view, the child may either back up first to their [Groups](#) — such as their Friends or their Class — and beyond that to a view of the entire mesh [Neighborhood](#), or, instead, zoom in to focus on a particular [Activity](#).

The Home view interface is minimalistic. In the center of the screen, the [XO icon](#)—rendered in the child's user-specified colors—represents the child to whom the laptop belongs. The activity ring surrounds the character, indicating all of the currently open activities. Furthermore, the section of the ring that a given activity occupies directly represents the amount of memory that the particular activity requires to run, providing immediate visual feedback about memory constraints and exposing a means for resource management that doesn't require knowledge of the underlying architecture. Most activity management happens here: starting new [private activities](#), ending current activities, and switching between activities.

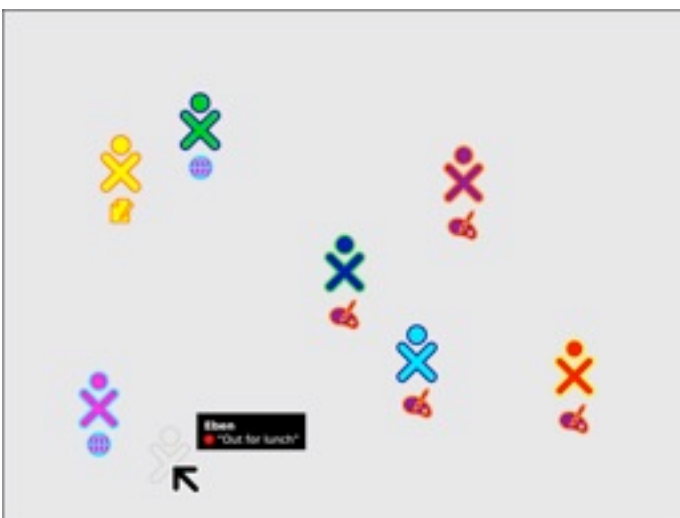
API Reference

[Package sugar.shell.view.home.activitiesdonut](#)

When used in conjunction with the [Bulletin Board](#), the Home view becomes the most direct correlate to a typical PC desktop as a place for keeping things handy: tomorrow's homework, a drawing one is working on, a favorite song, a reminder to oneself to do one's chores, etc.

See [Activity Management](#) for new design images.

Groups



The Groups view: Members of the currently selected group and their current activities are visible from this view. Hovering over a "missing" XO reveals an "away message."

The Groups view takes a small step back from the child's Home space, opening up to include their circle of friends, their classmates, and any other groups to which a child belongs. The Friends group essentially represents a spatially viewable and editable buddy list. From here the child can add or remove friends and move individuals around, perhaps arranging them logically. The Class group is defined dynamically, and includes all others in the same class, and their teachers as well. This group

provides the perfect space for working and sharing with classmates, posting projects for class critique, or for picking up the homework assignment the teacher posted to the class [Bulletin Board](#).

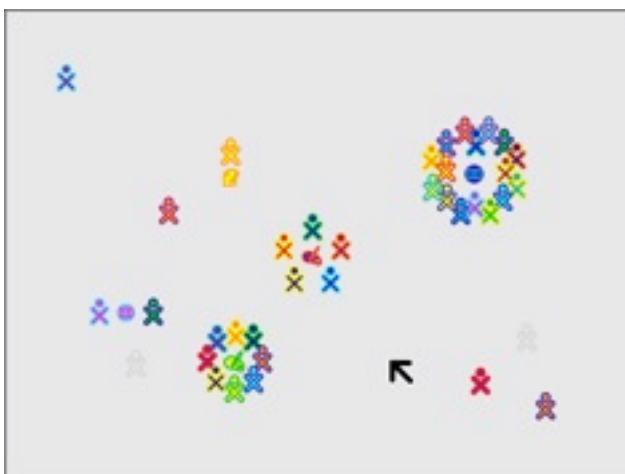
In addition to several special classes of groups, children may also generate groups on their own. This might provide a way for a close group of friends to keep up with each other's activities, or for a group of aspiring photographers to share photos. In a classroom setting, this provides a way for the children to create temporary groups for working on classroom exercises, or long term groups for extended projects. To create a group, a child can search for or select any number of individuals on the mesh. Each of these individuals will receive an invitation to join the group, and upon accepting the invitation will have its name added to their list of Groups, where they can see and chat with members, and post to the group Bulletin Board. Although one person initially creates a group, groups are not managed. Instead, people may choose to leave a group on their own, and anyone in the group may invite other members into it. When this happens, all current group members receive introduction [notification](#), making them aware of the new member. This open model simplifies the interaction and encourages the learning of natural social dynamics instead of attempting to enforce them via rules and restrictions.

Groups have several advantages. First, it allows the children to view their friends, classmates, and other groups, and allows them to freely chat with them as well. Additionally, each group will have its own private Bulletin Board where members can post notes and share Objects. Finally, all of the members of the selected group — a child's friends, for instance — receive [invitations](#) whenever the child starts an activity from the Groups view, making collaboration implicit. Moreover, this view allows the child to see what activities their class, friends, and other groups are presently engaged in, providing the opportunity to [join](#) any non-private activities. Already, you can see how this view changes the usual method of application launch, allowing one to start new networked activities or join existing ones directly.

API Reference

[Package sugar.shell.view.home.FriendsBox](#)

Neighborhood



The Neighborhood view (or Mesh view): All children visible on the mesh network can be seen clustered around shared activities; Away messages are also accessible from this view.

Zooming out one more step we reach the Neighborhood view. Here children can see everyone on their local mesh. At this level we intend to support a variety of views, each with a different focus: the individuals; the activities in which they are presently engaged; etc. In the figure, individuals are shown clustered around their currently active activities, providing a direct visual representation of the popularity of an activity, since group size is immediately perceptible.

While the Neighborhood view doesn't currently provide any true spatial or geographical data, it does provide an at-a-glance social geography of the mesh and its participants. Similar to the Groups view, launching an activity here implicitly opens that activity up for anyone in the Neighborhood to join. While no one receives an [explicit invitation](#) in this case, the newly started activity will appear in the view, with its participants clustered about it, so that anyone who wishes to may join. Of course, this also means that the Neighborhood provides an excellent space for exploration. Here, one can both search for, locate and [join activities](#) of interest using a powerful and adaptable [search](#) technology, and also interact with and make friends with other children in their neighborhood they haven't yet met.

API Reference

[Package sugar.shell.view.home.MeshBox](#)

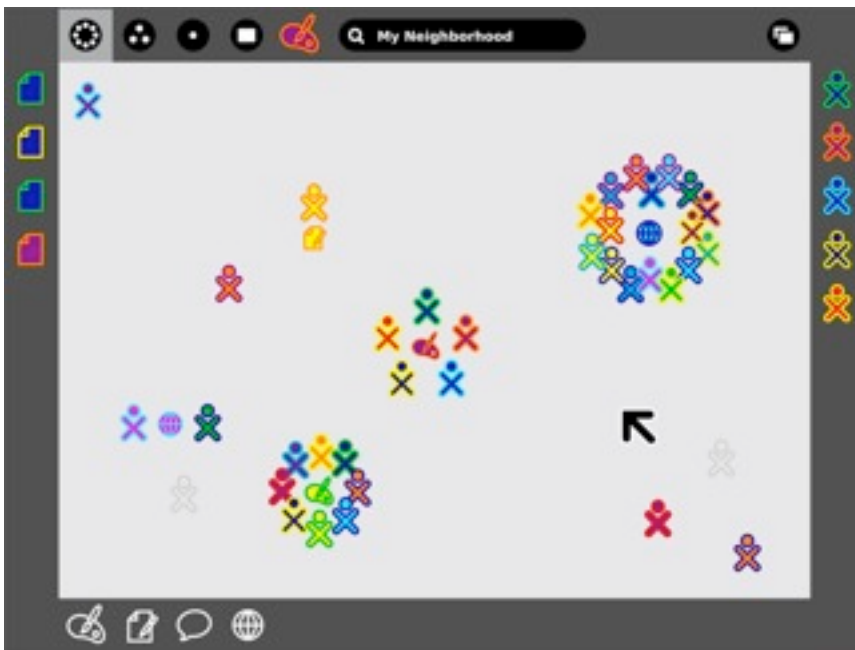
Activity

Zooming in from the Home view, a child finds the Activity view. This view contains the activities where all of the actual creation, exploration, and collaboration takes place. This is where you, the developer, come into play, providing new and engaging tools, extending the functionality and encouraging new types of creative exploration.

Though multitasking has become somewhat of a standard in today's desktop computing world, we've chosen to break away from this model, instead adopting a fullscreen activity view that focuses the children's energies on one specific task at a time. Although one may have several activities open in the activity ring at any given moment, only one can be denoted as the active activity (similar to focus in a window system). Several factors contributed to this decision: first, although the laptops have an extremely high-resolution display—200dpi—the actual viewing area remains quite small—a modest 7.5-inch diagonal—leaving little room for multiple activities on the screen; second, as noted, it naturally focuses efforts on a specific task. The [Frame](#) detailed below serves as the interstitial tissue between activities. As a visual extension of the [Journal](#), it enables objects to move between activities.

For extensive detail on the various aspects of the activity interface and their design guidelines, see the [Activities](#) section.

The Frame



The Frame: The left, top, and right sides of the frame represents nouns: persons, places, and things. The bottom of the frame represents those elements that require action: activities, invitations, and notifications.

Always Available on the Periphery

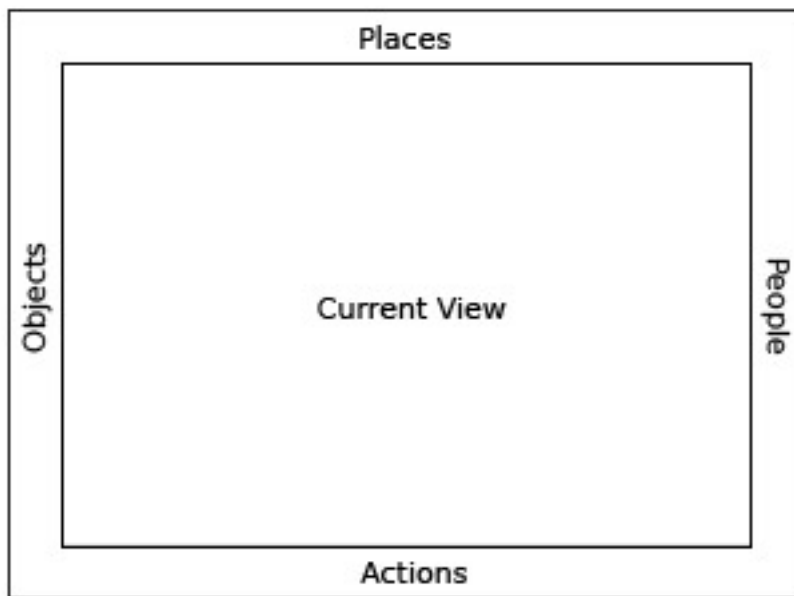
Glancing at the previous screen shots, you might have noted the absence of a menu bar or other form of persistent interface element. Such a persistent element reduces the screen space available for activities; since screen is at a premium, we have opted to use

a frame—always on the periphery and just out of sight—to contain all of the peripheral information that a child might need, across all views. Since the Frame persists at all zoom levels, it provides a consistent place for those interface elements which apply to all views, including search, incoming invitations and notifications, a clipboard, and buddies you are currently interacting with.

When [activated](#), the Frame slides in atop the currently visible view, providing access to needed functionality, yet quickly retracting from view once the task for which the child invoked it ends. Although these transitions happen quickly, a forgiveness parameter prevents unintentional Frame retraction in hopes of making interaction with this interface element completely natural.

See [Frame](#) for new design images.

Frame Components & Organization



The Frame is organized into distinct logical groups.

At a high level, one can consider the frame in two parts: The left, top, and right sides of the frame represent nouns: things, places, and persons. The bottom of the frame represents those elements that require action: activities, invitations, and notifications. More specifically, each edge of the frame is dedicated to one of people, places, objects, or actions.

People

As previously mentioned, the presence of others on the

mesh defines much of the laptop experience. In order to surface this at all times in the interface, the right-hand edge of the Frame provides an easily accessible list of all the individuals a child is collaborating with in the currently active activity, represented by their colored [XOs](#). This has a number of benefits. First, it provides a quick reference of the people the child is working with, which updates as new people join and others leave. As new people arrive, they appear in the upper right corner, and as they leave they simply vacate their current location. Additionally, the secondary rollovers for these XO objects reveal biographical information about them: name, age, class, interests, and even a small photo. This makes the frame a great resource for meeting new friends, for what better place to meet them than in the activity shared with them?

API Reference

[Package sugar.shell.view.frame](#)

[Module: ...frame.FriendsBox](#)

[Module: sugar.shell.model.BuddyModel](#)

Places

Of the various frame components, the Places category is the most abstract. However it also emphasizes the metaphors that the zoom levels build upon. In the upper left-hand corner reside the zoom buttons, which can instantly transition the user among the Activity, Home, Groups, and Neighborhood views. For clarity, the upper left-hand function buttons on the keyboard have identical icons and functionality.

On the other side of the Places edge resides the Bulletin-Board button. Again, this button has an analogous key on right-hand side of the keyboard's function keys. Discussed later, this button acts as a toggle for an auxiliary layer which can provide contextual chat and a place to share objects. This button functions within the Places bar because it acts as a modifier to any view. In a sense, it adds an additional layer of context to any other "place" on the laptop.

API Reference

[Module: ...frame.ZoomBox](#)

Finally, though not less importantly, this section of the Frame contains the [global search field](#).

Objects

The clipboard has become a staple in any modern operating system. Nonetheless, its implementations have changed little, if at all, in decades. The clipboard has one "page", to which you can copy to, cut to, and paste from, and in most cases this hypothetical page remains invisible: to see what's

on it, you've got to paste its contents. While this isn't strictly true (On Mac OSX, for instance, an item at the bottom of the 'Edit' menu allows you to 'View Clipboard Contents'), most users are oblivious about viewing its contents, as one must explicitly seek it out. This basic model, while simple, often falls short of many use cases. Thus, OLPC has extended the traditional clipboard, empowering the user with added functionality without increasing complexity.

On the laptops, the clipboard takes the form of the left-hand edge of the frame. This region serves as temporary storage for objects - a paper, an image, a sentence, a URL - facilitating their transfer among activities and, perhaps more importantly, among the various zoom levels. Any type of object that can be stored in the Journal can likewise be transported via the clipboard. A child may place an object on the clipboard in a couple of convenient ways. First, keyboard shortcuts will provide an interface for simple copy and paste functions in the way already familiar to us. Additionally, since objects support direct manipulation, the child may simply drag a photo, file, or selection onto the frame in order to copy it, and may then drag it out to paste it in another location, such as within another activity, on a friend, or to a [Bulletin Board](#). As items are placed on the clipboard, they are arranged temporally in a push-down stack, the most recent clipping appearing in the upper-left-hand corner of the frame.

With the presence of a clipboard which contains multiple items, it becomes necessary to add a means for selecting an active clipping as the source for any paste command. Since the usual copy/paste keystrokes will quickly become familiar to all, any invocation of the copy shortcut will automatically place the resulting clipping at the top of the stack, selecting it as the source, so that a subsequent paste command behaves as expected. When not using these traditional shortcuts, a single click on any object in the clipboard will select it, visibly indicating it as the new source. Additional copy commands (or drags) will continue to add elements to the clipboard stack. Once the clipboard reaches a predefined limit, the elements at the bottom of the stack will begin to drop off making room for the new ones. Elements may also be removed explicitly by the user via their contextual rollover, and a modified paste shortcut for advanced users will serve to both paste an item and pop it from the stack at the same time.

The resulting clipboard will behave identically to those on current operating systems, while at the same time providing drag and drop support, clipboard history, and previews, as well as advanced functionality for advanced users.

API Reference

[Module: ...frame.clipboardpanelwindow](#)

Actions

The bottom edge of the frame functions primarily as an [activity launcher](#), but it also accumulates both incoming [invitations](#) and [notifications](#). As a starting point for instantiating activities, this part of the frame is fairly straightforward. Whenever an activity receives a click, a colored instance of that activity appears within the activity ring in the child's own colors, and invitations are automatically sent as appropriate. On the other hand, anytime the child receives an invitation it appears as a colored activity icon (in the color of the inviting XO, of course), clearly distinct from the uncolored outlines of the activities which reside on the child's own machine. Since an invitation to join an activity has no functional differences from starting, the invitations appropriately indicate this by their similar form. The rollover state for these invitations allows the child to accept or decline the invitation, optionally providing a reason for declining.

Notifications, the third aspect of the Actions edge of the frame, function slightly differently. While they don't represent an activity that the child can join, they do come as messages from activities or from the system, conveying important information about the state of the activity or system status such as battery strength or wireless signal. Though slightly different from activities and invitations, these notifications still require some action on the child's part, and are an appropriate addition to the frame which provides a convenient way to access them from within any view. Just as in the other edges of the frame, invitations and notifications organize by time, the most recent always in the lower left-hand corner, so that the child may handle them in a timely manner.

API Reference

[Module: ...frame.ActivitiesBox](#)

Activation Methods

The Frame has multiple activation methods.

Hot Corners

Hot corners serve as the Frame's primary invocation method. As [Fitts' Law](#) implies, the corners are the easiest part of the screen to hit with a cursor. Moving the cursor to any corner of the screen will instantly invoke the frame. From a corner, one can readily scroll along an edge in search of a desired element. Since newly added people, objects, and invitations insert from the corners, the latest invitation, clipping, or participant is always close at hand.

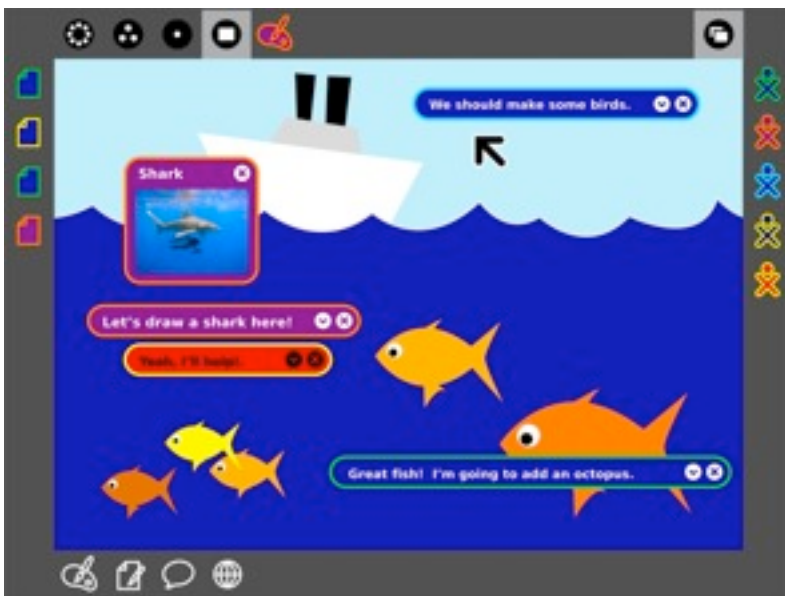
Function Key

In addition to trackpad-based activation, the information within the Frame lies just one keystroke away. A dedicated key has two modes of functionality: (1) momentary presses act as a toggle, turning the Frame on and off with each press; and (2) holding the key down, the Frame will appear on screen until release of the key. This latter method provides a quick way to glance at incoming invitations or other system status elements for a brief moment and then return full focus to the activity view itself.

Notification Overrides

Though rare, some urgent [notifications](#) such as low battery levels may override the Frame, automatically bringing it into view without user interaction. These overrides come from the system only; applications do not have privileges for override, although they may alert the user via standard notifications.

Bulletin Boards



Bulletin boards provide a layer for contextual chat and sharing around any view.

Since the laptops have implicit connectivity via the mesh network, an additional layer of the UI has been designed to take advantage of it: Bulletin Boards. Taken literally, the Bulletin Boards provide a space for *posting things*.

Context is key to the usefulness of Bulletin Boards on the laptops. A button in the Places edge of the Frame toggles the Bulletin Board layer on and off, and although only one button exists for this purpose, each view among the various zoom levels has its own Bulletin Board.

The scope of individuals who have access to a given Bulletin Board matches the scope of individuals that the view currently represents. For example, any items posted to the Home view Bulletin Board may only be seen by the child that posted them, effectively providing a traditional desktop environment. Likewise, anyone within the child's list of Friends may view things the child has posted to the Friends view Bulletin Board, and all of a child's classmates and her teacher can view her posts to the Class Bulletin Board; the Mesh view Bulletin Board provides an environment for sharing with the entire laptop community. Furthermore, each activity has its own Bulletin Board, providing a space for sharing files and ideas surrounding the activity itself that don't have a place within it.

Spatially Contextual Chatting Interface

As a transparent layer above any view, the Bulletin Board provides a spatially contextual chatting interface. This means that, unlike traditional forum-style threads that organize temporally, chat bubbles may be freely positioned on screen. Discussions formulate around specific areas of the activity beneath. Annotation-style comments open the door to a wide variety of conversational interactions. In a drawing of the ocean, for instance, one conversation could be happening below the water's surface, while another group of children discuss what kind of birds fly through the sky. In another situation, one child could remotely assist another in learning how to use a new activity, pointing out specific interface elements with detailed descriptions of their functionality. In a literary application, child or teacher could proofread another child's story, correcting spelling mistakes, pointing out grammatical errors, and sharing thoughts about specific sections of the story without directly editing the work on the activity layer beneath.

An Environment for Sharing

In addition to contextual chats, Bulletin Boards provide a space for sharing. Any object may be posted to a Bulletin Board for others to look at and enjoy and to pass on to others, promoting viral sharing. The sharing environment is an integral element of all views.

In the Home view, for instance, only the child to whom the laptop belongs has access to the contents of the Bulletin Board. Here, the Bulletin Board provides a convenient area for the temporary storage of objects and activities, as well as those things kept around for quick access: tomorrow's homework assignment, the pictures taken last week, the book the child is reading, a favorite game. In this way, the traditional desktop that the zoom levels replaced finds its way back through the Home view Bulletin Board. The functionality described here mimics the traditional desktop to some extent. Note that the contextual chat bubbles are also available in the Home view, providing a mechanism for writing "notes-to-self". The Bulletin Board metaphor emphasizes a temporary and ever changing space for placing objects, distinctly separate from the space in which they are stored. This may prevent the common overuse of the desktop as the primary place to store *everything*, which limits its usefulness as a quick way to find the files that matter most at any given point in time.

From the Friends and Mesh view, the Bulletin Board serves as a place to share interesting things a child has found or created with friends, and the entire mesh respectively. The important thing to remember with regard to sharing, of course, is that "Share this JPEG (or GIF or SVG or any other picture format) file with Bob and Sue" translates to "Share this photo (or picture) with Bob and Sue." The sharing metaphor functions much more naturally than the file transfer systems we're used to, since file transfer really just represents a technical implementation of the more abstract idea of sharing in the first place. Of course, children can also view the things that others have posted as well. Moreover, as a community space, group sharing occurs naturally.

Finally, the shared space the Bulletin Boards provide take on a slightly different, yet quite powerful meaning at the Activity level. Again, contextual to the current view, each activity has its own shared Bulletin-board. Posts to this layer provide supporting materials for the underlying activity that other participants in the activity may view (Or, if they'd like, keep for themselves). This actually means a great deal, since any object at all can be shared this way, including many objects that the activity itself may not provide support for. For instance, though one couldn't paste a song inside a drawing (no compound document), a song posted to the Bulletin Board layer could provide inspiration for it. Similarly, it provides a means of collecting materials relevant to the task at hand within the activity. Rather than having 5 individuals each pasting images of a shark directly into the drawing, they could instead post them to the Bulletin Board for others to see and discuss before deciding which to use as a basis for the drawing. Thus, Bulletin Boards provide a space for gathering research and supporting materials, and holding discussions around both them and the activity.

View Source

While Bulletin-boards provide a layer of abstraction on top of any given activity, the [olpc:View Source](#) button allows one to look behind the activity, peeling away layers of abstraction in order to reveal the underlying codebase which makes it tick. This feature will integrate cleanly with the [olpc:Develop](#) activity, encouraging children to view, modify, and redistribute variations on the activities they use. Through collaboration and sharing, a garden of home grown activities will begin to develop on the laptops, created by the children themselves.

The Journal

See [Journal](#) and [Design Team/Proposals/Journal](#) for new design images.

The Notion of "Keeping"

We believe that the traditional "open" and "save" model commonly used for files today will fade away, and with it the familiar floppy disk icon. The laptops do not have floppy drives, and the children who use them will probably never see one of these obsolete devices. Instead, a more general notion of what it means to "keep" things will prevail. Generally speaking, we keep things which offer value, allowing the rest to disappear over time. The Journal's primary function as a time-based view of a child's activities reinforces this concept.

Most of us have heard the "save early, save often" mantra, largely ignored it, and incurred the consequences. Sugar aims to eliminate this concern by making automatic backups. This lets the children focus on the activity itself.

The physical analog for an Activity is paper, pencil, & writer at a desk, canvas, paint, & artist at an easel, clay & hands at a wheel, or sidewalk, chalk, & children at a driveway. They exist as created and modified. No 'saving' action, per se, is needed. The 'Keep' toolbar button (proposed to be 'Copy', [1]) provides the capability to replicate the current state of the Activity into a new, separately available* Activity object in the Journal. Outside of software culture, such rapid and perfect replication has few physical analogs, and perhaps so has been a point of confusion.

* Note: The Keep(Copy) action currently produces an object that shares an ID with its parent and so may not be opened simultaneously with its parent in the same Sugar instance. (This is a bug.)

Use case: A teacher could easily prepare a series of [Activities/Physics](#) models illustrating the construction of a system through stages. The learning of the physical principles could be staged into separate lessons or exercises.

Incremental backups occur regularly and are also triggered by such activity events as changes in scope, new participants, etc. To cater to the many needs of the editing environment, activities can also specify "keep-hints" which prompt the system to keep a copy. For instance, a drawing activity may trigger a keep-hint before executing an "erase" operation immediately preceded by a "select all". Of course, a child may choose to invoke a keep-hint by selecting the "keep in journal" button, but the increasing adoption of this new concept of keeping should ultimately eliminate this.

Based on the Object model associated with files, each kept Object is, technically speaking, a separate instance of the activity that created it. This eliminates the need "to open" a file from within an application, and replaces the act of opening with the act of resuming a previous activity instance. Of course, a child will have the option to resume a drawing with a different set of brushes, or resume an essay with a different pen, and will so be provided with an "open with" (a different tool kit)-style of functionality; but, no substitute for an "open" command will exist within an activity's interface.

Deprecating Hierarchy

Temporal Organization

Along with the idea of implicit keeping, the laptops will drastically minimize the hierarchical filesystem as a means for organization, replacing it with a temporally organized list of activities and events, furthering the Journal metaphor. This drastically simplifies the auto-keeping behavior, since it eliminates the need to specify a location in which a newly started activity should be kept; naturally, the newly started activity will appear as the most recent entry in the journal.

Temporal organization functions naturally in the absence of explicit or hierarchical methods, since humankind's intrinsic relationship to time gives them, at the very least, a relative notion of "how long ago" something happened. By moving back through the Journal, a child can simply locate the period in time within which she knows she made something, and then employ additional use of searching, filtering, and sorting to pinpoint exactly what she's looking for.

Falloff

Due to the laptops' limitations in storage capacity, the potential exists for the Journal to contain so many entries that no more may be written. However, the frequency of such occurrences is limited by temporal falloff, which tidies up the Journal contents and keeps space available for new entries. One might think of this as an intelligent combination of garbage collection and disk defragmentation.

The driving principle here is that of temporal granularity, derived directly from our very capacity for human memory. Our minds, generally speaking, maintain a high level of granularity with respect to very recent events, but only a low granularity for events from several years ago. Moreover, this granularity tends to follow a logarithmic curve, where the past few minutes remain quite clear, the past few hours more blurry, and by last month quite vague. When we look years into the past, only specifically memorable events stand out in our minds.

On the laptops the policies are a bit more strict, but the principle remains the same. With a finite amount of memory, there must exist some means of managing what's remembered, or kept; and what's forgotten, or erased. An intelligent algorithm will assist children in identifying "forgotten" entries. Taking into account how old an entry is, how many times she's viewed it, how recently she's worked on it, how many hours she's worked on it, how many people she's worked on it with, its tags, and even more forms of automatically generated metadata, the Journal can suggest to her those entries which it feels can be erased. She will then have the opportunity to review those items prior to their erasure, if she wishes, and can keep any she still feels attached to.

In a time where gigabytes have become cheap, many of us still manage to fill our hard drives. Excepting the cases of multimedia collections of audio or video files, much of that space is consumed by files we either don't remember we ever made, or will never open again. On the laptops, where space is precious, so too will be the objects and entries that remain in the journal years down the road. The temporary, the experimental, the duplicate, and the unwanted files will naturally fall off the bottom, maintaining a browsable history of those that remain important to the children.

Journal Entries

Implicit

Implicit journal entries will be the most common. These appear as the result of many kinds of a child's interactions with her machine, but most commonly when engaging in an activity. Other implicit entries might appear when she takes a photo, or receives a note from a friend, or downloads a file from the Web. In all of these cases, the journal entry itself has a basic format which conveys important information about the event which created it. Most importantly, the associated Object - the photo, the message, the drawing, the story - becomes embedded within the entry. It also includes key metadata, such as its name, when it was made, and who collaborated on it.

The journal entry also provides some means to interact with it. For instance, each entry has a description field where a child can tag it with meaningful related words which will make searching for

it in the future a breeze. This field will automatically receive any tags that the activity itself associates with the entry. In addition to this tag field, several buttons will allow direct manipulation of the Object, making it possible to resume the activity, place the Object on the clipboard, send it to a friend, print it, or erase it, among others.

Note

In addition to implicit ones, children have the opportunity to create several special kinds of entries on their own. The first of these, the Note, has the simplest form. Taking a cue from a traditional journal, a Note entry simply provides a large text entry field. This freeform entry allows the children to write down short descriptions of their day to day experiences, just as one would within a real journal. Providing this layer of personalized entries further emphasizes the idea that the Journal really does provide more than a filesystem, as an actual record of events and interactions of the children with the laptop and with their peers.

In practice, children may also use this feature as a means of jotting down a note to themselves - a reminder. In these instances, a simple control within the entry will turn the note into a "to-do" note. As a to-do entry, it will have a checkbox indicating its completion status. By filtering the Journal to show only these entries, it doubles as a basic to-do list, providing another useful tool for learning organizational skills.

Clipping

Clippings serve a slightly different purpose in the journal. Similar in spirit to notes, a child can create a clipping from anywhere, or from within any activity on their laptop. As an extension of the copy to clipboard idea, clippings copy a selection - some text from a chat session with a friend, an image from a web page, etc. - directly to the journal for safekeeping. This provides a quick and easy way to keep a quick record of anything that you might want to keep around for future reference: a phone number, a link, a password, etc.

Event

Taking the temporal aspect of the Journal one step further, Events act like "future" journal entries. By specifying a name for the event, a brief description, and a time, these Journal entries serve as a basic planning system. A control within the entry also enables an audible alert, so that Events can act as alarms. Events also tie in closely with some implicit actions of the laptops. For instance, a child might want to go on a photo safari with her friend after school. While still in class, she sends him an invitation to join a photo capture activity, but schedules a time of 3:00. He then receives an invitation, as usual, but upon accepting it receives an Event entry in the journal, with a reference to the scheduled activity, instead of immediately entering it. When 3:00 arrives, both children receive notifications that their scheduled event is about to start, and join each other both physically outside and virtually in the referenced capture activity.

Progress Indicator

In many cases, entries will appear at one point in time but the desired result of the entry won't be immediate. This might occur, for instance, when downloading a file from the Web, receiving a file from a friend, restoring a file from the server, or saving a large Object such as video or audio data. All of these processes take some non-trivial length of time, and so when necessary, Journal entries will provide a progress indicator stage. When the download, transfer, or task begins an entry will be created in the journal to indicate that. This entry will include a progress bar with estimated time until completion; once completed, it will transition to a standard entry. This makes the Journal a consistent place to keep track of progress, and also provides an easy means to pause and resume transfers, which will prove extremely useful when in areas with intermittent connectivity.

The Power of Metadata

Despite the flatness of the Journal, finding past entries shouldn't prove difficult thanks to a tagging structure built from the ground up for the laptops. By associating relevant descriptive words with each journal entry, searching for an entry becomes as easy as describing it. These descriptions will manifest in two ways, tagging and metadata. The former provide a straightforward manner for the

children to describe and organize their stuff, while the latter provides a more technical means by which activities can associate relevant data and tags with all Journal entries they create.

Tagging

Tagging will become a fundamental process for all types of data and activities on the laptops. Fortunately, children have a natural inclination to describe their world and the things they see and do. This actually aids kids in learning, as they will enjoy describing the drawing they've made, the stories they've written, or the composition they produced, and can learn new vocabulary in doing so. Of course, the kid-like desire to describe things doesn't detract from the usefulness of this tag-based system as they grow older.

As such an integral part of the system, the tagging interface will be exposed in various places. Of course, as mentioned, each journal entry will have a field for tags. Likewise, each open activity instance will have a tag field adjacent to its name field, so that the act of naming a particular activity or Object becomes associated with describing it in their minds. Additionally, activities could offer specific places within the interface for tagging to occur, such as in the description field for a photo the child just took.

Metadata

Metadata adds an additional level of sophistication to the tagging model. Rather than thinking of this as data about data, consider it a means of tagging tags. Metadata on the laptops will be an extension of the basic tagging model where the tag itself consists of a key:value pair. Or, you could simply consider a tag to be a metadata pair with a null key. Whichever way you look at it, this categorization of tags has powerful implications when it comes to organizing and categorizing data.

The Journal itself assigns a variety of useful metadata tags to entries as they appear. These include the time of the entry, its sharing scope, who participated in the activity, its size, and more. The Journal will also keep track of other useful metadata, such as the number of times a child views a particular entry, the number of revisions an entry has gone through, etc. Likewise, activities will deal primarily with metadata rather than simple tags. This allows activities to define specific parameters, or keys, that make sense for the Objects they produce, and then assign values to those dynamically. In a music composition activity, for instance, potential keys might be beats per minute, the key the composition is written in, the length of the track, and the composer, among others. See the sorting section to fully understand the usefulness of this metadata within the Journal.

Of course, since tags and metadata both follow a very basic format, children can assign their own metadata associations with Journal entries once they have enough experience simply by typing key:value pairs into the description field.

Powerful Search, Filter & Sort

Searching

The search field provides the most direct means of locating a particular Journal entry, returning instant results as the search is typed, and offering auto-completion for popular tags. In order to find anything on their laptop, a child need merely describe it, since the tags she's associated with it already appear within its description field. Her searches also apply to the metadata associated with the entry by either the Journal or the activity that created it, making it even easier to find things.

For simplicity, the search field will employ OR logic to all terms entered, which ensures the least amount of confusion when used by children who don't yet understand boolean logic. As such, a search for "orange cat" will return a list of everything orange and also every cat. Of course, any entries tagged with both orange and with cat will match more strongly, and will automatically filter to the top of the results. However, in keeping with a primary goal of the laptops, this won't eliminate the possibility for more complex boolean searches. Full support for AND, OR, NOT, and parenthetical grouping of terms will be built into the search engine, providing advanced functionality for those who desire to enter more complex queries.

Since the laptops will find themselves in the hands of many children, additional modifications to the search algorithm will assist them as they grow. The youngest children who receive them will still be learning how to spell, and those that can may still require some time to learn typing skills. For these reasons, a fuzzy match algorithm will assist the children, returning some results even when the corresponding tags don't match what they typed exactly. This algorithm is adaptive, and so as they become more comfortable with their language and with using the technology, the extent of the fuzziness and therefore the number of fuzzy results returned will lessen, preventing false matches from aggravating more advanced users. Several other kinds of fuzziness could also be applied, though such possibilities are only speculation at this point. For instance, fuzzy matches based on thesaurus entries could turn up items tagged with "funny" even when the child searches for "humorous". Likewise, translation fuzziness could return an entry tagged with "cat", even though the child searched for "gato." These advanced fuzziness algorithms could prove invaluable in a laptop community that has been built with sharing and collaboration in mind.

Filtering

Support for basic filtering also exists within the journal. The search and filter functionality appear together in the toolbar, since searching could also be interpreted as filtering by tags. Additionally, their appearance together allows an easy method for the children to visually construct their query in a sentence-like format, with relevant parts of their query displayed as icons — just as those within the entries themselves — for visual reinforcement.

Several fundamental filters exist. First and foremost, there is an advanced date filter, which can only be expected in a Journal organized temporally by default. This control will present a timeline to the child, with visual indication over the length of the timeline of the number of entries present in the Journal from any given point in time. By expanding and contracting the selected area she can select anything from a single day to all time, and by sliding the selection through time, she can filter out all entries that don't lie within the specified range. Other basic filters include the activity that the entry represent, and the activity participants.

Other available filters allow children to locate specific kinds of entries. For instance, a child may want to view all entries that have been tagged by the Journal for possible removal when memory becomes low. They may also want to see all their notes, or to-dos, or events. They could also show only starred items, or in progress items, and more. The system will provide adequate flexibility for finding anything in the Journal nearly instantaneously.

Sorting

Whereas searching and filtering provide a means of defining what entries get shown in the list of results, sorting determines how those entries are organized. A unique approach to sorting on the laptops makes the metadata associated with entries even more valuable. The sort bar, which the child can expand in order to more precisely control their view of the journal, offers a popup menu from which a number of options such as date, title, activity, size, participants, and others may be chosen. In addition to this fixed list, a dynamic list of options also appears, providing a list of metadata keys that are present in the majority of the entries within the results list, the utility of which will become apparent below.

The true functionality of the system arises from "then by" sorting. When desired, a child can specify up to three levels of sorting hierarchy. This feature shouldn't be overlooked, since it serves as an extraordinarily powerful means of viewing and organizing data hierarchically, even when no hard hierarchy exists. In fact, when used to its full advantage this approach can be more useful than a hard hierarchy, since the order of the hierarchy can be adjusted dynamically to suit the child's needs at the time. And, in conjunction with the intelligently compiled list of metadata keys on which to sort, children can not only find what they're looking for, but can browse through their journal in any way that suits them. Consider, for instance, that a child filters her journal so that all of her music appears in the results. Since nearly every song Object in her Journal has metadata keys for artist, album, track, and year, she could sort by these keys to arrange her music collection for browsing. Sorting by artist, then album, then track she can obtain a traditional view of her music. In

order to view a discography for each artist, on the other hand, she could sort by artist, then by album, then by track. Or, to see a timeline of her music, she could sort by year, then by artist.

This powerful sorting method isn't necessarily limited solely to a bunch of songs, or pictures, or other specific type of data. Since many forms of metadata will apply across Object types, the possibilities are nearly limitless.

Implicit Versioning System

As mentioned before, the laptops automatically save, or "keep", objects in the Journal at regular intervals. This eliminates the need for the children to constantly worry about saving, and reduces the chances that an unexpected circumstance will cause data loss. These individual keep events are incremental, meaning that the changes within the file are kept in a nondestructive manner. Therefore, the Journal not only stores Objects as children create them, but also keeps track of the revision history for each one. This allows the Journal to function as a versioned filesystem.

The space limitations on the laptop cause some concern with the mention of revision history. However, the differences between revisions will often be small. Additionally, Objects with large revision histories provide one easy way for the journal to regain valuable space when memory becomes tight, since it can collapse the history, storing only every few automatic revisions in addition to those explicitly kept by the child.

Automatic Backup and Restore

Backup

In most locations where laptops deploy, a nearby school equipped with a server will provide additional functionality when children bring their laptops within range. This server has one major implication for the Journal: backup. Due to limited storage space on the laptops, children will have to choose to erase older entries to make way for new ones. The automated backup system will ensure that, even once their creations leave their own laptops, they will remain available on the school server. This process will happen in the background, fully automatically, anytime the child comes within range and bandwidth allows. Her Journal will keep track of which entries have and have not yet been backed up, taking this into account when recommending items for removal.

Restore

The backup service provided by the server will allow several types of restoration, based upon the child's needs.

Full restore functionality provides a fail-safe for the children. Though the ruggedness of the laptops should minimize the need for a full restore, any event which causes permanent loss of any or all data on a laptop can be recovered from nearly completely from the backup files. These files also provide a means of restoring the child's settings and data should they for any reason ever need a new laptop. In practice, this is a temporal restoration, recovering all files stored on the backup server within a given time period from the present.

The backup server will also provide partial restores, which allows children to select individual objects and entries to recover. Any recovered items will appear as a new Journal entry on the date of recovery. This form of restoration will occur with much greater frequency, purposed mainly to restore an accidentally deleted entry from a week or so ago, while flexible enough to restore any entry ever backed up on the server.

In addition to these hard restoration methods which physically copy the data back onto the children's laptops, the server will also provide soft restore functionality, allowing children to browse through the backups on the server directly from within their Journals. Since this browsing functionality will integrate cleanly with the entries stored on their own laptops, the children will be able to search, filter, sort, and view anything they ever entered without having to think about the technicalities of the data's physical location. Apart from a visual indication within the entry, the experience of browsing through backups will be seamless. Using the temporary restore method, children can browse through their past creations, much as we might browse through a photo album. They will

have full ability to resume any instance of an activity, to view its contents. A copy-on-write approach will be taken, so that if a child attempts to modify a temporarily restored item, it will behave identically to a partial restore for that Object, writing the modified revision into a new journal entry.

Global Search

Coming soon...

Activities

A New Model

We make a distinction between the typical single-application, multi-document model of computing and the Sugar full-screen activity interface, where each object (document) runs within its own instance—multiple instances of a given activity may run in parallel. Activity instances within Sugar provide a way to handle files as objects; each instance may represent a different group of collaborating individuals, and creating a new instance of the Draw activity implicitly creates a new drawing. "Open" and "Save" actions are relegated to a Journal interaction; In fact, we strengthen this by replacing the notion of "Saving" with the more general notion of "Keeping" things. To "open" a drawing you've kept, you simply [resume](#) it.

Starting Activities

Activities appear in the Actions section of the frame; starting an activity amounts to creating an active instance of it, represented in the activity ring. They can be started with a single click. An activity may also be directly manipulated; dragging an activity into the ring will also create a new active instance of it.

Visual cues differentiate between instances of an activity and the [activity icon](#) in the frame. Specifically, any activity installed on the system and appearing in the Actions edge is drawn as a white outline stroke, with no fill. Upon instantiation the icon receives a fill; both [stroke and fill colors](#) match the XO colors of the child who created it.

Private Activities

Newly created activity instances inherit the scope of the view in which they are created. This means that any activity started from the Home view begins as a private one by default. Children may later [share](#) private activities, opening them up to friends, classmates, group members, or anyone on the mesh through an [explicit invitation](#).

Shared Activities

Since newly created activities inherit the scope of the view, any activity started directly from the Friends Group view will be open for her friends to participate in. This applies to any group the child belongs to as well. [Implicit invitations](#) are sent to all of the members of the currently selected Group, alerting them of the activity. Likewise, any activity started from the (unfiltered) Mesh view will be open to everyone on the mesh, although invitations are not sent.

The views provide scope for instantiating activities. For finer granularity, the search (located in the Frame) provides an incremental filtering system that enables arbitrary selection of scope. As a query is entered into the search field, the view—Friends or Mesh—dynamically updates to reveal the matching selection. Matches remain in color, while those filtered out appear with a white outline. The filter terms apply parameters such as the names of activities, the types of activities, the names of individuals, and the interests of individuals. For instance, a child could search for anyone who likes games before starting a new game of Memory, or everyone in the same grade in a classroom setting, or a specific group of individuals by name. The results of the query become the

scope for any new activity instance, and all XOs within that scope receive implicit invitations when an activity begins. These groupings may be saved as groups for future use.

Once a shared activity begins, the child who initiated the activity is taken into Activity view. Others who received invitations won't join the activity until they accept the invitation; white outline placeholders for their XO icons appear in the People section of the frame to indicate their potential arrival. If they accept an invitation, their XO fills with their colors; if they decline the outline disappears.

Joining Activities

Children will often find themselves joining activities already started by others. Activities can be discovered through search; searches may specify an activity name, an activity type, interests of individuals, and names of individuals. For instance, one could search for all activities that relate to music, or all activities that have participants who like camping, or all the active chat activities, or a few specific people by name. Once an appropriate activity is found, a single click on the activity icon will engage it.

Sharing Activities

Activities may begin as private, or restricted to a small group of individuals. There may be occasion to open up activities to a broader scope. For instance, a class may break into groups to work on a project within private group activities. At the end of the session, all groups may wish to open up their activities to the rest of the class for discussion and critique. Through selection in the activities contextual rollover, one may set the scope of children who may join an activity to one of Private, Mesh, or any specific Group to which she belongs, including her class, her friends, and potentially others.

A child may lock activities in a similar manner, tightening an activity's scope. Participants must leave on their own volition or at the request of others within the activity before locking it.

Switching Activities

The activity ring indicates the activities currently running on the laptop. From the Home view, a single click on any activity in the ring will select it as the active activity, automatically transitioning back to its Activity view. Keyboard shortcuts enable quick transitions among open activities.

Ending Activities

Ending an activity happens as easily as starting one. To complete the metaphor, dragging an activity out of the ring will end it. Selecting the End action in the activities contextual rollover will do likewise. Note that ending a shared activity—even one *you* started—does not necessarily "close" it. An activity instance remains active on the mesh as long as one or more individuals remain as participants.

Resuming Activities

In lieu of an "Open" command, one may simply resume an activity. If a drawing resides in the Journal, then resuming it will automatically restart the Draw activity, allowing modifications to that drawing. Due to the emphasis on collaborative activities, special consideration has to be given when resuming them; An activity fingerprint identifies a particular instance on the mesh. Resuming an activity implicitly invites all others who at one point participated in its creation that also remain within its currently specified scope. Additionally, cases may arise when an activity being resumed is already active on the mesh. In such cases, the child will automatically join the already active instance.

Activity Robustness

All activities designed for the laptop should place a strong emphasis on robustness. Two essential robustness considerations are input and network.

Invitations

Invitations perform an essential functionality in a computing environment that so strongly emphasizes collaborative learning and creation. For this reason, two forms of invitations are present in the OS: explicit and implicit.

Explicit Invitations

Explicit invitations are used to invite specific individuals into already active activities. The ability to send explicit invitations to others serves particular use when in a private activity, be it a private group or a solitary one. In these cases, an explicit invitation can extend the group by including one or more specific individuals, without opening up the activity to a broader scope.

A child may initiate an explicit invitation either from within the activity itself or by identifying an individual or group in either the Groups or Neighborhood views.

Implicit Invitations

Implicit invitations do not require specific action on the part of the child. These invitations go to the appropriate individuals whenever actions suggest it, such as when starting an activity from the Groups or Neighborhood views. All individuals within the activity's scope receive implicit invitations to join. When an activity is resumed, those who participated previously receive an invitation.

Receiving Invitations

Incoming invitations appear within the [Actions](#) section of the Frame, adjacent to the installed activities; they are rendered in the color of the inviter. Rollover reveals both the name of the inviter as well as the name and type of the activity. On extended rollover, the options to accept and decline appear. There is an optional message back to the inviter upon declining an invitation.

Notifications

Notifications behave similarly to Invitations; they also appear in the [Actions](#) edge of the frame. However, unlike invitations, which are sent from people on the mesh, Notifications come from activities or directly from the system. As new notifications come in, they form a queue, with the most recent in the lower left-hand corner for quick access.

Sticky Notifications

By default, notifications will remain in the frame until the child acknowledges them.

Transient Notifications

Transient notifications alert a child when they arrive, but as they contain information that has a limited lifetime, they expire. Thus Activities may specify timeouts on notifications, after which they will automatically disappear.

The Activity Bundle

Activities will exist in the form of bundles. These bundles will manifest as groups of related files—source code, images, documentation, etc—that compose a given activity. As self-contained modules, the distribution and installation of an activity distills to a simple transfer of the activity bundle to a laptop. Properties stored within a bundle provide information about its version and its creator(s).

API Reference

[Activity Bundle technical specifications](#)

Bundle Types

OLPC will support a signed "official" bundle type. Signed bundles have been tested and verified by an authority such as laptop.org or any other organization through which children obtain bundles in some official capacity, such as a country's official repository. This system may support a trickle-up metaphor through which locally signed bundles propagate upward to higher authorities, allowing wider distribution of newly created activities and content to other regions and countries.

Personal bundles, on the other hand, have been created or modified by an individual among the laptop community. A personal bundle isn't signed or verified by an official source; instead, it is signed or watermarked with the identity of the individual who modified it. This watermark remains attached to the bundle throughout its lifetime. As others modify or change it, their own watermark should be appended to the bundle. This gives a personal bundle some sense of origin and a means through which it is possible to give credit or responsibility.

Bundle Versions

Bundles always automatically update to the latest officially signed version present within the laptop's network. If a child's friend has a more recent version of a signed bundle, Sugar will download that newer version and update the laptop automatically. This requires bundles to communicate a unique bundle identifier and version, as well as their signature if they have one.

Naming Activities

OLPC aims to provide a platform which encourages expression through creation. In support of this idea, activities — not applications — provide the main tools through which objects are created. Whenever possible, activities should be named with descriptive verbs, or suitable pseudo-verbs, in order to emphasize their function as *things you do*.

Activities as Verbs

Activities *are* verbs. As such, the phrase "<activity> with my friends" should make sense. For instance "draw with my friends," "browse with my friends", "chat with my friends" and "edit text with my friends" all make much more sense than "text editor with my friends." Similarly, "Tam-tam with my friends" reads as an action, even though you may have never heard "tam-tam" before. Treating the activity as an action (verb) and not as a thing (noun) maintains the interaction model that the laptop tries to embody.

Meaningful Naming

Of course, we don't mean to impose arbitrary limits on the types and number of activities that the platform has the potential to support. Just because there is a "Draw" activity doesn't mean that one must either find a synonym or come up with a different activity. (However, note that the former can be a very reasonable approach, as a synonym might actually have subtly different connotations which better support the concept of the activity. For instance, drawing and painting typically imply two very different types of media, dry and wet respectively. Much is gained when these types of differences are reflected in the nature of the activity, and are not simply arbitrary.) In some languages, [verbification](#) has become common practice in speech. Many words function as both nouns and as verbs, indicating the action of creation and the resulting product of that action; additionally, many nouns can also function as verbs. For instance, if you speak English, you've probably "Googled" something in the past few days. Many nouns, not just proper ones, can be used in a similar manner.

Additionally, while straightforward names can simplify the interface and provide a means of understanding an activity before entering it, compound names may also be used. Providing a modifier, such as an adjective, can personalize the activity and provide that extra bit of information which differentiates it from similar ones. For instance: "Finger Paint." However, please refrain from resorting to simple [one-upmanship](#) in the form of "Super Sketch" or "Ultra Paint," especially if another activity already uses the modified base. Such names only serve to indicate superiority, and don't provide any useful feedback about the particular activity which makes it unique or useful. Providing

a meaningful name goes a long way to making the activity intuitive and enticing to the children using it.

Credit

Finally, please avoid integrating the name of yourself or of your company into the name of your activity. As an open-source initiative we fully believe in giving due credit, but the name of your activity doesn't provide the appropriate place for accreditation.

Activity Tags

Tags provide additional information about the context of a specific activity, enabling powerful searching on the Mesh for generalizations or categories of activities. For instance, searching for "game" should return the "Memory," "Chess," and "Tic-Tac-Toe" activities. Likewise, searching for "drawing" should return any activities that relate to drawing, painting, sketching, etc.

Obtaining Activity Bundles

Officially signed bundles should spread freely across the mesh Neighborhood; their information and the bundles themselves should be readily available to anyone within communication range. Installation and updates should occur implicitly.

While personal bundles are slightly more restricted, current thinking would limit distribution of personal bundles amongst a child's friends only. This should help limit the destructive power of a malicious bundle from spreading across the Neighborhood, yet still allow people to open up their bundle source code, improve it and share it explicitly.

```
We may wish to allow distribution to any Group rather than just to Friends, so that if a child wrote an activity that is useful for her whole class, she does not have to add everyone to her Friend group, breaking the metaphor.
```

Implicit Bundle Sharing

Implicit bundle sharing will automatically update signed bundles on a child's machine when the network allows. If a child finds an interesting activity running on the mesh Neighborhood, she will implicitly download and install the activity on her own machine when she joins that activity. Additionally, this provides a means of obtaining completely new bundles, since she doesn't necessarily need to have an older version of the bundle installed prior to joining. Of course, since there will likely be some download time before the activity can begin, a visual indication of the progress will appear during launch.

In cases where a child joins a group running an older version of an activity she has a newer version of, the same will happen. Her laptop will silently download the older version of the activity so that when she joins, her active instance is service and communication level compatible. However, in such instances the old version will not overwrite the newer version, and will instead remain a transparent detail for compatibility reasons. The newer will remain present on her machine, so that future activities which she initiates begin with the new version, ultimately encouraging the spread of newer bundles.

```
We might need some kind of warning when joining an activity on the mesh whose bundle is not signed...
```

Explicit Bundle Sharing

In the case of personal bundles, explicit sharing will be required. This results from the fact that many children may ultimately edit and redistribute new and altered bundle versions of a variety of software; automatic distribution of such modifications is neither secure nor efficient.

In these cases, activities may be posted to private Bulletin Boards, or distributed directly to a child's friends through the drag and drop metaphors used elsewhere in the interface.

Where Are Bundles Stored?

The Journal keeps a record of all bundles on the laptop. Installing a bundle creates an entry that indicates who the child downloaded the bundle from and its version. If she installed the bundle through the joining of an activity, the activity entry in the journal will reference the newly updated bundle. Of course, once stored within the journal, the Bundle will be available for activation within the Actions section of the Frame.

Removing Bundles

The journal entry for an activity bundle also allows for its removal; it is deleted in the same way one would remove any other item from the Journal.

Security

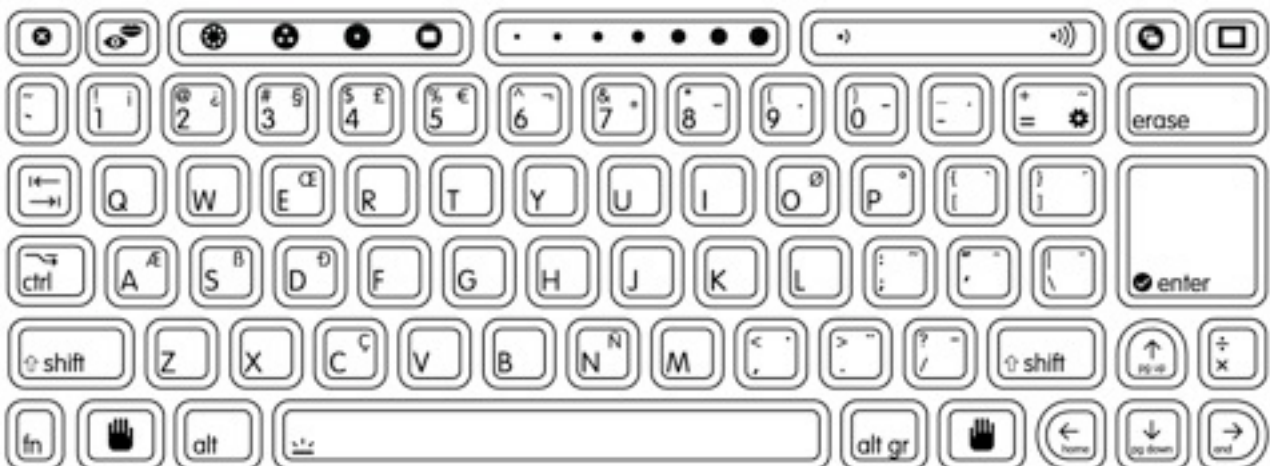
Coming soon...

see [olpc:Bitfrost](#) for now.

The Sugar Interface

Input Systems

Keyboard





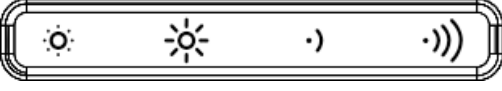







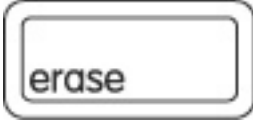


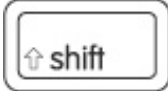




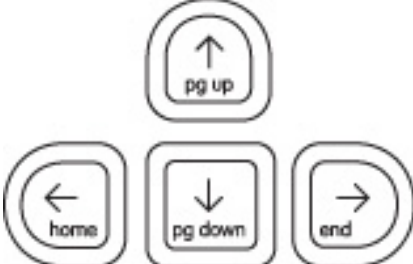
The basic laptop keyboard layout.

Localized Keyboard Layouts

- [Arabic](#)
- [Spanish \(Argentinian\)](#)
- [Portuguese \(Brazilian\)](#)
- [Nigerian](#)
- [Thai](#)
- [US International](#)

Description of Keys

Key	Function
View Keys	
	Transitions among Neighborhood , Groups , Home , and Activity views.
	Toggles visibility of the Bulletin Board for the current view
	Toggles visibility of the Frame
Hardware Controls	
	This key will invoke a Journal search.
	This slider key functions as controls for both the display backlight (left two buttons) and the speaker volume (right two buttons).
Special Functions	
	The View Source key (gear) peels away the activity layer, allowing children to view the underlying source code. It is accessed in combination with the Fn and space keys.
	Grab Keys are for panning/scrolling; they are used in conjunction with the touchpad.
	The middle of the three large " slider " keys at the top center of the keyboard is available for use by activities. The slider can be mapped directly to a control in software.
Editing Keys	
	We've enlarged the Enter Key, and given it a visual indicator that maps directly to the graphics used in the UI. All instances of the confirm and cancel icons within the interface will be selectable directly via the Enter and Escape Keys – a relationship strengthened by this visual mapping.
	The Escape Key has a visual indicator that maps directly to the screen graphics, complementing the Enter Key.

	<p>We've replaced the Backspace and Delete Keys with an Erase Key. This new term more accurately describes its functionality both for erasing a few characters of text, but also for erasing drawings, sounds, and other objects. (Fn-Erase deletes beneath the cursor.)</p>
	<p>The Tab Key differs little from those on modern keyboards. Shift-Tab functions as a reverse tab, as visually indicated on the key.</p>
Modifier Keys	
	<p>The Control Key is the primary modifier for keyboard shortcuts on the laptops. Note that the control key takes the place of the nominally useful caps lock key on the OLPC keyboards. The removal of caps lock was a design decision on the part of the OLPC team, however placement of the control key in its location followed naturally, since this was its original placement prior to the PS/2, and is still widely accepted among many communities.</p>
	<p>The Shift Key is used as a modifier for typing capital letters and other "upper" characters.</p>
	<p>The Alt Key is a multipurpose modifier. The Alt Key is typically used to provide a related but alternative functionality (often increased scope) to a Control Key. For example, Ctrl-C is copy; Alt-C is copy and erase.</p>
	<p>The Alternate Graphics Key (alt gr) is used to select the alternate (additional) characters printed on the right half of the key caps. A common use is the Unicode combining characters used for inserting accent characters. (On some keyboards, there are two separate sets of symbols printed, e.g., Thai, Arabic, Urdu, Ethiopic, etc. In these cases, the Language Key, described below, switches the entire keyboard between languages.)</p>
	<p>The Language Key, found on keyboards that have both full Latin and a second alphabet, e.g., Arabic, Thai, Urdu, Ethiopic, etc., is used to toggle the entire keyboard between alphabets.</p>
	<p>Fn is the Function Key. It is used to further modify keys; it is used to access the View Source Key; it modifies the arrow keys to home, end, page up, and page down; and it is used to enable the analog slider controls.</p>
Navigation Keys	
	<p>The standard Arrow Keys – up, down, left, and right – also operate as page up, page down, home, and end respectively when used in conjunction with the Fn Key.</p>

Softkey Sliders

The slider keys have two modes: "digital" and "analog." In digital mode, the discrete functions printed on the key caps are accessed, four per key. In analog mode, accessed with the Fn key, intermediate key codes are enabled—there are seven positions along the slider; intermediary positions are interpolated in software, turning each of the keys into a 13-position slider.

Keyboard Shortcuts

For the purposes of development, you may want to review the detailed specifications for keys and their codes on the [Keyboard Layout](#) page. For a complete list of agreed upon keyboard shortcuts in the Sugar environment, at both system and activity levels, please refer to the [olpc:Keyboard shortcuts](#) page. Following is a high level description of the types of shortcuts the available keys should pertain to.

- **◆ CTRL** (U+25C6) will be the main modifier key. It will be used to define "base" shortcuts. For instance, **◆A** will "select all" in a text editor.
- **◇ ALT** (U+25C7) should be used for optional modifications (or ALTERNATES) to base shortcuts. For instance, **◆S** will perform a standard "keep" and **◇S** could be "keep as...". **◆V** is paste, **◇V** is "paste and remove from clipboard."
- **⤴ SHIFT** (U+21E7) can work in two ways. Its primary use would be as an inversion modifier, such that it inverts the meaning of the base shortcut. For instance, **◆Z** is undo and **⤴◆Z** is redo. **◆TAB** is next activity, **⤴◆TAB** is previous activity. SHIFT can also be used to create a second set of "base" shortcuts which are less often used.

The FN key is reserved solely for system level operations, and generally for those that map to functions which are printed on the keyboard itself.

Trackpad

The laptops employ a capacitive (finger controlled) trackpad.

Only the center region of the trackpad has capacitance, responding to a finger.

Trackpad as Mouse

The use of the finger on the central trackpad area serves as the primary input device for pointing. Though external [USB mice](#) will work seamlessly with the laptops, their availability will be limited, and activity designers should not expect that children will have access to them. This means that a certain lack of precision can be expected when moving the cursor about the screen, and activities should not require extremely precise motion. Excessively small controls should also be avoided for similar reasons; details on how to design interface elements reside in the [Controls](#) section.

In addition, the laptops have two buttons positioned beneath the trackpad for input. The left button is the primary button with which elements of the interface are selected, pressed, or activated. The right button has secondary functionality. Typically, the right mouse button invokes contextual menus, the content of which pertains directly to the interface element the mouse is positioned over.

Trackpad as Graphics Tablet

No stylus support is currently planned for the XO.

Microphone and Speakers

The laptop has a built-in microphone and stereo speakers to allow for voice communication and recording. You may integrate audio functionality directly into your activities by requesting access to this hardware in the Functional Manifest. There are also an external microphone and speaker jacks.

Need a section on using sounds in activities; particularly in the background...

Camera

The laptops have built-in cameras to allow for still photography and video recording. You may integrate camera functionality directly into your activities by requesting access to this hardware in the Functional Manifest.

"Hand-held" Mode

The laptops feature a hand-held mode of operation in which the screen swivels around 180 degrees and folds flat, similar to a tablet PC. In this mode, the screen covers the keyboard and trackpad; however, the microphone and camera, mounted within the display bezel, remain available for use. Additionally, bezel-mounted controls provide auxiliary input suitable for the activities that Hand-held mode is designed to support: reading an eBook, playing games, etc.

Energy Saving Benefits

The laptop is engineered for extreme operating efficiency—a goal furthered by Hand-held mode. The CPU can be suspended while still displaying on-screen graphics. While reading an eBook, since the screen need only be updated when a page is changed, the time spent reading any given page requires no use of the CPU. The screen can run in reflective (daylight) mode—with the backlight off—for additional energy savings. These factors combine to create an extremely low-power, energy-efficient machine; Hand-held mode provides a usage scenario where maximal energy savings can be attained.

Implementing Hand-held Controls

Unlike a typical tablet PC, the OLPC laptop does not have a touch-sensitive screen. Primary user input comes from two bezel-mounted button sets: the D-pad (directional pad), which has 8 directions of articulation; and the button controller, which houses 4 discrete buttons (labeled ○, ×, △, □, on the B1 machine).

The Directional Buttons

The D-pad should *not* be, in general, used to move a cursor around the screen— in fact, the cursor will hide by default in Hand-held mode. Instead, it should be used for more discrete operations, such as flipping through pages, scrolling a view, or jumping to focusable elements on screen. When an interface necessitates focusable elements, these should be visually apparent and arranged in a natural order. In most cases, "natural order" will mean scan-line order, or the way in which one reads a page of text, but this may adapt to suit the needs of the activity. For instance, some activities may opt to scan first by column, then by row; some may use a clockwise ordering some may even zig-zag across the screen. All of these arrangements are acceptable as long as their orderings logically follow from one to the next according to the visuals provided on screen.

Specific activities may in fact benefit from a more traditional cursor. Some games, for instance, may require one. To support these cases, the cursor may be explicitly shown. However, these instances should be carefully considered, since in many cases a cursor will provide a simple yet inefficient solution to a problem for which a better one exists.

The Controller Buttons

Generally speaking, the controller buttons can act either as standard event triggers, or as modifier buttons to the target of the D-pad controls. A common use for standard buttons is as select and cancel buttons. In such instances, the ○ button should always represent confirmation, selection, or forward progress, while the × button represents cancel, escape, or backward progress. Adhering to these guidelines will make navigation of Hand-held interfaces consistent.

When used as a modifier, the visuals on screen should clearly indicate which of the directions—up, down, left, right—perform actions, and those actions likewise should be clearly indicated. For instance, in the eBook, holding down △ displays an overlay listing the book's chapters, and the up/down arrows will have focus within this list while the modifier key remains pressed. The currently

selected chapter appears in the center of the screen, and up and down arrows above and below the selected chapter clearly indicate how to scroll through the list. When activities implement a combination of both standard and modifier buttons, we encourage × and ○ for standard, and △ and □ for modifiers, since the former two are easier to hit with natural finger placement.

Built-in Hardware in Hand-held Mode

Both the camera and microphone reside within the display bezel, and as such remain available for activities to use within Hand-held mode. The important trade off to consider before using the camera or microphone is that of energy efficiency: while the laptop conserves energy in Hand-held mode, continued use of either of these two devices requires constant CPU usage, virtually eliminating the benefits. Therefore, do not simply integrate these hardware components unless they provide a fundamental service to your activity—but don't let this deter you from doing so where appropriate.

Screen Rotation

While in Hand-held mode, the laptops support screen rotation; by pressing a small button on the bezel of the display, the interface will rotate 90 degrees to provide a portrait layout of the currently active activity. Just as any activity can implement Hand-held mode, those which can benefit from a vertical aspect ratio may also implement this feature, and we encourage developers to take advantage of this functionality. The Read activity serves as a prime example of the usefulness of such a feature, since a vertical layout is well suited to displaying a single page from a book. This is just the type of activity one might want to do in Hand-held mode, and by providing two orientations a greater number of use cases can be covered.

In the current revisions of the laptops, it's important to note that the buttons for interacting with Hand-held mode are slightly less accessible when the laptops are held vertically. For this reason, activities that require heavy or frequent use might not be best suited for this mode. However, OLPC is working hard on introducing touch-screen technology in the near future, which will nearly eliminate the dependency on the physical buttons, expanding the possibilities as every activity can take advantage of screen rotation. Therefore, even if screen rotation doesn't make sense for the first version of your activity, please construct your interface in such a way as to allow future adaptation to this new and potentially useful functionality.

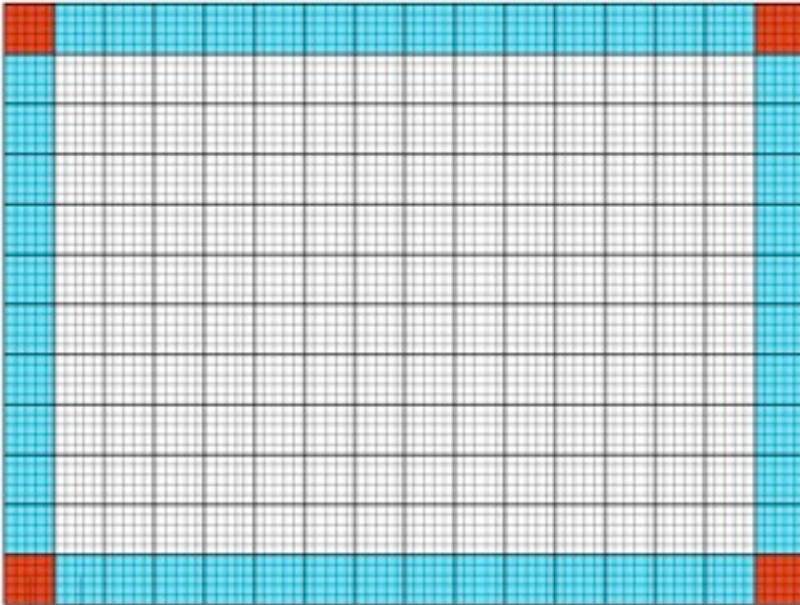
Layout Guidelines

The Grid System

In keeping with the simple, flat visual style of the Sugar interface, we have designed all of the interface elements on a straightforward grid system. The system functions like a basic floorplan which allows interface elements to tile neatly within its boundaries. The grid consists of a 16x12 array of square tiles.

The diagram below shows the screen dissected into cells. Furthermore, the cells highlighted in blue indicate the areas suitable for activity toolbars. Note the exclusion of the corners, which are reserved for the hot corners which invoke the frame. In order to prevent accidental invocation, no buttons should be placed in these locations. Though all edges of the screen are suitable for placing toolbars, we recommend that the top and left edges of the screen serve as the primary location for them unless circumstances specifically suggest an alternative. Conforming to these guidelines will both make the variety of activities supported on the laptop feel more consistent, and also reserves the right and bottom edges for scroll bars, making navigation (when required) within the primary pane much simpler. Though the grab key provides the primary means for panning, preserving these edges for scroll bars even as visual indicators will increase usability.

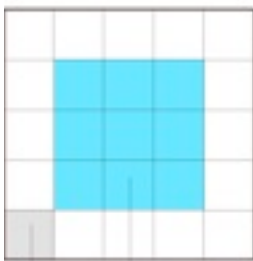
Screen Grid (16x12 grid cells)



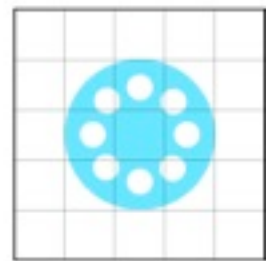
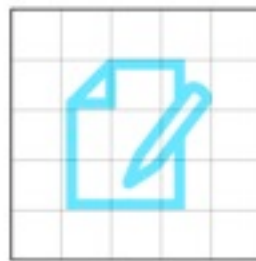
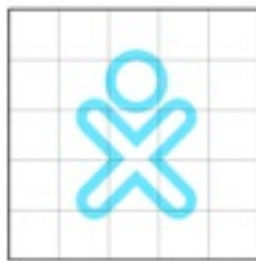
Frame/Toolbar area
Hot Corner area

Considering the screen at its maximum resolution of 1200x900 pixels in luminance mode, each square tile is 75x75 pixels in size. Each cell is comprised of a 5x5 array of 15 pixel subcells. These subcells provide layout guidelines at a finer level of detail. In general, the 3x3 subcell interior region is icon-safe.

Grid Cell (75x75 pixels)



Icon-safe area
(45x45 pixels)
Subcell
(15x15 pixels)



Icons

Categories of Icons

The XO



The icons which represent People have special status on the laptops. Referred to generally as the XOs, they represent the children and their laptops on the mesh Neighborhood, and furthermore represent the OLPC project and its goals to place a laptop in the hands of every child. Each child will select a stroke and fill color for their XO, and their chosen colors will then apply to the icons of any Activities or Objects they create.

Activity Icons

Object Icons



Action Icons

Active vs. Inactive Icons

Many instances may arise in which some elements of the interface are inactive. Sugar specifies a consistent visual style to represent the concepts of absence and inactivity. Inactive elements are buttons that are not currently actionable, or controls that are temporarily disabled. Absent elements are object icons that represent people or things which aren't actually present at the moment; for instance, an incomplete download, or an invited friend who hasn't yet joined the activity.

Generally, interfaces represent such inactivity through grayed out imagery. Of course, since the laptop also operates in grayscale mode, such a color distinction *must not* be used under any circumstances. Instead, Sugar will take advantage of the vector graphics used for rendering objects and buttons by rendering inactive ones as a white outlined stroke, with no fill color.

Active

Inactive

Icon Design Guidelines

Icon Format

All icons designed for use in Sugar must be provided in [SVG format](#). Since all icons exist as vectors, dynamic scaling and coloring of the icons occurs without any degradation. This allows variably sized representations of particular icons to exist depending on context in the interface. Additionally, this provides support for dynamic coloring of activity and object icons based upon a child's chosen XO colors.

Icon Sizes



Icons should be developed and saved at Standard (S) size, though their actual size and appearance in the interface may change dynamically. When developed at standard size, icons should fit loosely within the 3 x 3 icon-safe subcell of a standard 75px grid cell, as specified in the [layout section](#).

Notice that when the interface scales your icons, strokes do not necessarily scale proportionally to the overall icon size. This ensures that the stroke weight remains visible enough at all sizes to convey its weight and color, but it may also limit the granularity with which you use strokes, which could begin to blend together at smaller sizes. The following chart relates the various icon sizes to their corresponding

scale factors and stroke weights. We strongly suggest that you try rendering your icons at XS, S, and M sizes in order to tweak their appearance for optimal legibility.

Icon Size Comparison Chart

Icon Size	Scaling Factor	Stroke Weight
XS	0.5	2.25px
S	1.0	3.5px
M	1.5	4.0px
L	2.0	4.5px
XL	2.75	6.0px

Strokes & Fills

All icons render in two colors: stroke and fill. The actual stroke and fill colors that an icon renders in are determined by the children, since they correspond to the colors they have chosen for their XOs. As such, the colors in which you choose to save your icon are arbitrary. However, note that any fills that have the same color as your strokes will dynamically take on their color when rendered.

All strokes within an "S" activity icon must have a line weight of 3.5px. All icons should have a primary fill which represents its overall shape. In addition, any number of supplemental strokes and fills may be used; not all strokes within an icon must have fills, and not all fills must have strokes.

More detailed instructions for making icons can be found [here](#).

API Reference

Module: [sugar.shell.view.stylesheet](#)

Colors

Imbuing Color with Meaning

Sugar treats color differently from the typical UI: colors used in the interface represent the *individuals* who are interacting within the mesh, not the activities or objects they are using. Children personalize their laptops and their presence on the mesh by selecting a dual-tone color scheme. All of the activities, objects, and comments belonging to a child take on her own colors—even when they appear on the laptops of other children on the mesh—forming a visual identity that supplements her name and attributes.

This color treatment extends even within activities. For instance, in the Web activity a link-sharing feature encourages children to browse the web in groups, sharing interesting or useful pages with each other. Each URL object posted for the others to view appears in the colors of the child who posted the link. Similarly, chat bubbles on the Bulletin Board take on the children's colors. Likewise, any object, text, or other interface element within your activities that corresponds to a particular child should be rendered in this manner. When the display runs in grayscale mode, this colored visual identity is less apparent. However, significant differences in value, according to the Munsell System, ensure that the XOs retain a level of visual distinction even in the absence of color.

To maintain a degree of purity to this system, interface elements, buttons, and other icons that belong solely to the activity and not to any particular child should remain in grayscale to the extent possible. While removing color as a primary visual clue may seem counter-intuitive, it does encourage the icon's form to clearly indicate its function. Since the laptops will also run in grayscale mode, clearly distinct shapes become essential in the absence of chroma information, and so limiting activity icons to grayscale by default ensures compatibility in both modes. Additionally, keep in mind that the traditional method of "graying-out" inactive buttons and controls simply will not function on the laptops and must be avoided. Instead, please adhere to the guidelines for [inactive icons](#).

See [OLPC:XO colors](#) for the full list of defined and approved XO colors.

Contrast in the Munsell Colorspace

The basic color scheme for the laptop is constrained by the need to work in both color (backlight mode) and grayscale (reflective mode); thus we have chosen guidelines that ensure at least some achromatic contrast under all conditions. Further, sustained legibility of text is accomplished by a combination of colors whose achromatic contrast is large and whose chromatic energy is of low to moderate level. For this reason, we are striving for achromatic contrast of at least two [OLPC:Munsell](#) value steps.

The default value for the Frame is N2.5; the default value for the background is N9. Therefore, to maintain sufficient contrast, the line values for icons that appear on both the Frame and the background should range between N5 and N7. The interior fill of those icons should maintain achromatic contrast with the line value, e.g., the fill color for an icon with a line value of N5 should be either $\leq N3$ or $\geq N7$.

Munsell Value Steps

Color	Line value 5
N10	delta 5 value steps
N9	delta 4 value steps
N8	delta 3 value steps
N7	delta 2 value steps
N6	delta 1 value steps
N5	
N4	delta 1 value steps
N3	delta 2 value steps
N2	delta 3 value steps
N1	delta 4 value steps
N0	delta 5 value steps

Text Against Default Laptop Colors

Frame (N2.5)	Background (N9)
delta 7.5 value steps	delta 1 value step
delta 6.5 value steps	
delta 5.5 value steps	delta 1 value step
delta 4.5 value steps	delta 2 value step
delta 3.5 value steps	delta 3 value step
delta 7.5 value steps	delta 4 value step
delta 1.5 value steps	delta 5 value step
delta 0.5 value steps	delta 6 value step
	delta 7 value step
delta 1.5 value steps	delta 8 value step
delta 2.5 value steps	delta 9 value step

Fonts

[DejaVu LGC Sans](#)

Sizes

The font used in Sugar menus

The OLPC display is 200DPI; therefore one point—1/72 inch—is just less than 3 pixels (2.78 pixels). We are settling on a default font size of 7pts. for the Sugar UI (using [DejaVu LGC Sans](#)). It is quite legible. This translates to a font size of



approximately 19.45 pt in Adobe Illustrator, which bases its units on the traditional 72DPI display. For the purposes of preparing activity GUI mockups, you must always remember to make the conversion to the laptops' display resolution by multiplying by a constant factor of 0.36.

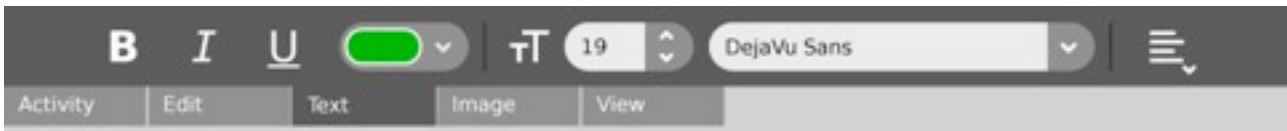
We will be looking at other faces, e.g., Arabic and Thai, and also looking into a large-type version of the interface for the younger children.

Readability

Due to the unique design of the OLPC display, particular techniques for text rendering will provide much better results than others. The dual mode display has a resolution of 1200x900 (200 dpi) in luminance mode, but only ~800x600 (133 dpi) in chrominance mode. Therefore, unless they are sufficiently large, fonts rendered with luminance and no chroma will appear sharper and more readable.

Additionally, the display has higher resolution in black pixels than in white pixels. This results from the fact that each pixel has a color part which contains either red, green, or blue information. In order to create white, red, green and blue parts must all work together; when off, each of the color parts is black on its own. Furthermore, a grey background limits the readability of the display in sunlight. Therefore, we recommend the use of black text on a white background for best readability of fine text, and color on white for larger print.

Toolbars



See [Toolbars](#) for new design images.

Every activity will have a "Toolbox" at the top edge of the screen. The Toolbox consists of a set of (at least one) toolbars, individually selectable via the tabs beneath them. Placement of the tabs beneath the toolbars themselves makes selection of tools and buttons within the toolbars easier according to Fitts' Law, since they will remain against the screen edges where they are "un-missable." Though this makes the tabs slightly more difficult to activate, we anticipate the frequency with which these toolbars require explicit switching to be minimal, specifically due to their contextual nature as described below.

Grouping by Context

Each toolbar will contain a logically grouped set of buttons and controls, as the name on its corresponding tab suggests. Each of these sets will represent a distinct editing or control context. For instance, the Write activity contains individual toolbars for Text, Images, and Tables. Each of these contains a set of controls relevant to the context they relate to. In order to streamline workflow, activities may initiate a toolbar-switch when the current editing context changes, automatically selecting the toolbar for the newly selected context. Clicking on an image within a report will automatically focus the Image Toolbar, revealing the associated controls; Clicking back within the body text will automatically re-focus the Text Toolbar. In this fashion, the controls visible on screen always remain relevant to the selection, virtually eliminating the need to select most toolbars explicitly, except perhaps to locate the "insert" button for an element for which no context currently exists.

Of course, a one to one relationship may not always exist. For instance, a selection may include both text and image, or perhaps for some reason focus isn't within any context at all. In these cases, it is up to the activity to decide the appropriate behavior. One suggested fallback is to switch automatically to the Edit Toolbar, since copy and paste, as well as most other editing commands, often apply across contexts.

Standard Toolbars

As mentioned above, every activity will have at least one toolbar within the toolbox: The Activity Toolbar. This toolbar will provide core activity functionality and will automatically be included in the Toolbox. Though sugar requires no other toolbars, it does provide a short list of potentially common

ones along with suggested names and controls for consistency across Activities. These are suggestions, not rules, and as a developer you should feel free to ignore the suggestions when you find a compelling case to do so. The following table suggests a standard ordering for some common toolbars. Note that the arrows indicate the relative position of the tabs, where those with double arrows can be interspersed with custom elements as long as their relative order remains. In the examples which follow for controls within each toolbar, the arrows should be taken to indicate alignment within the toolbar, double arrows indicate control regions which expand to fill the remaining space.

< Activity	< Edit	< Text >	< Image >	[Custom Toolbars]	Format >	View >
------------	--------	----------	-----------	---------------------	----------	--------

Activity Toolbar

Always the first of the tabs, the Activity toolbar will remain consistent in every activity, providing a place to name and tag the instance, set preferences, share within the mesh, or stop (close) the activity, among others. This toolbar will always have focus when a new activity instance is created, encouraging the children to provide a name and any related tags. An API will allow developers to associate various palettes with some of the buttons which reside in the Activity toolbar, such as preferences.

Edit Toolbar

Though not automatically included within the Toolbox, we anticipate nearly every activity will have an Edit Toolbar, since nearly every activity should at least allow copying if not pasting as well. Likewise, we are strongly encouraging every activity to support complete Undo/Redo functionality, which should also reside within the Edit Toolbar. Finally, the edit toolbar will also provide a common interface for performing find operations on any text within the activity. Of course, activities should only include those functions which pertain to them, and additional editing tools may be added to the toolbar as necessary.

< Undo	< Redo	< Copy	< Paste	[Custom Controls]	Find >
--------	--------	--------	---------	---------------------	--------

Undo/Redo: The undo/redo commands have extremely high importance on the laptops, since their presence encourages creative exploration without the fear of unrecoverable changes. They should function in a manner chosen by the activity, and although that manner should reflect our current expectations, the collaborative nature of most activities complicates the matter to some extent. A broad approach to managing collaborative undos requires a general notion of collisions between editing events. The [AbiCollab tools](#) which make the Write activity possible define this idea in detail in relation to text-based editing. The overall concept applies generally: For instance, a collision in a drawing activity could mean the collision of the bounding boxes of two drawn shapes. The secondary rollovers for the "undo" and "redo" buttons contain "undo all" (essentially revert) and "redo all" functionality. When supported, these controls should be the left-most item in the toolbar.

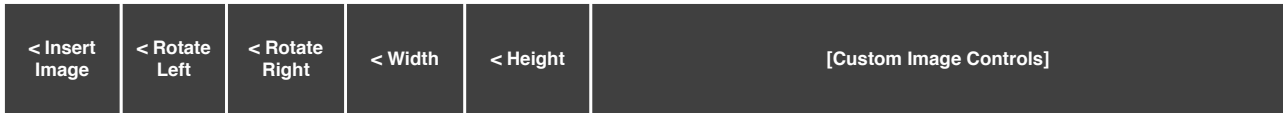
Copy/Paste: Sugar has a fully featured clipboard within the Frame, and as such we want to encourage children to copy and paste text, images, or anything else both within and between activities freely. The copy/paste, reuse, reorganize, modify, and share approach is core to the educational and creative experience that the laptops are designed for. We've simplified the paradigm, eliminating "cut" command from the top level editing commands. The distinction between "cut" and "copy" can seem unclear to those unfamiliar with computing, and so we've chosen to embed "cut" functionality in the secondary rollover beneath the "copy" button, and called it "copy and erase." When present, these controls should be left-aligned, immediately following the undo/redo commands.

Find/Replace: Wherever

Text Toolbar



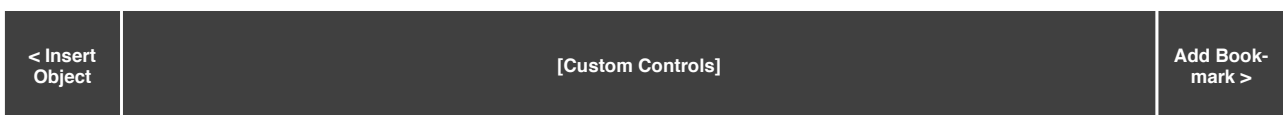
Image Toolbar



View Toolbar



Custom Toolbars



Inserting Objects

Many custom toolbars will provide controls useful in editing contexts which don't always exist by default. The Table Toolbar within the Write activity is a good example of this: Until a table has been inserted into a document, there is no "table context" within which to edit. Of course, once there is one, the Table Toolbar will be automatically selected whenever the table is selected. Since the act of inserting the corresponding object creates the context that the toolbar associates with, this control should always appear first within the toolbar. The Icon guidelines provide further information on the visual style for insert buttons.

Bookmarking

We hope to encourage discussion, iteration, and sharing on the laptops, and so we hope to encourage the idea of annotation across many different activities. When activities support textual annotations, highlighting, or other complex forms, they should have an Annotation Toolbar containing all of these features. Some activities, especially early on before a system-wide annotation system exists, may simply like to implement basic bookmarking. Though we hope to implement bookmarking as a subset of the annotation model, this particular feature is essential to some activities, and can be implemented in simpler ways in the meantime. When bookmarking exists as a single action within an activity, it should be placed to the far right of any custom toolbar which seems appropriate.

Hiding Toolbars

Although every activity requires at least the Activity Toolbar, developers may desire to hide the Toolbox in order to provide an all-encompassing experience, such as in an adventure game or a slideshow, or simply to make use of the full screen as an editing area. However, activity developers should consider these use cases carefully and take the following guidelines into account when taking advantage of this feature.

Soft Hiding

Most activities should use soft-hiding, which means that although the Toolbox will be hidden completely from view, it will still be accessible by moving the cursor to the top edge of the screen, provoking it to slide out and exposing the controls. This works great for casual or turn-based games, as well as any games which don't require the mouse. In these instances, the ability to access pref-

ferences, share or invite friends to the activity, start a new game, and of course exit the activity remains available at all times. This is also useful for presentation modes, such as slideshows, allowing the child to access the bar to perform operations such as next, back, and of course stop slideshow, thus showing the toolbar permanently again. When a soft-hidden toolbar slides into view, it slides in on top of the activity view beneath, eliminating the need to reflow the content; When hiding is turned off, it again embeds itself within the view, thus shifting the content downward.

When the sole purpose for hiding the Toolbox is to provide additional screen area for viewing or editing, a control within the View Toolbar should provide this option. Activities should not automatically invoke soft-hiding for this purpose (unless the aforementioned toggle is stored as a preference in the selected state). Though the laptops have a small viewable screen area, the choice to hide potentially frequently used controls should be left to the children.

When soft-hiding of the Toolbox is in effect, pressing the escape key should always reveal it again, exiting any mode related to its hiding (such as the slideshow). If no explicit toggle button or action (such as "start slideshow") exists to turn soft-hiding back on, activities may institute a timeout, after which it turns soft-hiding on again without input from the child. The activity must ensure that soft-hiding is never initiated without explicit interaction whenever the cursor remains within the Toolbox area.

Hard Hiding

Activities which make use of the entire screen, and moreover require active cursor movement across it, may wish to hide the Toolbox completely from view, eliminating the possibility that it could be invoked at the screen edge via the mouse. Hard-hiding allows activities to do this. The primary use case for this mode is action games which could be interrupted accidentally during gameplay. As such, these guidelines are written with respect to a fullscreen game, but their principles should carry over to other uses activity developers may find.

Hard-hiding removes all access to the Toolbox for an extended period of time, and therefore should only be used within activities which don't have their own toolbars. Also, since hard-hiding eliminates any means of invoking the Toolbox, the activity must always provide any of the basic functionality otherwise contained within the Activity Toolbar itself. Any game which supports networked play over the mesh should always provide a way for the initiator to name the instance.

To remain consistent with soft-hiding mode and with general expectations, the escape key should always provide a mechanism for exiting the game, even if another means exists. The escape key could immediately exit the game, returning to an intro screen, but it will more likely pause the game and reveal a menu containing an option to do so.

Rollovers

Rollover Animation Phases By Time

Animation Phase	Time Offset (seconds)	Time Offset (seconds)
Immediate Background Change	0.0	0.1
Primary Rollover Begins Expanding	0.1	0.2
Primary Rollover Displayed	0.3	0.4
Secondary Rollover Begins Expanding	0.7	0.3
Secondary Rollover Displayed	1.0	----

Primary Rollover

Secondary Rollover

Rollovers as Menus

Rollovers as Contextual Menus

Rollovers as Dialogs

Controls

Sugar defines its own set of control widgets that will create the user interface on the laptops. Though most of the available controls will match developers' expectations, we've also developed some new controls and altered behaviors of others to better suit the user experience on the machines. With these new developments comes a new set of guidelines for their usage and placement which both maintains the broader set of goals and metaphors setup within Sugar and aids in the development of collaborative user interface environments.

As the control widget specification and guidelines compose a substantial portion of this document, internal page by page navigation is available by clicking on the header for any control set. In addition, the individual pages provide detailed specifications for the widgets and their various states. Finally, we're providing developers with an Adobe Illustrator file of the most up to date Sugar Control Specification in [.ai \(07.04.05\)](#) and [.svg \(07.05.18\)](#) in order to streamline early application layout and design mockups. Please understand these specifications and their implementations remain in early development, and their designs may change as the UI progresses. We'll provide links to APIs for the controls as they become available; Thanks for your patience.

Control Regions

In sugar, a predefined set of control regions provide the surfaces upon which controls may be placed. Limiting the background colors that the controls sit on allows for a consistent set of visual rules to define both the type and state of various controls while maintaining sufficient contrast even without the use of color. Shown to the right, the basic regions are the Canvas, Panel, Toolbar, and Palette.



Canvas: The canvas is the general purpose "creation" space within any activity — the region for drawing, writing, or otherwise working within it. Since creation ranks high among our goals, we also hope that this is the dominant region of the screen in most activities. The canvas region is specifically for generating content and 'should not' have any controls placed upon it. As such, the canvas may be any color the activity or the user desires, though Sugar specifies both white and black as the basic defaults.

Toolbars: Activity toolbars have a unique dark gray color which distinguishes them from the other control regions. The Frame also uses this unique shade of gray to indicate its purpose as an omnipresent toolbar that

a child may activate from within any view or activity. Each activity will have a toolbar at the top of the screen, though when necessary additional toolbars may also be specified.

Panels: When appropriate, an activity may devote a portion of the screen to additional controls which make more sense outside of the toolbar and within the global activity context. Controls placed in this region should always remain relevant at all times, regardless of any other state associated with the activity or editing context.

Palettes: Palettes serve as multipurpose control regions which appear in several contexts. In an interface without traditional windows and menus, palettes step in to provide a versatile solution in many aspects of the user interface as the second layer of control. For a full description of their uses, please see the section on Palettes and Rollovers.

Controls: The remainder of this section focuses largely on the specific controls and guidelines for their usage within your activity's UI. The designs of all controls presented here follow a basic set of rules regarding colors, line weights, and sizing for each of its various states. More detail about these broader design decisions is discussed in the custom controls section.

Buttons

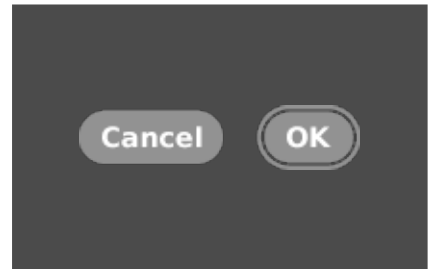
Text Buttons

Usage:

Behavior:

Guidelines:

States:



Unpressed



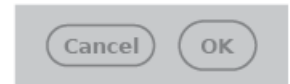
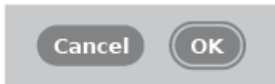
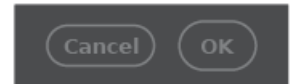
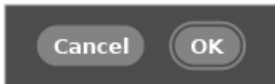
Focused



Pressed



Inactive



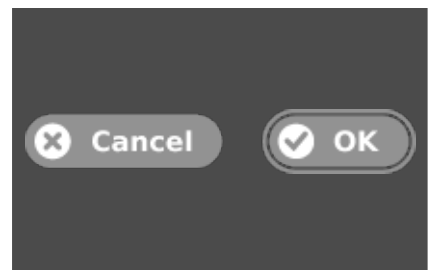
Text with Icon Buttons

Usage:

Behavior:

Guidelines:

States:



Unpressed



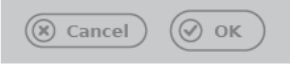
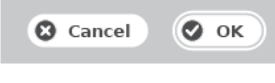
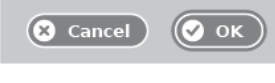
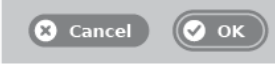
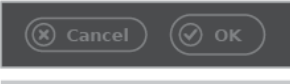
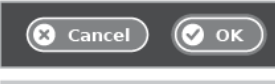
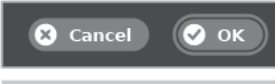
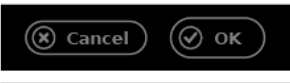
Focused



Pressed



Inactive

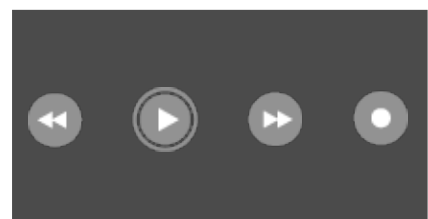


Icon Buttons

Usage:

Behavior:

Guidelines:



Toggle Buttons

Usage:

Behavior:

Guidelines:

Palette Buttons

Usage:

Behavior:

Guidelines:

Basic Selection Controls

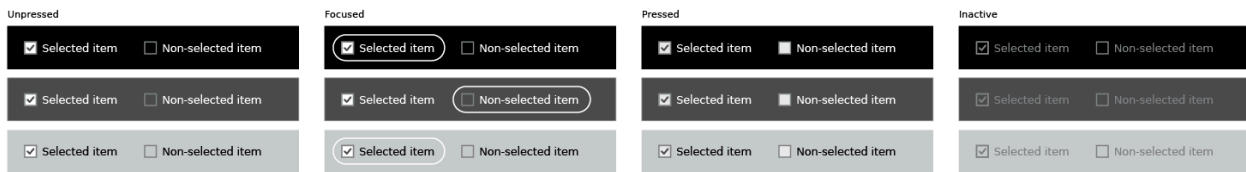
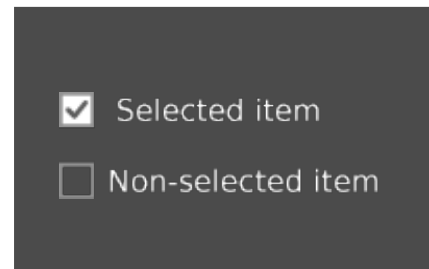
Checkboxes

Usage:

Behavior:

Guidelines:

States:



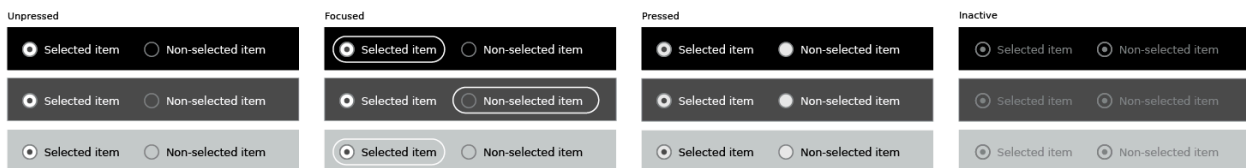
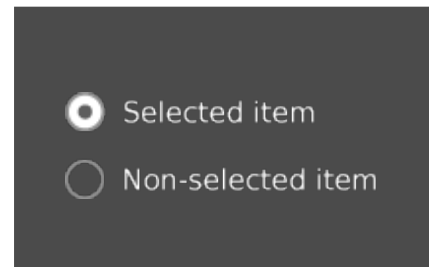
Radio Buttons

Usage:

Behavior:

Guidelines:

States:



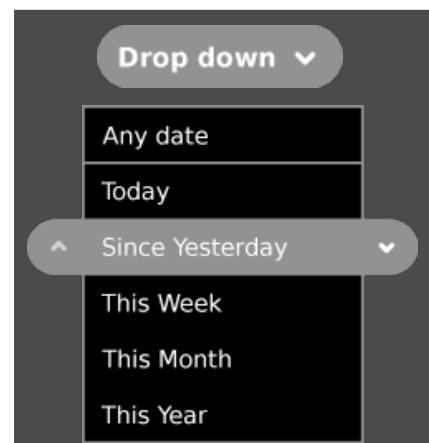
Popup Menus

Usage:

Behavior:

Guidelines:

States:





Popup Palettes

Usage:

Behavior:

Guidelines:

States:



Combo Boxes

Usage:

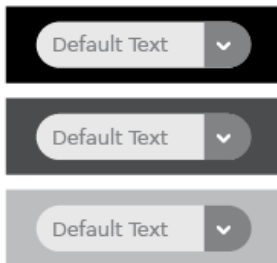
Behavior:

Guidelines:

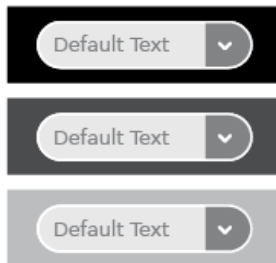
States:



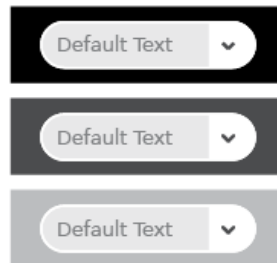
Unselected



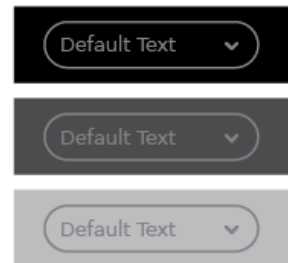
Focused



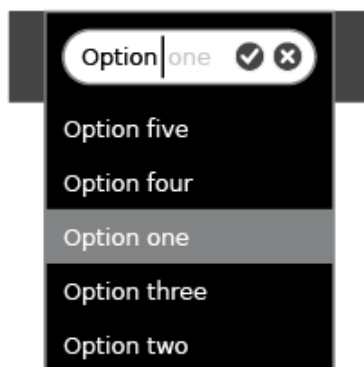
Pressed



Inactive



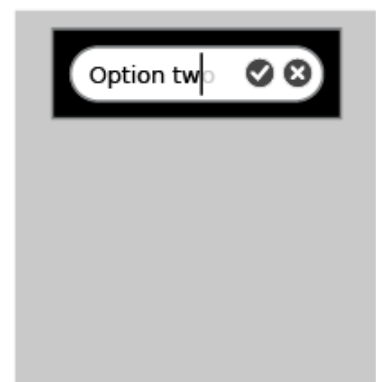
Selected with menu



Selected with menu, narrowed results



Selected, unambiguous match



Advanced Selection Controls

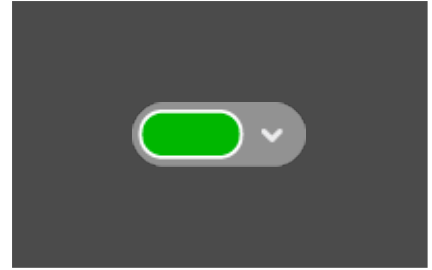
Color Picker

Usage:

Behavior:

Guidelines:

States:



Unfocused



Focused



Pressed



Inactive



Date Picker

Usage:

Behavior:

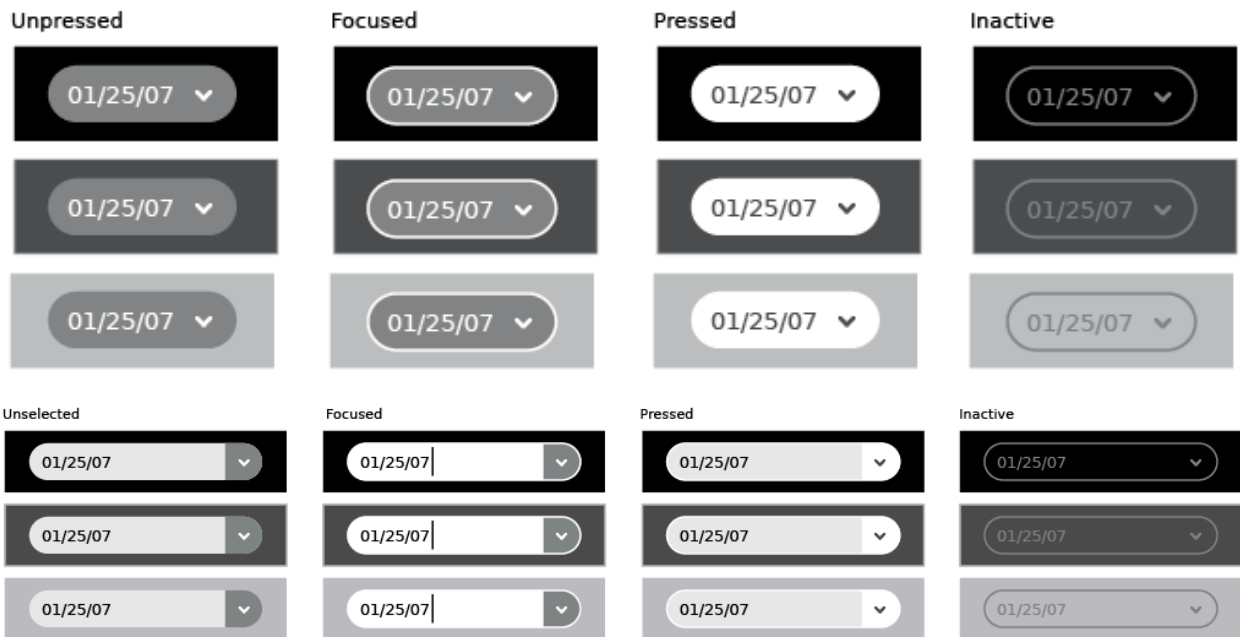
Guidelines:

States:



Selected with palette





Object Picker

Usage:

Behavior:

Guidelines:

Popup Palettes

Usage:

Behavior:

Guidelines:

Find/Replace Palette

Usage:

Behavior:

Guidelines:

Adjustment Controls

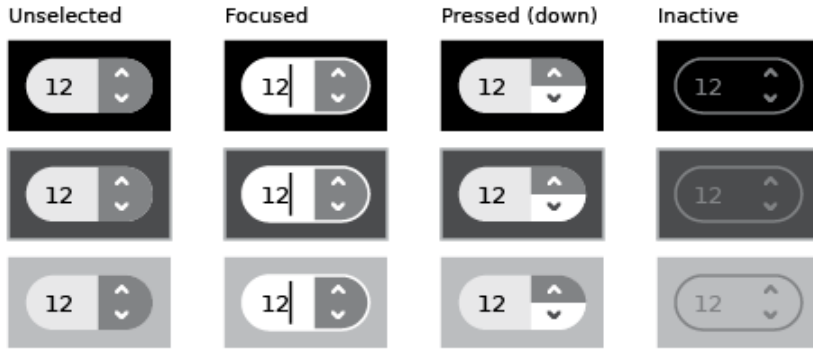
Steppers

Usage:

Behavior:

Guidelines:

States:



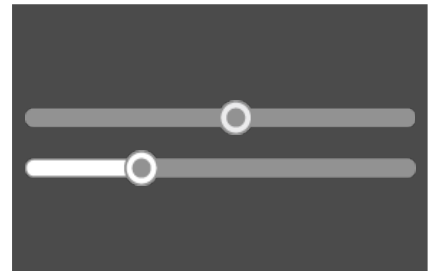
Basic Sliders

Usage:

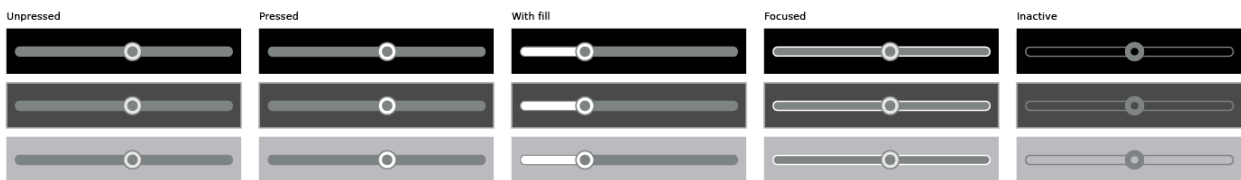
Behavior:

Guidelines:

States:



Sliders, plain



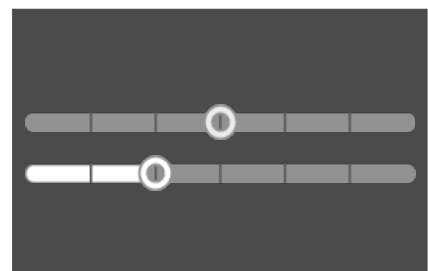
Sliders with Indication

Usage:

Behavior:

Guidelines:

States:



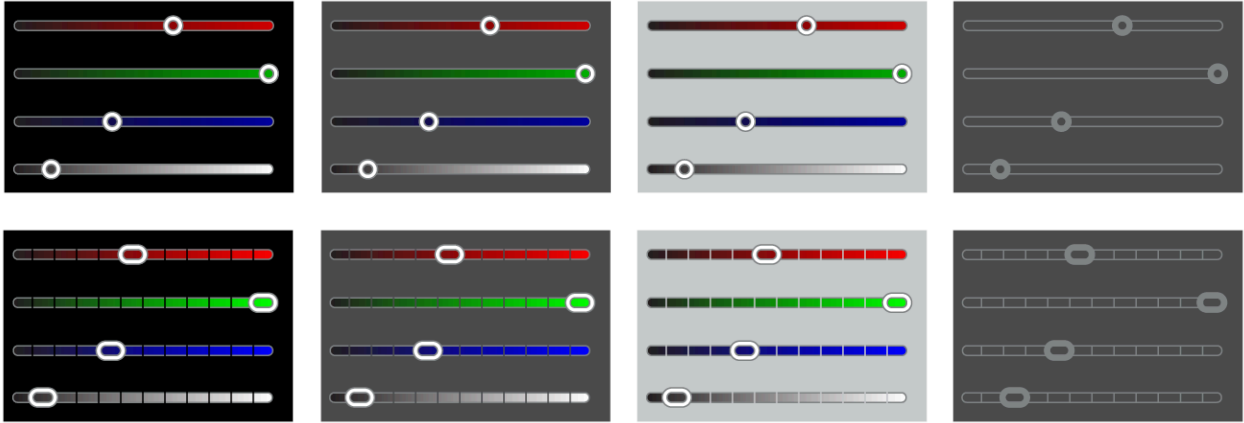
Color Sliders

Usage:

Behavior:

Guidelines:

States:



Indicators

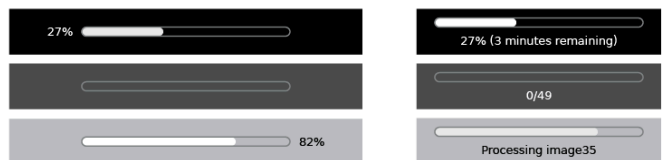
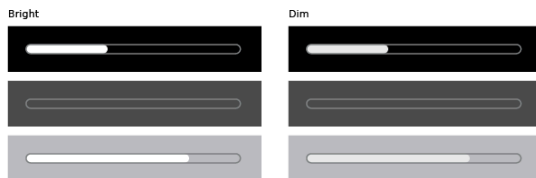
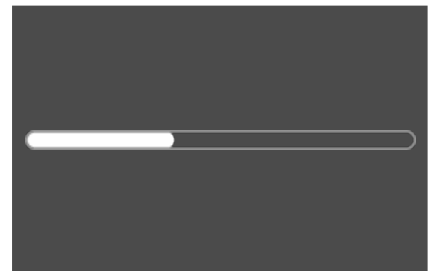
Determinate Progress Bars

Usage:

Behavior:

Guidelines:

States:



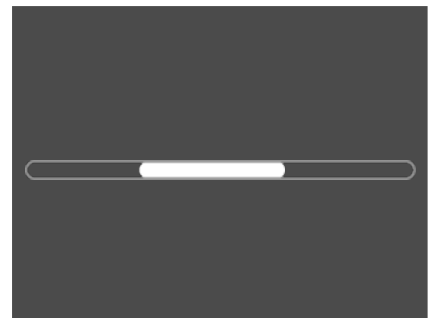
Indeterminate Progress Bars

Usage:

Behavior:

Guidelines:

States:





Activity Spinners

Usage:

Behavior:

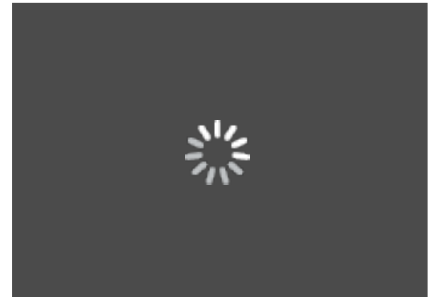
Guidelines:

States:

Small



Large



Level Indicators

Usage:

Behavior:

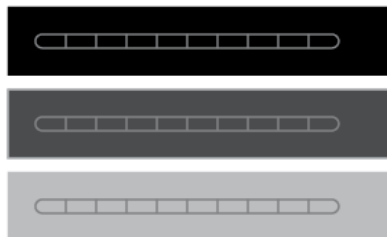
Guidelines:

States:

Active



Inactive



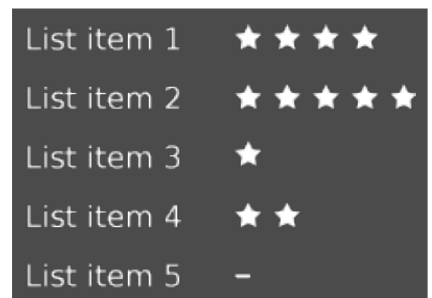
Rating Indicators

Usage:

Behavior:

Guidelines:

States





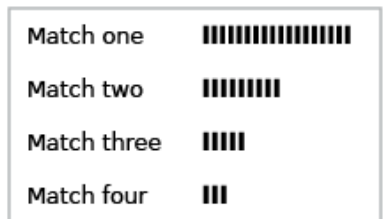
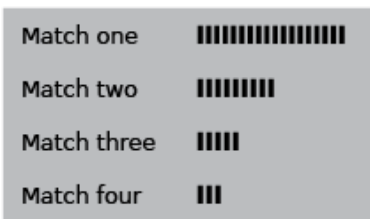
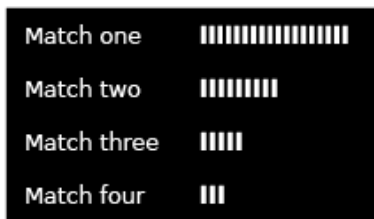
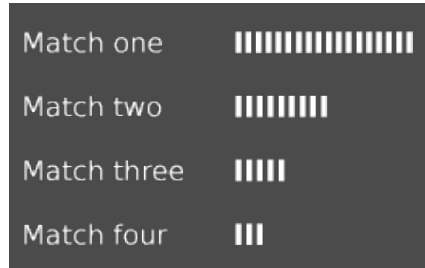
Relevance Indicators

Usage:

Behavior:

Guidelines:

States:



Text Controls

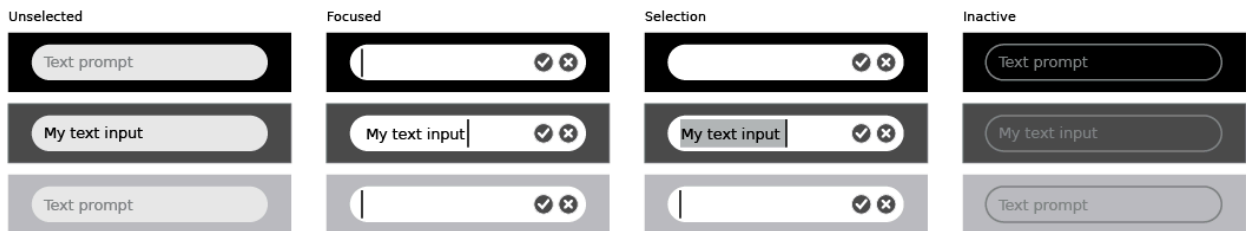
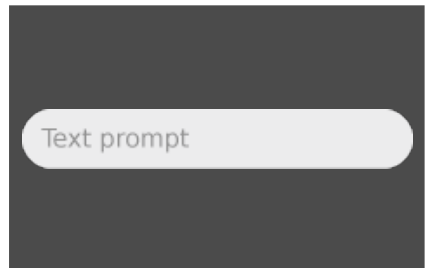
Text Fields

Usage:

Behavior:

Guidelines:

States:



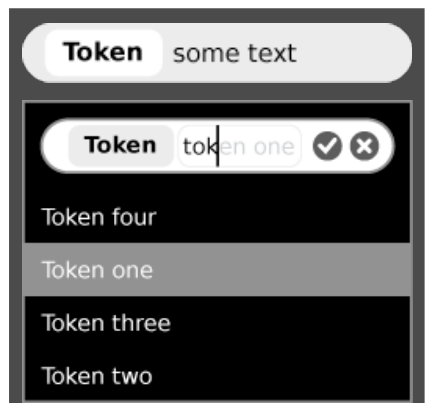
Tokenized Text Fields

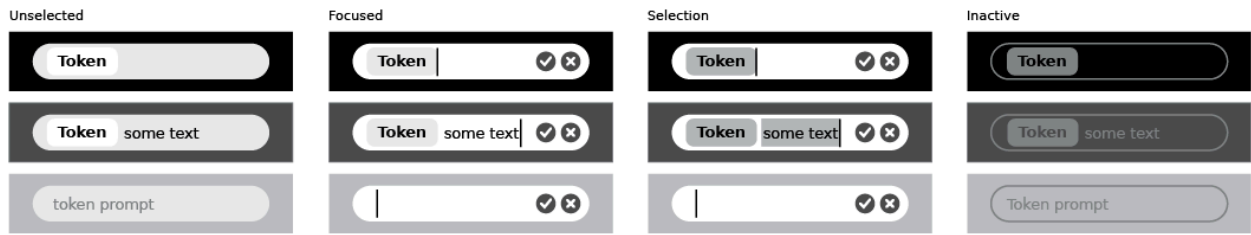
Usage:

Behavior:

Guidelines:

States:





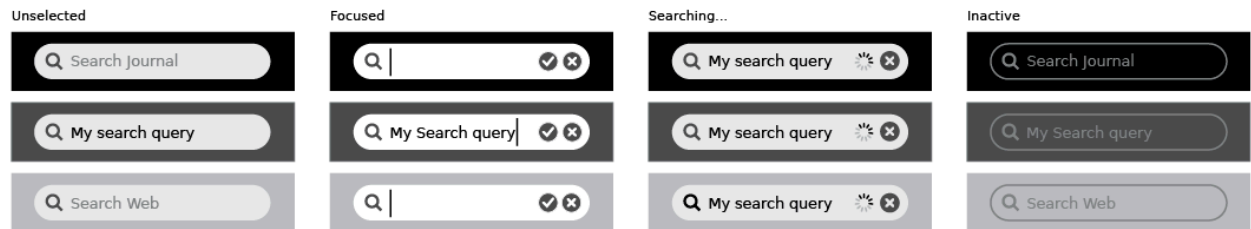
Search Fields

Usage:

Behavior:

Guidelines:

States:



Tokenized Search Fields

Usage:

Behavior:

Guidelines:

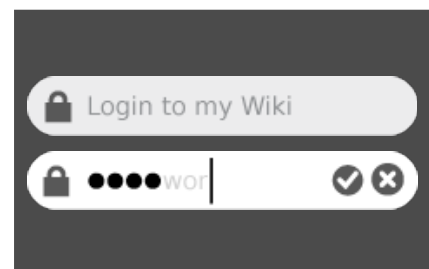
Password Fields

Usage:

Behavior:

Guidelines:

States:



Multiline Text Fields

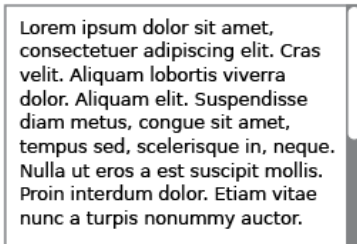
Usage:

Behavior:

Guidelines:

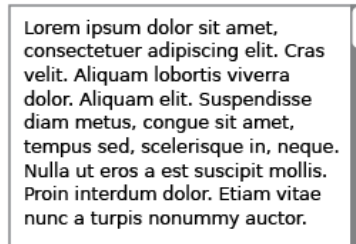
States:

With scrollbar

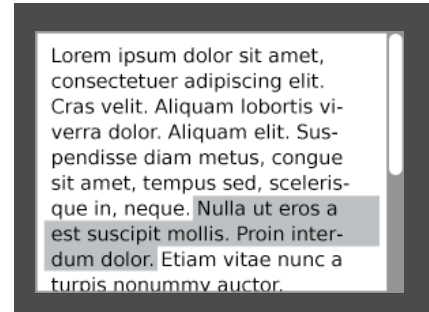


>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras velit. Aliquam lobortis viverra dolor. Aliquam elit. Suspendisse diam metus, congue sit amet, tempus sed, scelerisque in, neque. Nulla ut eros a est suscipit mollis. Proin interdum dolor. Etiam vitae nunc a turpis nonummy auctor.

Minimum scrollbar height

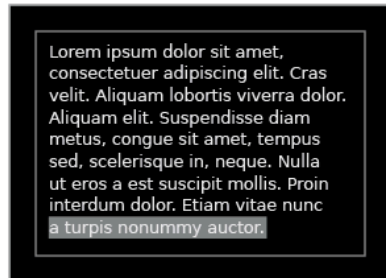


>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras velit. Aliquam lobortis viverra dolor. Aliquam elit. Suspendisse diam metus, congue sit amet, tempus sed, scelerisque in, neque. Nulla ut eros a est suscipit mollis. Proin interdum dolor. Etiam vitae nunc a turpis nonummy auctor.

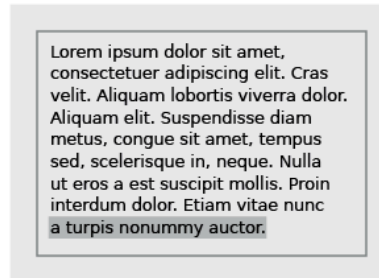


>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras velit. Aliquam lobortis viverra dolor. Aliquam elit. Suspendisse diam metus, congue sit amet, tempus sed, scelerisque in, neque. Nulla ut eros a est suscipit mollis. Proin interdum dolor. Etiam vitae nunc a turpis nonummy auctor.

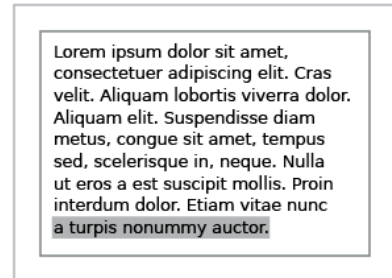
Selected (non-editable)



>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras velit. Aliquam lobortis viverra dolor. Aliquam elit. Suspendisse diam metus, congue sit amet, tempus sed, scelerisque in, neque. Nulla ut eros a est suscipit mollis. Proin interdum dolor. Etiam vitae nunc a turpis nonummy auctor.

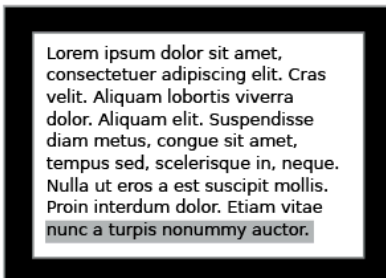


>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras velit. Aliquam lobortis viverra dolor. Aliquam elit. Suspendisse diam metus, congue sit amet, tempus sed, scelerisque in, neque. Nulla ut eros a est suscipit mollis. Proin interdum dolor. Etiam vitae nunc a turpis nonummy auctor.

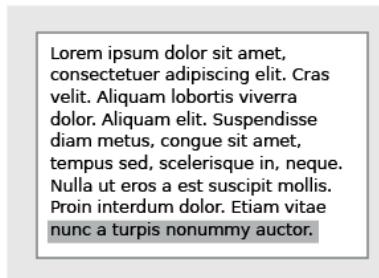


>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras velit. Aliquam lobortis viverra dolor. Aliquam elit. Suspendisse diam metus, congue sit amet, tempus sed, scelerisque in, neque. Nulla ut eros a est suscipit mollis. Proin interdum dolor. Etiam vitae nunc a turpis nonummy auctor.

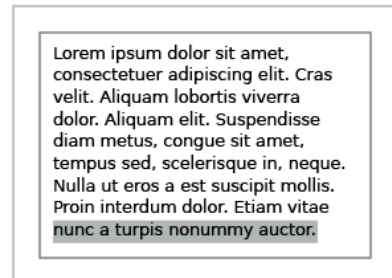
Selected (editable)



>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras velit. Aliquam lobortis viverra dolor. Aliquam elit. Suspendisse diam metus, congue sit amet, tempus sed, scelerisque in, neque. Nulla ut eros a est suscipit mollis. Proin interdum dolor. Etiam vitae nunc a turpis nonummy auctor.



>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras velit. Aliquam lobortis viverra dolor. Aliquam elit. Suspendisse diam metus, congue sit amet, tempus sed, scelerisque in, neque. Nulla ut eros a est suscipit mollis. Proin interdum dolor. Etiam vitae nunc a turpis nonummy auctor.



>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras velit. Aliquam lobortis viverra dolor. Aliquam elit. Suspendisse diam metus, congue sit amet, tempus sed, scelerisque in, neque. Nulla ut eros a est suscipit mollis. Proin interdum dolor. Etiam vitae nunc a turpis nonummy auctor.

[View Controls](#)

Basic Sort Bars

Usage:

Behavior:

Guidelines:

Advanced Sort Bars

Usage:

Behavior:

Guidelines:

Sort Bar View Toggles

Usage:

Behavior:

Guidelines:

Tab Bars

Usage:

Behavior:

Guidelines:

Tabbed Sidebars

Usage:

Behavior:

Guidelines:

Disclosure Triangles

Usage:

Behavior:

Guidelines:

Tooltips

Usage:

Behavior:

Guidelines:

Grouping Controls

Separators

Usage: Separator width has not previously been specified in the HIG, but historically, two separators are the same width as one toolbar icon.

Behavior:

Guidelines:

Trays

Usage:

Behavior:

Guidelines:

Custom Controls

When to Use Custom Controls

We have created a set of controls to suit most basic needs for Activity development. Using these controls wherever possible ensures that interfaces across Activities remain consistent and clear, and also maintains the visual style of the Sugar interface. Nonetheless, their basic functionality may prove limiting in some cases, and we certainly hope to inspire new and exciting uses of the

various capabilities of the laptops, including their collaborative nature, and don't want to stifle additional ideation and design. For this reason, we are providing a general set of guidelines for the creation of custom controls that match the visual style of Sugar and maintain some basic standards for compatibility and usability within the Sugar interface.

Before implementing a custom control, be sure that the need truly exists. In some cases, you may find that a combination of simpler controls may suffice. In other cases, more complex functionality can be accommodated through use of a Palette. Additionally, be sure that the need is a functional one; Custom controls should have additional or different behavior that the current controls don't provide, and not simply define a different aesthetic for existing ones. Again, think carefully about the requirements for the control, and be sure that the functional need itself merits the work associated with creating the new control.

Control Color Palette

The Sugar interface defines a strict palette for the control widgets, along with guidelines for using these colors for various states and control types. Custom controls should adhere to the same set of colors and associated rules in order to ensure that their behavior mimics those of the core Sugar controls, and can be easily inferred by those who are familiar with them. Though their name and primary uses define these colors, any color within the palette provided below may be used for various parts of your control where it doesn't interfere with the basic rules associated with them.

Name	Primary Uses	Hex	% Gray
Black	Palettes, Popups	#000000	100%
Toolbar Grey	Toolbars, Trays, Palette Groupings	#404040	75%
Button Grey	Default Button States	#808080	50%
Selection Grey	Selections, Panel Groupings	#A6A6A6	35%
Panel Grey	Panels, Desktop	#C0C0C0	25%
Text Field Grey	Text entry background	#E5E5E5	10%
White	Pressed states, multiline text areas	#FFFFFF	0%

Control Sizing

In accordance with [the grid system](#), we have defined two standard sizes for controls: 75px and 45px. These sizes determine the bounding box for a given control, with 45px being the smallest recommended active (clickable) area for any control component. Note that the control itself may actually be smaller than this canvas, depending on the desired context. For instance, the icons for buttons fill up to 60% of the canvas area, with the actual icons having sizes of 55px and 27px.

Cursor

Coming soon...