BananasandPears

Contents [hide]

1 MeeGo Handset Interaction Guidelines
2 Introduction
2.1 How to use this document
2.2 Introducing MeeGo
2.3 What Makes MeeGo Great?
2.4 Design Principles
3 MeeGo Basics
3.1 Overall UI Model
3.2 Overview of Gestures and Touch Interactions
3.3 Lock Screen
3.4 Launcher
3.5 Switcher
3.5.1 Order of Applications, Browsing, and Changing Modes
3.6 Widget Space
3.7 Application Lifecyle
3.8 Widget Lifecycle
3.9 Extension Lifecycle
4 Core Interactions
4.1 UI Feedback
4.2 Text Input
4.2.1 Variation Examples
4.3 View Menu
4.4 Object Menu
4.5 Object View
4.6 Full Screen Mode
4.7 Confirm vs. Cancel
4.8 Deleting
4.9 Applying Actions to Multiple Objects
5 Integrating Your Service to Core Experiences
5.1 Services in MeeGo
5.2 Integration to Native Applications
5.3 Plug-ins
5.4 Offering Content
5.5 Extending Content
5.6 Case Study: X Service
5.7 Integrating Native Application Functions
5.8 Localizing Services
5.9 Overview of Extensibility in Core Applications
6 Designing Your Application
6.1 Design Tips
6.2 Application Types
6.3 Application Basic View
6.4 Backstacking Behavior
6.5 Multi-tasking
6.5.1 Deep Linking From Widgets, Other Apps, and Notifications
6.5.2 Launching Multiple Windows
6.5.3 Re-opening Windows/Window Merging When Deep-Linking
6.6 Navigation Templates
6.6.1 Combining Drill Down with Tab Bar
6.7 Recommendation on Layouts
6.8 Other Application View Examples
6.9 Portrait vs. Landscape
6.9.1 1 vs. 2 Columns
6.10 Device Content

6.11 Settings
6.12 Navigation within your applications
6.13 How to structure commands (embedded vs. object menu)
7 Theming
7.1 Design Tips
8 Framework Features
8.1 Relying on the Framework
8.2 Accounts
8.3 Pickers
8.4 Status Bar and Status Menu
8.5 Notifications
8.6 Notification Drawer
8.7 Presence
8.8 Sharing
8.9 Content Manager and Search
8.9.1 Online vs. Offline
8.10 Tagging
8.11 Favorite
8.12 Transfer Manager
9 Other Considerations
9.1 Localization
9.2 Connectivity
9.3 Crashes and Errors Handling
10 Appendix
10.1 Common Components

MeeGo Handset Interaction Guidelines

Introduction

How to use this document

Purpose Ultimately this document will act as a guide for both Nokia services and 3rd parties who wish to design an application for MeeGo devices. This document is written primarily for interaction and visual designers. Visual design guidelines may be added on a future version, or may also be a companion document.

This is not a Specification Document This is not meant to replace detailed technical specifications. Instead, it is meant to represent a description of the overall interaction model for MeeGo in an at-a-glance fashion. It is the expectation that technical specifications will be available as a companion to this document, including a detailed description of each available tool and a recommendation of when to use it.

Introducing MeeGo

MeeGo has been designed for the new mobile world, where people are at the center of fast flowing streams of information, where they expect to tailor their device with a myriad of applications and where new economies are evolving around those applications.

These guidelines have been written to help you design and develop applications for the new mobile world.

MeeGo is a direct touch UI, meaning that users manipulate objects, such as a thumbnail of an image, directly through touch interactions. Content is surfaced and navigation hierarchies should be shallow and accessed through simple navigation systems. In addition to direct touch, MeeGo is optimized for multi-tasking usage and provides a rich platform integrated Applications.

The MeeGo interface is scalable for different screen sizes, resolutions, and aspect ratios and it supports both portrait and landscape orientations.

What Makes MeeGo Great?

Multi-tasking People’s daily lives are complex and busy. We tend to do a lot of things at once and often the things we want to do on our mobile devices – making an appointment over the phone, sending a link to a friend, finding a listing on the map - involve two or more of its applications. The heart of the MeeGo UI is the switcher, which allows people to quickly move around the applications they need to get something done.

Connected, vibrant and alive MeeGo is designed to be connected to the web all the time. MeeGo’s applications should feel ‘alive’ with activity, making the mobile device a true life hub. This empowers people and makes them feel connected.

Personal and expandable MeeGo provides users the ability to make the device their own by adding applications, extending applications, and by easily arranging quick access to favorite applications. Personal touches can be added through content and social feeds, profile customization, and standard features, such as ringtones and wallpaper.

Comes with Ovi Understanding that a mobile device is so much more than calls and text messages is key to MeeGo’s value. Ovi services and applications feel like part of the device because they are integrated with core components like search and sharing through integration with each other, allowing people to flow across several applications and services to do what they need to do. The Ovi store helps people discover and expand their device, offering a large range of applications and extensions.

Design Principles

Direct Objects are manipulated directly with your finger. Never hide object-related functions away from the main object. Hiding related functions/actions in menus should be avoided. All actions should take place near the object itself. Touch is an analog control (well, almost).

Responsive Direct UI must give immediate feedback to the user’s actions. When scrolling or panning, the moved objects need to follow user’s finger without a noticeable lag. Visual, audio, and haptic feedback must feel that they are in perfect synch. The multimedia playback has high resolution and frame rate that gives a feeling of performance.

Simply beautiful The UI must please all senses. Visual, aural, and haptic feedback must be beautiful, yet simple. Visual and aural (and haptic) metaphors and effects must be consistent. Decoration should have meaning, for example show what can be touched. The UI is alive, for example showing subtle animations.

True Human Design The key attributes of MeeGo's design vision are: human, simple, and friendly. MeeGo prioritizes people over any other content.

Getting the basics right People still need to meet the demands of every day life, calling their mom, remembering the milk, and messaging a colleague from the bus stop. MeeGo should make these things a joy and never let its power get in the way.

MeeGo Basics

Overall UI Model

Lock Screen Presented when user first wakes up the device.

Home Access to open applications (through the switcher) and widgets (through Widget Space). User can toggle between both. Favorite applications are presented, as well as access to launcher.

Overview of Gestures and Touch Interactions

To Add

Lock Screen

The lock screen is presented when the user presses the power key to wake the device from idle state.

The screen presents the wallpaper (customizable by the user), date and time, and the lock action button. The device is unlocked by dragging the lock button into the wallpaper area.

Notifications are also supported in the lock screen. They stack up in a similar way to the Notification Drawer (Please see notification page 50 for more details).

Launcher

The launcher is a view that contains links to all applications installed in the device. In the launcher, the user can browse through the applications and add up to 4 links to the quick launch bar at the bottom of the screen. In editing mode, the user can also change the order of application links.

Applications are presented in a 4 x 4 grid. In the case where there are more than 16 applications, more pages of the same grid are added to the right. Paging through the different grids is done by swiping the current grid off screen, hence bringing the new one into view. When the user installs a new application, it goes to the end of the application grid.

The user opens the launcher by selecting an icon at the bottom of the Homescreen (placed together with Favorite Application links). To close, the same icon is used.

All applications require a name and icon for the launcher.

Switcher

Users will often want to (or have to) use more than one application at a time, such as listening to music while writing a message. This is known as multitasking, and the switcher is an area where multiple tasks are managed. The basic behavior of the switcher is as follows:

When an application is first launched, a new task starts. This task is represented in the switcher by a live thumbnail of the current content within the application, that is a blog web page would always display the latest blog entry. The switcher is always accessible via the home button in the top left corner.

Whenever a second application is opened, the new task is included within the switcher. With that, MeeGo allows users to quickly switch between applications by pressing the home button. Applications can be opened from the launcher, widget space, or even from other applications, such as opening a map from an email.

When the user clicks on an application in the launcher, that is already open in the switcher, the system opens the window that was already open. It does not restart the application. Additional behavior exists for opening deeplinked views of applications, see Re-opening Windows/Window Merging When Deep-Linking for more details.

For detailed information about designing your application with taskswitching in mind, see Multi-tasking, which covers backnavigation behavior, spawning windows from within an application, deep linking from one app to another, and opening applications from notifications.

Order of Applications, Browsing, and Changing Modes

Tasks are shown in the order they were opened, from left to right. Every time a new task is opened, it pushes the previous task panels to the left and adds its thumbnail view to the far right.

The default view of the switcher presents the latest accessed task in focus, with the other, different tasks running to the right. Thumbnails are presented in chronological order, as they are opened.

It is possible to browse through the thumbnails, one by one, by slowly dragging them, or to use a quick swipe to rapidly pan from one side of the list to the other. During this quick swipe, a tap stops the transition at tapped point, but does not open the application. To open the application, the user needs to tap the desired thumbnail.

Users can use a pinch multi-touch motion to change the switcher into an overview mode. By pinching in, thumbnails move into a grid display. The grid can scale up to a 3x3 version, thereafter the switcher then starts to create new pages. Pinching out, while in the grid mode, changes the view back to larger stacked mode.

Widget Space

The ‘Widget Space’ is a customized area where users can add/edit their widgets.

The user can find the widget space by flicking vertically from the switcher. Once the user is within the space, they can horizontally scroll to view multiple areas for widgets.

Widget Widgets are a way of providing quick access to surfaced content and functionality. Widgets should be snappy, bite-sized pieces of content and should not be confused with applications.

Please refer to the ‘Designing your Widget’ section for more information on Widgets.

Application Lifecyle

The user can extend the device capabilities by installing new applications, such as widgets, applications extensions, and software plugins to support new services.

The user can get installable items from multiple sources: from various online content sources (such as Ovi store), or by downloading from various device services or Applications (such as browser, e-mail, and Bluetooth file transfer). All installed applications appear in the device main menu as icons.

Package Manager The device package manager provides the UI style and mechanism for handling application installation flows, displaying installation progress, giving ways to handle the technical issues and solve common error cases.

Installed applications are generally linked to the repository that contained the installed packages. The package manager tracks these repositories and provides a mechanism to detect if new updates are available for the installed application packages.

If updates are available for any installation, the package manager notifies the user (through an Event Banner – see Notification section), who then allows installation of the updates.

The user can uninstall applications, either through the main menu (from the object Menu or menu editing mode) or then through the package manager UI.

Widget Lifecycle

Almost all the rules about application lifecycle apply to widgets, the main difference being the point of discovery (that is, installing new widgets) and removal.

Following the direct UI style, objects get added where they will exist. For widgets that means the widget space. Long pressing anywhere in the widget space takes the user to edit mode, where it is possible to remove existing widgets, add new ones and adjust settings per widget. Settings is not compulsory (such as shortcuts have no settings), but for widgets that do, it is launched by pressing the settings button in the bottom left corner.

Widgets can be found in the local library, as well as the Ovi Store (both presented when the add button is actioned). To remove widgets, the user long presses the widget itself, which opens the object view with the option to delete (or find updates).

Extension Lifecycle

The user can discover new extensions through the parent application or directly through discovery channels (such as Ovi store client).

Each application, that can be extended by plug-ins, has to provide a UI view for extensions management. Extension view displays overview of currently installed extensions, and provides integration point to discovery channels.

Once the user selects to install an extension, the application extensions view displays the progress of download and installation of extension. If the extension has a visible UI component on application view, the installation can also be indicated at location of UI component.

The extensions are delivered as debian packages, and installation is handled by the package manager. It remains the responsibility of the application to indicate if an extension requires user action or configuration to set it up or if the extension was not installed successfully.

The user cannot install an extension for a device, if the parent application does not exist in device.

Updates can be released for extensions. The package manager handles informing the user about available updates and handles the installation of updates.

The user can uninstall any extension from device through the package manager. If the parent application is uninstalled from the device, all extensions that have been installed for that application are also removed from device.

Core Interactions

UI Feedback

Direct feedback refers to the response the user receives when using the touch UI. When there is a change, the user must get feedback. Indirect feedback refers to the response the user receives while either not using the device or using it so that the attention is not directed to the UI.

Direct feedback can be haptic, auditory, or visual feedback from interaction with the touch screen or hardware buttons. Indirect feedback is typically in the form of notifications. Notifications may contain both sound and haptics, but they can also be visual only, or contain only visuals and sounds (or in principle also visuals and haptics alone).

When you need to catch the user’s attention, you can use auditory and haptic feedback, in addition to visual feedback. The user’s attention is usually directed away from the device when the device’s UI is locked and the screen is blank. However, the user’s attention can be elsewhere also, when the UI is in use. A time-consuming operation may cause the user to look elsewhere, or the user’s attention may be on the hardware keyboard, so the events on the screen are easily missed. When feedback is needed almost immediately after the user’s last action (such as when the user tries to send an email without defining a recipient), a visual notification is usually enough, because the user’s attention is already on the device.

In some cases, you can also use auditory and haptic feedback to emphasize the visual message. For example, a confirmation query may have an additional questioning sound to emphasize that the user needs to pay special attention to the device. However, be careful with this kind of use for sounds and haptics, because users find it annoying if the UI requires a lot of attention all the time. You can use sounds alone for presenting events, or sounds combined with visuals, but do not use sounds alone with haptics.

You can use haptic feedback alone when you do not know whether the user’s attention is on the device. For example, when downloading a web page lasts so long that the user’s attention is not likely to be on the device, even though the device is not locked, a subtle tactile pulse can be a useful indicator.

You can use tactile feedback alone, when it is appropriate to confirm the user’s action, but a visual confirmation message is too much. For example, connecting the charger and the device starting to charge (as opposed to not starting, due to a wrong charger type, for example). In this case, haptic feedback can also be used because the user is in physical contact with the charger connector and the device.

Note that the presentation of alerts and notifications depends both on the settings in the active profile and on the current UI state. For example, during a phone call the auditory part of many alerts is changed (presented as a beep, or the auditory part can be a simplified version of the original, or the sound may be omitted altogether).

Text Input

MeeGo provides support to both virtual and hardware keyboards.

The virtual keyboard is invoked automatically when an input field is in focus (that is when the user taps an input field). The Title Bar and Status disappear to optimize screen real estate. It has a portrait and landscape layout, opening according to the orientation it is being called from. If the user rotates the device, the layout is smoothly changed according to the new orientation.

The user can close the virtual keyboard by either tapping outside the text field or by dragging down the virtual keyboard. Note that the virtual keyboard does not act as a drawer in the sense that the user cannot pull it up. It will come up again once the text field is actioned again.

Whenever the hardware keyboard is opened, the virtual keyboard does not get loaded.

Clearing Pressing backspace clears a single character. The long press clears characters with key repeat. Long press duration and key repeat rate are the same as with Hardware keyboard. If the user selects the text and hits backspace, the whole selection is cleared. This is script (written language) dependent.

Language The language is set automatically, based on the UI language that the user selects in the first device startup. This can be changed in Settings.

Variation Examples

Extra keyboard line The framework offers the possibility of customizing the keyboard by adding an extra virtual line to the pad. This line may receive custom-made buttons, for example, a ‘.com’ button for the browser keyboard. When the hardware keyboard is opened, the extra virtual line still appears (it is possible to dismiss it by dragging it down).

In addition, the system uses this extra line for advance functions, such as Copy/Paste. If the user selects a section of the text (drag and hold gesture), the extra line appears with the copy function available. Once the user actions copy, the same button reverts to paste. More on Copy & Paste section of this document.

Confirming In most cases, applications present confirm buttons as part of their UI. This is true when sending a message or login in a website. Other applications perform actions as the user types. The perfect example is search, which automatically updates the list result with every letter typed.

However, other methods can be used. In a one liner text field, the enter key works as a natural confirmation key (similar to websites and other operating systems). This works when entering a password for example.

It is also possible to add a confirmation button to the extra line. It is not recommended for UIs like messaging, where the an overall view is desired before confirming (tapping outside the text brings the whole UI back), but useful for quicker inputs. This option could be used in a form, with a next action linking to the next text field.

View Menu

The View Menu is accessible by tapping the Title Bar. Tapping the Title Bar, the content area outside the View Menu or Close closes it. The homescreen button closes the view menu and takes the user back to the switcher. Not every view requires a View Menu (for example, Settings usually does not). In those cases, the title bar has no interaction.

Menus should contain second order items, presented as text strings, that are less frequently used. For primary and frequently used actions, consider placing them either in the toolbar or in the content itself. Even of it is scalable (that is scrollable), avoid overloading it.

View Menu may only include actions related to the current view (or application). These items may include commands (such as Add), and navigation options (such as Settings). It is also possible to include sorting (such as Alphabetical and Filtering, such as Favorites). However, these should never be mixed (consider a drill down to filtering in those cases). Furthermore, whereas filtering creates a history step (that is the back button takes the user back to the previous filter option), sorting doesn’t. When using the Tab Bar model, if needed, it is possible to present distinct View Menus for each navigation option.

Applications are generally responsible for ordering of menu items. The overall recommended ordering of View Menu is: Filter or Sort, Commands, Add-ons and Settings (in the main level of the Application). In deeper views, it is possible to include a ‘Go to Main View’ command, which takes the user to the top of the application. There is no need to include exit, since Maemo provide more tangible ways to perform those actions.

View menu options can have sub-views which are displayed as dialogue boxes. This should only be used for very simple/flat suboptions. For more complex sub-views, an actual drill down should be done, like drilling down to a Settings View.

Object Menu

An Object Menu is a list of related actions users can performer with objects. The menu is accessed via a long press on the object, which presents a dialogue with the available options. The menu is closed by tapping the content outside it. None of these, Home, View Menu, nor Close, is accessible.

Object Menu is a good place for shortcuts, for things that would take several clicks and actions to otherwise access from the view the user launched the menu from. If an action would take two short clicks in the views vs. a long click to open the menu and a click in the menu, it doesn’t really save time and effort from the users anymore.

The options available will depend on the object. For example, the People Object Menu includes “Call” and “Send a Message”. Music track Object Menu includes “Set as Ringtone” and “Share”. Whenever relevant, some actions are highly recommended to be included: “Details”, Delete, “See related” and “Mark as favorite”.

It is recommended that the menu presents these actions, in the following order:

- Primary action (such as call, view, play)
- Secondary actions (such as send message, set as wallpaper)
- When relevant, link to the object
- When relevant, share
- Negative actions (such as delete, remove, unsubscribe)

Note, however, that all users do not use the Object Menu, so it must not be the only place for an action. In addition, do not duplicate the primary action. For example, if an item on the UI opens with a tap, do not place an option for opening the item in the Object Menu.

Object View

The Object View is a sub-view, usually used for Media Objects, accessible through an info button placed close to the object. By clicking the button, the user drills down to a sub-view that revealing more (or related) information around that object.

This sub-view can present content from different parts of the device (through meta-data connection) as well as cloud based information (coming from either Nokia or 3rd Parties). Good examples are presenting tags related to an image, or recommended artists or videos related to a music track.

The back button takes the user back to the object. Alternatively, this can also be done by tapping on the object reference.

The information button works both as an independent floating item, or can also be attached to a bigger info bar. This info bar can carry some of the object info, which helps connecting the interaction with what the user gets with it. The info button should never be featured in the toolbar.

Full Screen Mode

For applications that need to provide a more immersive experience, Full Screen Mode is available. It is important, however, to always provide a way out of the application (and to application navigation when appropriate).

Full Screen can be presented in two different ways:

Controls a tap away All controls (Title Bar, Status, extra controls, etc) are removed, presenting only the maximized content. Controls are made visible by tapping the screen. This is specially useful for media related experiences.

Fixed controls Because MeeGo may run in devices with no other hardware navigation keys, it is important to provide a persistent way out of the application. In some cases, it is not possible to use the tap to show the controls (for example, a Flash Player application displaying interaction content). Whenever that happens, a persistent access to exiting the full screen is still mandatory (back or close).

Embedding Exit in Application Content In a few specific cases, like games, a more immersive experience might be desired, with no chrome. It is only possible to remove the persistent navigation out of the application if an easy to access exit command is available.

Confirm vs. Cancel

There are two distinct types of actions when it comes to confirming/saving or canceling.

Reversible actions The first type treats the back button as a non-destructive action. We should respect the actions of the user and assume those are intentional. Hence, by default, everything that is trivially reversible while staying in the same task is saved instantly when created or changed. In other words, these actions do not need to be confirmed or saved. The confirmation of that action happens twice already, with the act of entering it and actually editing the content.

Edits and operations within a view are assumed to instantly saved. The user should not lose something he has worked on by pressing "back". An example is when the user goes into a contact card and writes text in it, pressing "back" must not throw the changes away. Hence, a separate save button should not be shown as "back" should feel as a safe action anyway. Changes are saved instantly when created, and "back" just navigates away from the view.

Other examples:

- Creating or editing a calendar entry
- Selecting/deselecting Pickers (search filters, recipients in a contact list, etc)
- Modifying settings

Non-reversible actions Action buttons are needed when a new definite non-reversible action needs to be started. For instance, sending an email needs a ”Send” button, deleting an image needs the ”Delete” button. Pressing the action button should immediately commit the action and leave the view, generally returning to the start point of the task.

That means the back button should never start new actions. If the user selects ”Delete multiple images”, after having selected one or more, the user needs to press an explicit delete button to perform the action.

Sometimes MeeGo offers a route to a historical change, such as history of image editing. Even though that would make some actions reversible, if that still performs a big “move forward” (and it is quite a new UI feature), that too requires a confirmation.

By default, pressing the back button should not display a prompt, as the user shouldn’t be burdened with excessive prompts. If it is a lightweight action, losing it should not impose a problem, like dismissing a modification in the image editor. For more time consuming actions, where a draft folder is available, the action should automatically be saved there (such as email composer). For that reason, in addition to the automatic content creation rule, there needs to be a way to delete items: calendar entries, notes, contacts, email drafts, message drafts, etc.

Other examples:

- Sharing a picture
- Editing a video
- Cropping an image

Deleting

Objects can be deleted through:

- Toolbar (inside the object and always on the far right of the toolbar, aligning all negative action to the right)
- Action button (inside the object)
- View menu (inside the object)
- Object menu (from a list or inside the object, if the option is not available in a toolbar)
- Through multiple selection (inside list/grid view menu, more on the following page)

Consider using a confirmation prompt (using the query component). The query should present the name of the object (or number of objects when on multiples are selected). If the application has a reserved folder for deleted items, no prompts should be presented.

After deletion, a few follow up views are possible. There is no general rule, and it depends on what is the most natural consequence for each application (that is what the user expects to follow). Some examples are:

- From a list, the view is updated
- From inside an email, user is taken back to the previous email list.
- From picture gallery, the user remains in the gallery, and is presented with the next picture.

Applying Actions to Multiple Objects

Typically only one item may be selected in a collective view. Selecting a contact from your phonebook list drills down to the correspondent card, selecting a thumbnail in a grid opens a full view, and so on. Sometimes users might want to perform an action to multiple items.

By selecting the specific action to multiple (such as Delete Multiple Items), it is possible to mark all objects one desires to perform the action. This is usually available in the View Menu. The action is complete by pressing the correspondent button, and canceled by pressing back.

Marking mode typically presents:

- Title bar, where the title remains the same, and the close (or back) gets replaced by a back. Entering Mark mode represents a drill down.
- Content: a container in multi- selection mode (such as list or grid with tick marks enabled)
- Actions: operative command that completes the requested action (such as delete).

A toggle in the view menu for Mark all/Unmark all may also be provided. When user selects “Mark all”, all items will be selected. If user deselects “Mark all”, all items are deselected. “Mark all” is automatically selected if the user manually selects all items. Mark all is automatically deselected if the user deselects one or more items.

Integrating Your Service to Core Experiences

Services in MeeGo

MeeGo provides many ways for services to be part of its mobile offering. This section gives an overview and recommendations to 3rd parties, focusing on the integration to native applications. The other two options are described in detail in the next two sections.

Think about the service being proposed, and what is the best way to showcase it. Before beginning any design, consider all the options. This does not mean a service has to opt for one or the other. It is important to note all options can still make use of most Core Framework Features.

Integration to Native Applications What makes MeeGo stand out from other platforms is the possibility to integrate with native device experiences. For example, a music service can be experienced inside the native music application, without having to creating its own UI. That means less development work, more discoverability for a service and a seamless experience for the user. Integration may happen on different levels, described in the following sections.

Widgets Widgets should be used either for offering shortcuts to service related content or helping user to start service related task. They are accessible through the widget space. Consider glance-ability and very simple interactions. Good widgets examples are RSS feed readers, weather information, stock market tickers, or music player controllers.

Standalone Applications Native Applications may not provide all functions a service may need. If that is the case, or if the intended experience deals with more complex features, it is easier both for the service and especially for the user to experience it in a separate application.

Core Frameworks and public services There are various platform-wide functions that are available to all 3rd parties such as usage of accounts, notifications and content frameworks. Also applications, such as contacts, provide their own public interfaces to be used in 3rd party integration.

There are benefits for both integrating to a core Application and standalone application. The balance relies on how much connected to other experiences vs. how immersive (yet, isolated) a service wants to be.

Integration to Native Applications

Integration to applications can happen in three different levels.

- Plug-ins
- Offering Content
- Extending Content

Each application defines to which extent these three different levels are supported and how much extensibility there is in each asset.

The following pages define the basics of the three extension types in the MeeGo 1 platform. The extent of which the different types are supported in different applications is represented in the overview chart that follows.

Plug-ins

The most powerful way of integrating services is with plug-ins. If any of the platform’s native applications provides functionality or content that fits to service’s offering directly, an application plug-in can be created.

From the user experience point of view, it is better to aggregate all the same kind of functionality into one single experience (such as no multiple e-mail, but all in one place). A single plug-in, which only handles the content transfer between the service data cloud and the device, can be much easier to create than a full UI for service features.

Application plugins can contain possibilities to add limited branding elements to the application plugin itself.

- Email accounts
- Instant messaging account
- Calendars

Offering Content

Services can offer their content straight from a native app. This provides optimal content discovery as well as allowing for interesting cross service mash-ups (such as a user can make a slideshow of content coming from different sources). The whole native UI structure can be used, including navigation hierarchies and players.

Most of the time content will appear as a drill down option of the native application, controlling the Subview structures from that point on. It is important to consider the behavior of the native application to avoid conflicting or unpredictable user experience. Never use the navigation buttons as an area to create new extension links.

Some examples:

- Internet radio station from music
- Facebook photos from gallery

Extending Content

Extensions are ideal when a service wants to provide extra information about content shown in native applications. An extension may be a single UI element embedded to native application views or even a complete Subview in the application. Object views are the perfect example of where to add extensions.

Even if a MeeGo application cannot manage to show all service related content with native UI components, with minor modifications that can be achieved.

Each native application has defined the extension points for 3rd parties. These extension points are illustrated in detail by application in the overview page.

Some examples:

- Lyrics to now playing track
- Concert tickets for an artist

Case Study: X Service

An example integration could look as described in the figure. A video service provider has created software for MeeGo. The user can send videos from the device to the service. When the software is installed the following assets are installed to the device:

The service account plugs in to MeeGo’s account framework, enabling a centralized single sign on and account settings.

The sharing plugin (using the sharing framework) enables sending video to the service.

Service sub-views (offering content) in video application enables browsing content from the service directly within the native applications.

The native video player is used to play videos from service (relying on the Video Application plugin).

In the Music Application, the service extends content by adding a 3rd party extension to a details view of a music track. It presents related videos to now playing song.

Integrating Native Application Functions

As seen in this section, it is easy to get services into MeeGo without much work. Extentions allow 3rd parties to focus on their business and leave the hard work for MeeGo’s native applications.

However, if a service opts to build its own full Application, it is still possible to use all core frameworks of the platform. Once again, that means less development, leaving MeeGo to handle the functionalities.

A phone number in a restaurant app, for example, can link straight to the call application, and so on. Notifications can give user notifications of different types of events in the service or application. Accounts and SSO can (and should) be used for storing account credentials and managing user settings for each account. Content and tracker database is the key dependency for providers that need centralized data storage for their services.

More on how to get support from the framework in Framework Feature.

Localizing Services

Localization of 3rd parties in principle has to be done by 3rd parties for the following UI elements:

- Client applications
- Widgets
- Application sub-views (offering content)
- Application extensions (extending content)

However, when service integrates to platforms frameworks or applications with plugins, service does not have to create its own UI and all localization is done based on platform localization. Also if common components with common texts are used those are localized by the platform.

Overview of Extensibility in Core Applications

To Follow

Designing Your Application

Design Tips

Keep it simple and easy to use

- Pick one problem to solve with your application and solve it well
- Make it obvious what your application does and how it works
- Be brief, succinct and sparing with layouts, functions and copy
- Design your application from the top down, with the most important information at the top
- Consider ergonomics and the size of people’s fingers when laying out your application and designing controls.

Use scrolling carefully Part of helping people understand your application and what it does lies in making scrolling predictable. Only scroll repeating content or UI elements like lists, don’t hide distinct functions below the fold line because people may not understand they need to scroll to find that feature.

Keep it consistent with core components Where possible, use MeeGo common components in your application. This helps the user apply what they learn from other MeeGo applications to yours and gives them a smooth, consistent experience as they navigate around their device. It is mandatory to have ‘Home’, ‘Menu’, and ‘Back’ in all views, the only exception is in full screen mode available in application-like media players.

Think outside your app When designing your application, consider what journeys will take people out of your application and into another, or vice versa. Optimize your design for the reality of multitasking.

Strive for iconic character Each view in your application should be uniquely recognizable and understandable, even when it is scaled smaller in the switcher, to help people find it and reopen it when they are multitasking. Applications should celebrate its unique content and functionality in a way that emphasizes the iconic quality of the application, without compromising usability gained through use of consistent components.

Application Types

Before moving on with your application design, it is important to identifying the nature of the task your application is dealing with. This should be the main drive behind the proposed user experience. Think what is the focus of the application and what are the motivations for someone to use it. Understanding the nature of your application also helps organizing and choosing how to display information and interactions.

Even though it is impossible to classify each, as it would be impossible to identify every possible application that could be created, the two types at each end of the spectrum are productivity and immersive.

Productivity If an application deals with more pragmatic actions, such as sending a text message, it is described as a productivity application. These are usually tasks where efficiency is key, not only in terms of performing the task but also precision on getting to it. A well organized drill downs structure and simple toolbars with key actions, are good examples of how to present a responsive and quick task.

Therefore, these applications usually rely more on common components to build their UI, as simplicity is preferred. This is not to say that such couldn’t achieve an engaging experience. The calendar application is a good example, its first view presents a clean yet iconic calendar view, a simple list of upcoming events and tabs to quickly hop between the main ways of viewing the content.

Immersive There are applications offering more entertainment-based or visually rich experiences, like playing a video or browsing a map. Full views are quite often used in such cases. Although connecting to core behaviors of MeeGo’s UI helps users feel in control when stepping into a new environment, custom-made layout or component might provide a more appropriate experience to what is being offered.

Games are the perfect example of immersive applications. Users expect to be entering a ‘new world’ as a new game is launched. Whether they have a simple structure or complex, it is down to the designer to decide how to represent content, navigation, or interaction.

Application Basic View

Even though Application may appear in many formatting options (such as full screen mode, grids, etc), most views follow a basic structure.

Status Bar The main use of the status bar is to show signal strength, time and battery life (operator optional). In addition, it supports notifications (see notification section). This bar may be removed in specific cases.

Title Bar Containing the homescreen button, title, View Menu (opened from the title) and the close or back button. Each view has a title specific to the view. In other words, most of the time the title gets update as the user navigate through an Application (such as drilling down the music library updates the title to ‘Artist’, ‘Albums’, and so on). This bar may be removed in specific cases.

Content Area Where most of the action happens and the layout gets dynamic. Content area can present navigation options, text, media, etc. Actions are not limited to View Menu and toolbar, they may be presented together with the content as well. Other types of actions, such as media control buttons, can be presented together with the content. Please refer to the Common Components for all button variants.

Tool Bar/Tab Bar Optional fixed bar available to an Application whenever core commands are needed at all times. It is also possible to use it for navigation options. The bar is limited to 4 actions. In landscape, this bar moves to the top, placed together wit the title bar. Even though either icons or labels can be used (but not both at the same time), the former is recommended due to localization issues.

Backstacking Behavior

Rules for backstacking: 1. Toplevel views have Close, subsequent pages Back An application's home screen (that is the screen shown when it is launched for the first time) will have a close button in the top right corner, following views have the back button. When the Tab Bar pattern (see page 38 for more details) is used, each of the top level views reached from the Tab Bar are treated as top level pages and also have the close button.

2. Returning to top page clears History Whenever the user returns to a top level page Close is present on that main view and the back stack is cleared at that point.

3. Back always returns to previous view Without exception, the back button should always take the user back to the screen they were just viewing. It should not take them to some other screen in the hierarchy. For example, if the user links from the top level Music page to the detail view of a specific album, then hits the back button, they should go back to the top level page NOT to the category that album belongs to. In this way, the backstacking model is historical (that is, takes the user back in their history), and not hierarchical (that is, always takes the user up a level in the application hierarchy regardless of whether or not they arrived from that view in the hierarchy).

4. Completed subtasks are excluded from backstack history In cases where the user completes a subtask, the views within that subtask should not be included in the backstack. For example, if a user goes through and completes a purchase process, then proceeds to another page, then uses the back button, the back behavior should skip over the purchase screens that were completed.

Multi-tasking

Multi-tasking is at the heart of the Meego platform allowing users to quickly switch between applications. As part of the multi-tasking model, applications can easily provide links and functionality that opens a second application. The behavior is as follows:

1. Applications are stacked When a second application is opened it is stacked on top of the previous application creating a viewstack. The view will have a close button and not a back button in the navigation. Closing this application will reveal the originating application "behind" it. This functionality is maintained even if the user navigates away from the first screen and then uses the back button to return to the first view which has the close button. This behavior goes away at any point that the user accesses the switcher by pressing the home button.

2. New application creates new window in Switcher If the user does press the Home button, they will now see both the originating application and the second application in the Switcher. Both will remain until the user closes one or the other of them using the Close button on the top level page or using the close function on the switcher. Closing one of these applications does not cause the other to be closed.

Please see the next page for a discussion of special considerations related to deeplinking from one application to another.

Deep Linking From Widgets, Other Apps, and Notifications

In most cases, when your application is linked from another application, accessed from a Notification, or linked from a widget, it will be a deep link. In other words, the user will open a view which would only be found a few levels down in the application.

1. Deeplinked view shows Close Even though it is not the top level page, the navigation element shown should be the Close button and not the back button.

2. Provide navigation away from view Because the user will not have a history to navigate via a Back button when the view is deeplinked, it is recommended that applications provide a means of navigation back to the top level for those views that are most likely to be accessed as deep links. Common examples would be opening an email or a chat thread from a notification, opening an image attachment into the gallery, or linking to a map from a Calendar event.

3. Two options for backwards navigation If your application has strong use cases where a user would want to move from the deep link back to the top of the application, provide an explicit link back to the top.

If going to any other view is a more secondary use case, place a link back to the top in the View Menu. This link should always read "Go to <app name>", that is "Go to Gallery".

4. Behavior of backwards navigation The link should always return the user to the top level page of the application unless another instance of the application is open in the Switcher. In this case the link should open that view in whatever state it is in, and also close the deeplinked view.

Launching Multiple Windows

For some applications it is permissable to open new task windows from the originating application. The guidelines are as follows:

1. Only used when absolutely needed</br> Launching multiple windows is discouraged except in cases where very strong and clear use cases exists for multi-tasking within the existing application. Some good examples are as follows:

Chat applications - chat applications can launch separate windows per chat. This allows users to open communication with multiple persons and manage those conversations via the task switcher.

Email - in order to support the functionality of composing an email in interrupt mode, i.e. being able to easily switch back to viewing the rest of the inbox and also while switching to other applications, email programs can spawn separate Compose views.

2. Window stacking and behavior in the Switcher The model for spawning new windows from a parent application is the same as what is described for launching one app from another. The navigation element should be a close button and not a back button.

3. Deeplinking considerations and backwards navigation These types of windows are also likely to be windows which may be deeplinked from other applications, notifications, and widgets and should therefore include some sort of navigation back to the top of the application. As noted on the previous page, the link provided should open the originating window and maintain the view already in that window or if it is not open it should spawn a new window for the application. It should NOT take the user back to the top level of the application within the same view. This is to prevent multiple instances of the same view open at once.

Re-opening Windows/Window Merging When Deep-Linking

Special considerations exists when an application that is already open in the Switcher is then deeplinked from another application, a Notification, or linked from a widget.

1. Deeplinked items should open a new window Rather than overwriting activity that the user may already be doing in an open application with the new view, another window with the view should be open when there is already a window open. For this reason it is strongly recommended that applications likely to be reached by deeplinking are written to support multiple windows.

2. Use window merging to take care of minimizing the number of open application If the user then accesses a “Go to <app name>” link as described on Launching Multiple WIndows, this should open the already opened application window. It is left on the view that was already open, not another and the deeplinked view is closed.

3. Avoid situations where multiple windows with the same view are created As a general rule navigation around to views directly related to the item being viewed that then return to that item upon completion are not problematic and can be provided from the deeplinked view. Navigation to other areas of the application that do not explicitly check for open windows should be avoided.

4. It may be necessary to create two versions of the same view The version which is reached through the parent application can have full navigation to other areas of the application, while the version reached by deeplinking would only have simplified navigation that takes user to the top of the navigation or reopens already opened views.

Navigation Templates

Applications may vary in navigational patterns. Three templates have been created to help support the Applications main functional goals.

Drill Down Most scalable template, used when access to all Application navigation structure is needed on the top level. Layouts are not restricted to lists.

The close button gets replaced by a back button as the user drills down the structure.

Such as: Phonebook, Gallery, TV & Video.

Navigation in View Menu For flat experiences, prioritizing content over navigation structure. View Menu used for hopping between navigation options.

This allows for many Navigation options, as oppose to the Tab Bar one. As all this navigation points work as the top level of the Application, the close button remains even after a navigation change. Note that Filters behave differently, even though they too are placed in the View Menu. Filtering works as a drill down, hence the Close button gets replaced by a Back button.

Such as: Switching between Messaging accounts.

Tab Bar For quick access between distinct areas or modes. Be careful about mixing actions (like “Create New Message”) with navigational links in the Tab Bar. Use buttons within the content area for actions if necessary in this layout.

This models is limited to maximum 4 navigation options. Navigation options in an Application should not change over time, hence do not use them to accommodate new installed Extensions.

Such as: Call App, Alarm, Calendar.

Combining Drill Down with Tab Bar

The combination of Drill Down and Tab Bar might be necessary for more complex structures. The main reasons to do it is when top-level links break down into multiple sub-links and still need to be accessible at any point of the journey. To avoid independent confusing journeys for the user (which also clashes with the task switching model), there are several key rules when combining these two patterns:

Highlight Selected Navigational buttons are always highlighted throughout the journey to help support the user.

History clearance When a user navigates to a top page, backstacking is cleared. A smart way to connect to the lost history is adding a link to “last viewed”.

One hit to top level Clicking the Nav button takes the user to the top level, even in sub-pages.

Interacting at the top level In some cases, the user might interact at the top level but not actually drill down. Those actions are not cleared when hopping between Nav links. Examples are search and log in form: search result is not erased, nor user is logged out. Moving across dates in a calendar is also not a drill down, so whatever date the user has in a tab when leaving it will remain there.

Bar is sticky when Application is closed Applications will always be opened in their last Navigation state, always presenting the user with the top of the Nav link.

For those reasons, this model works best when there is no or very flat Subview navigation inside the tabs.

Recommendation on Layouts

As seen on the Application Types section, not only how you organize but also how you present your layout will have a major impact on the experience. While a list presenting content maybe suitable for a pragmatic task, other formats might be more engaging for other applications.

Whereas navigation models help organize how to structure an application, they do not imply any format or layout of how to present it. Drill downs are the perfect example: it serves as the model for several applications, such as a phonebook, music library and gallery. These present different types of content and, therefore, an array of distinct experiences.

Here we present two options of how to format a gallery, both using drill down as navigation structure. The Application does not present a long list of navigation options, and has images as its core experience. A grid, presenting items in the top level of the navigation helps make it more engaging as well as providing quick access to the latest content from the first screen.

Consider the type of content, how complex the structure is and explore the screen real estate to its Max.

Other Application View Examples

These are just a few possibilities of secondary key screen types. The latest layout variations, including precise measurements, can be found in the Common Component Library.

Applications can always create their own layouts. Whether templates or customized screens are used, it is advisable to use common components. It makes development easier and brings coherency to the user experience.

Portrait vs. Landscape

Device orientation will suite multiple Applications in different ways. While watching a video is best done on landscape – with the media taking up all the screen, scrolling long lists is more comfortable on portrait.

MeeGo common components are always provided for both orientations. In other words, by using them, there is not need for extra work. It is advice able, however, that anyone building an Application acknowledges such change, as it might impact the desired experience.

Applications can overwrite default changes. However, it is highly recommended that all Application provide both portrait and landscape mode. The main reason is the form factor of some available MeeGo devices. In a devices with a slide hardware Qwerty, the landscape experience becomes key, and somehow extended. Therefore, the user shouldn't be forced to change orientation to complete a task. Playing a game is one of the few exceptions to this rules, as its experience flow does not require (in most cases) the use of any other application.

Changing views should not introduce new or different content or functionality. Assets can be subtlety resided and/or reshuffled to make better use of the screen real estate.

1 vs. 2 Columns

Lists are probably the most common UI layout, and have a particular behavior depending on the orientation.

Whereas portrait allows for more items to be presented, landscape presents the possibility to extend the content area of a particular item. This is specially useful for text strings, which in landscape get more character area. The title of a message, for example, has less chance of being truncated.

Some Applications may consider splitting lists in two columns when on landscape to optimize space usage. This may be used when there is no hierarchical order, usually in first level drill down pages. However, this should never be done when the list follows a specific order criteria (such as chronological, alphabetical, etc).

Buttons should always be centralized (specially important due to two column split) and, full width in both orientations.

Device Content

In MeeGo, content is king, and experiencing it from different sources, whether if it is Online or Offline, or if it is part of the user's library or not, should be seamless.

However, sometimes it is important to express those what makes them distinct, specially if the interaction available according to the source or where the content is different. An opened picture from an MMS still uses Maemo's image viewer, but it is different from what is in the user's gallery. The same is true for a vCard received in a message: opening it is different than opening a Contact Card already in the Contact List.

Consider what is the most appropriate options depending on the objects. If it's a media preview from a store, purchasing could be as a top level option. For received content, Saving to the device library is a highly recommended command (such as picture, music, contact card, etc), as well as Sharing. By saving, the object becomes part of the device and, subsequently, the options are updated with the correspondent local ones.

Settings

Always consider designing the Application settings. User accesses settings from the View Menu in your application. The Settings view presents Home, Title and Back (View Menu is not necessary).

In some cases, it might be desirable to provide quick access to specific settings of an Application sub-views. These will be a corresponding sub-view of the Application global settings. For example, in a subview of Messaging Application showing a specific account, Settings could take the user to the sub-settings of that Account. However, there is a navigation difference in such cases: if the sub-view of Settings is accessible from the correspondent sub-page, Back would take the user back to the page where it was opened from, and not Back to the Settings top level.

In the settings view, group similar settings (by using visual dividers) together to help the user find related options. Each group can contain multiple items, such as texts, images, and buttons.

If the user can change settings directly on the UI, do not include those settings in the Settings view. For example, if the user can set a clock alarm directly in the Clock Application main view, do not include the alarm setting in the Clock settings view.

Note that when the user changes the settings, the change takes place immediately, with no need to Save changes (more on Confirm vs. Canceling section). You can use the Undo button in the view menu to discard any changes, and reset all changed settings back to the previous states. The Undo button is shown only after the user has made changes.

Navigation within your applications

To Add

How to structure commands (embedded vs. object menu)

To Add

Theming

Design Tips

To Add

Framework Features

Relying on the Framework

This section describes framework features that are extremely powerful to proceed with design. It declares the minimum information, and is not intended to be an exhaustive catalog of all possibilities. In addition, it does not describe all these feature in detailed technical terms; for that, individual guidelines and system specifications can be found separately.

Using these features avoids having to reinvent the wheel, speeds up development effort and results in products that are more predictable for users. Even though some are presented with specific Application examples, most of them are customizable to any kind of content or interaction an Application may want to provide.

In addition to this list of features, MeeGo offers an extensive library of common components. A common component packages both visual and interaction design into a UI block designers can use when building their application. Examples are buttons, sliders and labels. Please see the appendix for a complete list and detail of common components.

Accounts

The user has accounts to connect with various online services, such as e-mail accounts, instant messaging accounts, and service accounts.

The account management is centralized in the device: there is one place where the user can add and manage all the various online accounts used in the device applications. Each account can have a separate page for settings, if needed.

Applications can create setting views that list only the accounts that relate to the application, for example, E-mail can have an account view that displays e-mail accounts. Editing the account should take place in the same settings view, no matter where this account is accessed.

If your Application deals with online content sources, use the common Accounts framework instead of creating your own solutions for handling accounts.

Do not create your own account settings UI within the application, but rather create the views as account settings and link with the UI to the Accounts UI.

Accounts can also be used for single-sign-on purposes, that is an account can enable automatic log in to certain web sites in the Browser.

Pickers

Certain framework Applications can provide shared UI views for all other Applications installed in the device. Pickers provide a common style for an application when selecting a task.

For example, Content picker gives Applications a mechanism to select one or multiple files from within the device file system. This provides the general “File open dialogue” functions. People picker allows users to select one or multiple contacts from the device contact list. Location picker lets users specify an address / real-world location on the map. Location picker also allows selecting an address on the map data provided by the device.

As far as navigation goes, pickers act as filters in the sense that they do not get added to the history. Users enter pickers and step out of it (either by selecting content or pressing back). When a picker view is opened and the home function is selected, the picker remains present in the task switcher.

Available pickers:

- Contacts
- Content
- Date
- Location
- …

Status Bar and Status Menu

The Status Bar is the vertical box at the top of the UI. The status bar can show a few different bits of information. The main use of the status bar is to show signal strength, time and battery life (operator optional).

The Status Menu appears by dragging down the Status Bar. It contain core functions that should be accessible throughout the UI. The Menu does not push down the content, but appears as a layer on top of it instead.

The Status Bar also supports notifications. It displays

Another use of the status bar is for the notification Soft Events to help remind the user of specific events/alerts. Please refer to the notification section for more information on soft events.

Notifications

Use notifications to give the user information without blocking the UI or forcing the user to switch to another view. Note that MeeGo UI provides connectivity and system information automatically. For example, the system provides notifications about lost connection, and low memory, so do not define notifications for these.

Note the following:

- Use notifications for unexpected events.
- For ongoing processes, do not use notifications for constant updates, but other views (such as the Notification Drawer) or Dialogues instead. Notifications might be used to report concluded processes.
- Do not allow the user to select unavailable functions.
- For actions that can be harmful or need immediate attention, use mainly query dialogues (queries have tones).

The main form of notification are Banners, which can be Information (such as download completed) or Event (such as new message) related, and interactive or not. They appear on the top part over the UI. Dismissed incoming Event Dialogues also move to the Drawer (such as New Message). It is important to note that other dismissed Dialogue, such as battery low alerts, do not move to the drawer.

Banners are an easy way for the user to monitor updates. A good example are social network updates (comments, new messages, etc). However, It is important for the application to allow for the user to switch this function on/off inside its settings; the user must have the ultimate control over what to receive.

Notification Drawer

The Status Bar displays modifiers representing the Events that happened. Each type of notification has its own icon to help the user identify the missed event at a glance.

The Notification Drawer is accessible by dragging The Status Bar, and it is displayed above the Status Menu. Banners stack up in chronological order. At the end of the events, a clear button allows the user to dismiss all listed notifications.

The Notification list behaves as follows:

Single Event Single event notifications are displayed as single item, consisting of a two lines text accompanied by an icon. Consider what is the primary and secondary information for the correspondent event.

Multiple Events Multiple event of the same type are aggregated into one line. The number of events becomes the primary information.

The Notification Drawer also supports tracking on going activities, such as file transfers and Application installing.

Presence

Presence lets the user publish their current availability status to known contacts, enabling them to tell other people beforehand if they are willing and available for communication within that specific service. Examples of presence supporting services include Skype, Gtalk and Ovi Chat. Certain services also allow adding a textual status message and/or location information to the basic presence information. Certain services may support custom service specific status states (such as “Out to lunch”) and/or additional features, such as service specific avatar pictures.

The device can connect to multiple services simultaneously and provide presence handling for each of the services separately. The presence setting (online/away/offline) affects whether the user receives communication and messages from the service. The device status area displays a global presence setting, combining the status from individual services into one meaningful combination. The user can control the presence status from the My Profile view, available from the device status menu.

The device also shows presence information of the contacts within the device Contact book. Presence information is always shown whenever the contact is listed and if the user is online.

Sharing

There is a common framework for enabling file sharing. Sharing provides means to share both to supported online services (such as Ovi, Flickr, and YouTube) and through the various communication mechanisms in the device (such as E-mail and Bluetooth).

Applications and services can register to the Sharing framework, telling what kind of content types they are able to share.

When the users initiate sharing, the user flow in all the various cases is the same. First the users select the files to be shared, and in the first screen of sharing they select where to share to and in the second screen they set the options for the selected service/method. When sharing is initiated, the Transfer UI is used to display the file transfer progress.

Only files can be shared. For example, if you want to enable sharing a bookmark, create a plain text file that contains the bookmark as a text string, and share the text file.

It is also possible to share multiple files at once (such as more than one music track). In such cases, the Sharing command is usually placed inside the View Menu, as it is an action that will act upon a view with multiple objects. The interaction follows the multiple object rule – see Applying Actions to Multiple Objects.

If you want to provide sharing in your application, use and extend the Sharing UI capabilities, do not invent your own UI and solutions to perform sharing.

Content Manager and Search

All the content in the device Applications is stored in a common content tracker database. The device provides a Global search UI for searching for content items by using various search criteria such as keywords, tags, and date ranges. The search UI is common for all applications.

You can also create your own custom search in your Application by querying the tracker system and drawing the application's custom UI's for the data that the tracker provides. Most views that Applications draw are basically the result of a query to the content tracker database, with certain parameters. The global search UI does not currently support custom search result interfaces.

Applications can also launch the Global search UI with certain Filtering criteria, for example, to find content items that match only the content items supported by the application.

If your Application creates new files, register your file types and use the tracker database to store your files to ensure that Search is able to find data in your application. In this way the users can use their learned patterns for file management and use the content picker to select content items from your application.

Online vs. Offline

Global search display online results in addition to the local ones. However, unlike extensions in native Applications, adding sources to global search is a more laborious process and has to be done by the MeeGo platform, following technical and business requirement.

Pre-installed sources include Nokia content, such as Ovi Music, Maps, Stores and a few partners (to be determined for each MeeGo release). These sources can be managed under Search Settings View, including the possibility to turn them on and off.

The mechanics of results for online sources may vary and is up to the source which criteria to follow.

Furthermore, their content does not need to be indexed in the tracker. The search gets the result directly from online services. That means it does not have to be specifically content based. An example is displaying suggestions based on search strings, which then links to an online search result from the specific Application.

Search by default, will always display online results. Under the Settings View, the user finds switches for both "online search via phone network" and "online search via wlan", allowing for data cost management.

Tagging

Hierarchical file system is not exposed to the users. Therefore, to be able to manage large amounts of data, users can add textual tags to content items to categorize content. Tags are stored as metadata to content items. The global search uses the tag information to give search results.

Applications can use a tag cloud UI component to display a range of used tags. The tag cloud can be shown either inside a particular object, to show tags assigned to that particular content item, or inside an Application view, to allow the users to find all content items assigned with a particular tag.

The tagging UI also provides UI views to allow editing/removing tags to a content item. Tagging widget can be either in the consumption mode, where clicking on a tag displays items, or in the editing mode, where clicking on a tag toggles the tag on and off.

Certain Applications allow marking content items as favorites: favorite is basically just a tag, although in the UI it can be given a higher separate status.

Favorite

Favorite is a tag that can be set to any content item. The UI represents this tag as a specific symbol (the star symbol). Certain UIs can provide a quick option (for instance in the toolbar) to set this tag on and off. This function is specially important for Applications with large amount of content, such as Music and Gallery.

Applications can provide quick option to display content items that are tagged as favorites, for instance a drill down option in the main view or as a navigation option under View Menu . This represents query to the content tracker system; the output format is set by the Application (such as a grid of favorite pictures or a list of favorite music tracks).

Transfer Manager

The UI framework provides a common mechanism for indicating and controlling file transfers. While file transfers are ongoing, the status bar shows an indication of activity, and the status menu contains an item to access the Transfer UI.

The Transfer UI lists all ongoing transfers. Clicking on an individual transfer gives a detail dialogue for that transfer, allowing the user to control that transfer.

Use the transfer manager to display the status of the ongoing file transfers. Do not invent your own solution for displaying ongoing file transfers.

Note that the transfer manager is not a global progress indication system, that is, it should not be used to indicate progress for tasks that are not file transfers. Note also that not all data transfers need to be displayed in the Transfer UI. For example, if the E-mail Application is downloading e-mails, this should not be indicated within the transfer UI.

Other Considerations

Localization

All UI texts in MeeGo UI are localized to various languages. Keep in mind that terms are significantly longer in other languages compared to English. As a rule of thumb, reserve at least 30-50% more space for language variations. Common components that support text will automatically truncate once the character limit has been reached.

Localization has also a big impact on layout, the main one relating to language direction. While English and other western languages read from left to right, many other ones read in the opposite direction. When supporting a language, MeeGo also includes their different rules in its common components (most of them being bi-directional).

An example is the support to Arabic. The basic rule is that everything in the UI is mirrored, including vertical sliders and progress bars. Again, this is all taken care by the common component library. Please verify the development tools library for more specific details on the latest available support.

Write the initial UI texts in English, which is called ‘engineering English’. The final wording and terminology is designed by Localization in cooperation with Language and Terminology Department (LTD) and Brand Architecture team.

Localized English is created based on the engineering English and a short description. Therefore, follow at least the following text style guidelines already in the engineering English:

1. Use clear and concise language.
2. Avoid abbreviations (some languages, such as Arabic, don’t usually allow abbreviations to be used at all).
3. Note that most languages need more space than English.
4. Do not allow automatic truncation with localized content.

- Abbreviation is better than random truncation.
- Automatic truncation is allowed only with variables and texts not provided by localization.

1. Do not use ellipses (…) in progress or wait notes. A progress bar or other animation is shown to indicate that an action is ongoing.
2. Use capital letters only in sentence initial position of texts. Do not use casing to indicate functionality or emphasis as not all languages have upper and lower case.
3. Use numbers in texts – ‘1’ not ‘one’.
4. Avoid personal address.

Connectivity

Use the heartbeat feature for connectivity. Heartbeat offers constant intervals for Applications to communicate with online services. The updating frequency is different depending on Application requirements and how actively the user is using the device.

In some situations (for example, data roaming), the system prevents the user from using unnecessary or high-cost connections, so mark each connection attempt as “user interaction” or “Application initiated”, depending on the situation. For example, the user taps a bookmark in browser, the connection is established, and the web page loads immediately. Or the E-mail wants to get new e-mails automatically from the server, so a connection is not established and no notifications or error messages are shown.

Also note that there are times when the device is offline, for example, in an airplane or a hospital. In the offline mode, the cellular network is unavailable.

For situations when there is no connectivity, note the following:

- Do not prevent opening the Application or interaction with the application. Applications that cannot meaningfully interact with users

without establishing a connection (such as Store) should launch even if there is no connectivity. While a connection is being established, use a progress indicator to inform the user that the Application is attempting to retrieve data.
- Do not show any notifications. The system provides all needed notifications when user tries to do something that requires a connection.
- Show locally stored data in the application, if possible. If data is created, modified, or Deleted while offline, these changes should be updated as soon as the connection is established again. For example, when the user creates a new high score while playing a game offline, the high score should be updated when the Application reconnects.
- For Internet, provide a refresh-button in the Toolbar or in the View Menu : tapping the button starts a new connection attempt: either connection succeeds or a notification is shown.

Define in your Application when a connection is not needed anymore.

Power save mode The most efficient way to save energy in the device is to turn all Application initiated connections off. This can be done by switching to power save mode manually, or activating automatic switch to power save mode when the battery level is low. In the power save mode, only user interaction has a connection.

All Applications flag user interaction (to differentiate it from Application initiated connections) when they request Internet connection. Only flagged requests get Internet connection.

Crashes and Errors Handling

As the designer, you are responsible for specifying the error messages in your applications. When designing the application, list the different error cases and error messages. Think if you could prevent the error completely by making changes to the UI, for example. If not, create an error message.

Use information banners or dialogues for error messages. The main difference between them is that information banners disappear after a time-out. If the user really needs to notice the error or do something to correct it, use a dialogue.

When you write error messages, think about the user. Consider the following points:

- Avoid engineering language.
- Try to help the user to realize what is happening and what the user can do.
- Focus on introducing a solution instead of stating the problem
- Do not prompt with obvious texts. Think what is essential information to the user to be able to proceed: avoid using notes like “Unable to save document. Try again?”
- Design positive paths to accomplish tasks.

Use information banners to provide feedback in error cases, for example when a mail recipient is missing (but not when the message field is empty).

If an error is a result of leaving mandatory fields empty, note the following:

- If user edits any of the fields, the file is saved and mandatory empty fields (such as name) are filled with content from another field (such as URL in the case of bookmark) or with predefined general name.
- If the user leaves a mandatory field empty and it cannot be filled with any predefined text, an information banner is shown to the user after the user has selected the action button, and the field in question is highlighted
- If user leaves some mandatory fields blank after editing the details and closes the view, the previous values are restored, but consider carefully which fields are really mandatory.

Appendix

Common Components

Font Style Sheet

Tool Bar and Tab Bar

Title Bar

Buttons Standalone - Single Label Button - Multiple Label Button - Icon Button - Icons with Label Button Group Buttons Tool Bar Buttons Tab Bar Buttons Disabled Buttons (Info Banners)

Lists List Items (single label) List Items (multiple label) List items (with thumb) List items (with controller) List Divider

Object Menu Align to single selection Dialogue?

View Menu List Item Dividers Sub View Items List Item with Controllers?

Dialogue Single Selection Dialogue Query Dialogue Extended Query Dialogue Popup Lists?

Controllers Checkboxes Switchers Sliders

- Volume Bar
- Seekbar

Scrollbar

Progress Indicators Known Data UnKnown Data Spinner

Text Fields Single Line Field Multiple Line Field Field with Image?

Notification Notification Banner Notification Drawer item

Important to Note This list is being built together with the MeeGo Common Components Team and is not yet final.

MeeGo is designed for finger usage and is fully scalable for different screen sizes and resolutions. For example, if a common component has a 8mm height, it will have that same height regardless of the screen physical size or resolution.

Sizes and final specification for each component will be presented in upcoming versions of this document.

Push buttons Use a push button to start an action, for example to call a contact selecting a call button in a contact card. Add a descriptive text label for a push button, and align it to left, right, or center.

Switch Use switches for functions that the user can set on and off. Place a descriptive text label next to the switch button.

A Switch should never be used as a link. Switches only do one function within a list and if there are some actions needed for that switch, they should be displayed alongside the asset.

Checkbox A Checkbox permits the user to make multiple selections from a number of options. A caption describing the meaning of the check box is shown adjacent to the check box. Inverting the state of a check box is done by tapping the box.

Icon button Icon button consists of an icon and, optionally, a text label. Unlike push buttons, the icon buttons do not have button-style edges, so they look like icons. An example of an icon button could be used in the toolbar or Tab Bar.

Slider bar Use a slider for continuous set of values. Define a descriptive title for the slider. Displaying thumb value (value box shown once user slides), minimum and maximum values is optional.

Seekbar Seekbar is a special type of slider used for displaying the playing status of a multimedia item. If you want to display gradual steps, define slider steps. Displaying minimum and maximum values is optional.

Progress bar - Known Data Use progress bar to indicate actions, such as uploading and downloading, when there is enough space available and the duration of the process is known. You can place the progress bar also on top of other UI components. Use progress indicator to show the user the current status of a process, when the process may take longer than two seconds.

Progress bar - UnKnown Data When duration is not immediately known, the progress bar can be temporarily in the unknown duration state until the duration becomes known. Do not indicate automated updates. Progress indicator has two variants, spinner and progress bar. Avoid using both variants in the same view.

Spinner Use spinner to indicate actions such as scanning or refreshing, when the available space is limited and the duration is unknown. It is also possible to use the spinner for known duration if the available space is very limited. You can place the spinner also on top of other UI components in the header row of a dialogue, container, or view menu.

Single Selection Dialog Use dialogues when you want immediate response from the user. Dialogues block and dim the background of the view, giving focus to the dialogue. Unlike notifications, dialogues do not disappear from the UI without user interaction. Keep the dialogue text short and simple.

Query Dialog The difference between the single the query dialogue is that the user can close the single and multiple selection dialogues without selecting an option, but not the query dialogue. Query dialogs should be used when deleting files, recovering from errors and for operations that can be harmful to the user.

If the dialogue has action buttons, include a button for canceling, as well. In order, from left to right, top to bottom, place the confirming action button as the first option and the canceling button as the last.

Multiple Action Dialogue Box An Exteneded Query Dialog should be used when the platform requires information along with displaying images or warning icons. The Dialog box requires some form of action before the user can continue.

List List is a container of objects displayed vertically. List is the most common UI component: when you have an undefined number of items to be shown on the UI, consider using a list. The number of items in a list can change while the list is being displayed. If there are more items than what fits the current view, the list automatically becomes pannable. A list occupies the full width of the area in which it is being displayed.

Sometimes lists can support list Dividers which help break up the long list of content. These Dividers can be used to help speed up the user experience when searching for something in a long list.

Object Menu An object menu is a list of related actions users can perform with objects. Although the object menu is scalable, do not include more than eight menu items to avoid panning. Note that when the user installs other applications that have plug-ins, new options may be added to the object menu.

Toolbar/Nav Bar Use toolbar for actions that are relevant to the content in focus. In the landscape mode, toolbar remains in a fixed position on top of the UI and in the portrait mode it is on the bottom of the UI.

Include two to four most important commands in the toolbar. Negative actions always sit on the right hand side of the toolbar.

View Menu The view menu consists of a list of commands related to the current view (or Application). The list has no limit and will scroll if items do not fit the screen.

Overloading View menu is not recommended and discouraged.

Information Banner There are two variants of Notification Banner, ‘Non-interactive’ and ‘Interactive’. Use non-interactive information banners to display feedback and notifications, if you cannot display the information on other parts of the foreground UI. Use interactive information banners for notifications in error cases and exceptions, when you want the user to react to them.

ComboBoxes Open ComboBoxes consist of sub-component button and Dialog assets. A ComboBox Dialog inherits attributes and interactions from the Single Selection Dialog, such as if the height is bigger than screen this becomes pannable and position indicator is displayed.

A closed Combobox can take the form of 3 different layout.