Best Practice Guidelines

for producing high quality mobile applications

version 2.2 - 5 February 2013

The App Quality Alliance (AQuA) has put together some **Best Practice Guidelines** for producing a high quality application that works and fits well in a mobile device irrespective of the platform for which they are being developed.

new For version 2.2 we now cover **Network Utilisation, Efficiency and Battery Life Considerations.** In addition, a number of other areas have been augmented in line with recommendations from other industry players.

Some of the guidelines within this document are common sense, and others are less obvious, but they're all worth checking off before your app goes live.

The user outcomes aren't supposed to be rigid, and of course applications that are intended to be different by design will always be acceptable, provided that the user experience is not adversely affected.

These best practices are intended to be used alongside the platform specific Test Criteria produced by AQuA. Currently these exist for Android™ and for Java™ ME applications.

For other technologies, to use these best practices as a basis for testing an app, we suggest you use a device that has been factory reset before the installation of the application to be tested. This will ensure that there is a known base with only pre-installed applications and any errors will be attributable to the application under test.

These Best Practice guidelines will be updated on an on-going basis as a result of input from the community and changing platform requirements.

You will find links to extra sources of information, other associated best practices and mechanisms to feedback comments and additions at the AQuA web site at: www.appqualityalliance.org

We hope you find these guidelines useful.

Update History

Version 1:	March 2011 Initial publication
Version 2:	August 2012 new sections: Privacy and Network Utilisation.
Version 2.2:	February 2013 Update with Google Android recommendations and alignment with GSMA Smarter Apps for Smarter Phones and AT&T Best Practices

List of Contents

The Mobile Environment and Best Practices

Mobile devices are very important to their users, often a lifeline and usually carried with the user. It is vital to most users that they are reliable and available to receive calls and messages. In addition they typically have limited display areas, a keyboard unsuited to lengthy input, and restricted processing power and memory capacity as well as interruptible connectivity, and all that running on batteries that can quickly be run flat by high power consumption.

High Quality Apps are designed with all of this in mind, and as such enhance the device for the user. These **Best Practices** (and others) are intended to help app developers achieve that high quality.

Connectivity is a major difference between mobile devices and fixed computers and must be considered in the design of an effective application. In addition to Bandwidth issues, there are questions on latency and on the state of the connection. Mobile devices cannot safely be considered to be always on, unlike the emulator in the development test environment. The mobile's radio is dynamically controlled and shuts itself down when it isn't being used. This has a significant impact on the design of an application, for example the mobile device manages the status of the radio connection to try and minimise the battery use, and something like an ill-considered keep alive strategy in the application, for example, can cause huge battery drain out of proportion to the data being transmitted. The details of this are described specifically for **AT&T Application Resource Optimizer** (ARO) developer guidelines and in general by **the GSMA best practice documents** (see below).

The bandwidth of a mobile connection is also usually lower than a fixed-line connection, and the latency is higher. The user may be limited in the amount of data they can consume without incurring additional charges. And most importantly of all, the connection may vary in availability, capacity and latency, so apps must make allowances for the network to change as the app is being used. Changing data connections from WiFi to LTE(4G) to 3G to GPRS to none (perhaps in an airplane or out of coverage, or out of credit) as the user's situation changes must not cause the app or the device to hang, crash, or leave the user in a confused state.

PC-format devices have had many years to develop consistent UI layouts that are used across most applications on a platform, while mobile device UIs are still evolving. Some applications may experiment with unique UI layouts that are different to the common visual language of the underlying platform. This can sometimes mean that user expectations are not always aligned with developer intentions.

For all these reasons, it is important that mobile applications operate in a predictable and consistent manner, and overtly manage their consumption of data, system and network resources as much as possible.

Best Practices from other organisations

There are a number of organisations and companies producing best practices or guidelines that work well together. AQuA has been working closely with both the GSMA and AT&T to ensure alignment across the board.

GSMA: Smarter Apps for Smarter Phones.

This document can be found through the AQuA website at www.appqualityalliance.org or through the GSMA website at http://www.gsma.com/technicalprojects/smarter-apps-for-smarter-phones.
It provides some detailed technical guidelines on optimising the network usage.

It also explains the background and gives some detailed techniques across Apple IOS, Android and Windows devices that can be used to improve the efficacy of network usage.

The GSMA document includes guidelines for using asynchronous techniques, efficiency for network connections, security, apps working offline and in background, working with cloud services and other aspects.

AT&T ARO: Suggested Best Practices.

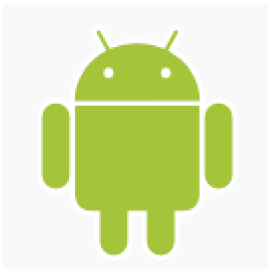
The AT&T developer program (www.developer.att.com) has published a set of developer guidelines that deep dive into details of network efficient apps, as well as details for privacy guidelines and more. This information to help deliver network efficient apps is backed up by a tool, the Application Resource Optimiser (ARO).

Links to both the information and the tool can be found on the AQuA website or direct at http://developer.att.com/developer/forward.jsp?passedItemId=200042

The focus is network efficiency, both in terms of radio usage optimisation, but also about caching and avoiding the re-transmission of data (for example images) that have already been transmitted to the device. To the user this translates to improved battery life and better speed of the application.

A high level view of the AT&T ARO advice is included in this guide.

Android.



The User interface section of this document, specifically the section concerning differing screen sizes and device formats, has drawn upon recommendations from the **Android Open Source Project Tablet App Quality Checklist** information.

This may be found in full at
http://developer.android.com/distribute/googleplay/quality/tablet.html

The AQuA Best Practices

Installing and Launching

Installation

The application should install from the intended distribution channel (e.g. over-the-air [OTA] from an app store), and the icon for the application should be found in the expected location on the device.

If the platform supports installation of applications to SD card, then an application larger than 10MB should offer the option to install there, unless core functionality is unavailable when installed to card.

Application Permissions

The application should only request the absolute minimum permissions that it needs to support core functionality.

The application must not request permissions to access sensitive data (such as Contacts or the System Log), or services that can cost the user money (such as the Dialer or SMS), unless related to a core capability of the application.

Long Launch Time

All applications should notify the user if there's going to be a long launch time. If the Application takes longer than five seconds to be ready for use, a progress bar or a message should be displayed to tell the user what is happening.

Memory and file storage during run

For an application that writes to file system, ensure that it correctly handles out-of- space exceptions during execution, and gives a meaningful warning to the user advising about lack of space when a file is trying to be stored.

Non- Blocking Connectivity

For an application using an HTTP network connection:

> For Java, Networking should take place in a separate thread, so as not to block other application activities and to allow the app to display progress.
>
> For Android, Networking should take place in a separate thread, so as not to block other application activities and to allow the app to display progress.
>
> For Symbian, it should be able to use simultaneous connections properly and the user should be able to control the connections through the Connection Manager.

Please note: Users might have two types of connectivity - eg WiFi and cellular - and the device might be switching between the two. The app must respond to this.

More detailed information on this can be found in GSMA Smarter Apps for Smarter Phones Section 2.2, and the technology specific chapters.

Network Utilisation, efficiency and Battery life considerations

An application must consider the system on which it runs - the network environment - and bear in mind that it may not be the only application using network features in the phone. By making the application well behaved in the device, you can achieve a stable and user-friendly application experience .

This section of the AQuA best practice draws on the AT&T ARO guidelines, and more details can be found via the links on the AQuA site.

HTTP 1.0 Usage

The Best Practice Recommendation is to use HTTP 1.1 with pipelining. HTTP pipelining parallelizes HTTP requests within a TCP session. It’s perfect for improving user experience in high latency environments, but it’s important to note that HTTP pipelining requires implementation on both client and server.

Multiple Simultaneous TCP Connections

It is common for applications on the wired-web to open multiple persistent Transport Control Protocol (TCP) connections so that different content can be served simultaneously from the same server.

Persistent connections allow the same TCP connection to send and receive multiple HTTP requests/responses instead of opening a new TCP connection for every request/response pair. When an application opens fewer TCP connections and keeps them open for a longer period, it causes less network traffic, uses less time establishing new connections, and allows the TCP protocol to work more efficiently.

If a wireless app opens too many simultaneous connections, performance will begin to decrease more quickly than it will in a wired app. As a result, the wireless app will appear less responsive, and energy will be wasted from the radio being turned on and off.

If an app opens too many simultaneous TCP connections, the size of each connection will be smaller. This means that throughput will be limited, performance will decrease, and energy will be wasted.

The Best Practice Recommendation is to manage the TCP connections in your application through using few TCP persistent connections or using HTTP Pipelining.

Few TCP Persistent Connections

Whenever possible, it is good practice to group requests together in order to improve performance, save energy and reduce bandwidth. Since a single request for more data is likely to provide a better user experience than several smaller requests, bundling, or batching up multiple requests at the application level is recommended. This can be done with persistent connections.

Persistent connections (also called keep-alive or connection reuse) are the technique of using the same TCP connection to send and receive multiple HTTP requests/responses, as opposed to opening a new TCP connection for every request/response pair.

In HTTP 1.1, persistent connections are the default behaviour of any connection. So unless otherwise indicated, the client should assume that the server maintains a persistent connection, the number of simultaneous connections can be controlled by closing connections when they are no longer needed.

By carefully managing the multiple TCP persistent connections in your application to keep the number of simultaneous connections to a reasonable level, you can improve the performance of you application, save energy, and reduce bandwidth.

HTTP Pipelining

Another technique for improving application performance is HTTP pipelining. If an application is running on a client that supports persistent connections, it can “pipeline” its requests.

Pipelining is the technique of sending multiple requests without waiting for each response.

Because a server must send its responses to requests in the same order that the requests are received, it is not necessary to wait for each response before sending the next request. By sending multiple requests through the HTTP pipeline, instead of one at a time, an application appears faster to the user.

Periodic Transfers

Periodic transfers can be very useful, but it is easy to mismanage them to the point where they can waste energy and slow down an application.

The two most common uses of periodic transfers are analytics and keep-alive pings.

Resource allocation and the energy state for a device are the result of a negotiation between the network and an application. When a device needs to transmit data, it signals the network and the network allocates a dedicated channel; or if data is being transmitted to the device, the network alerts the device. The culmination of this process is a promotion of the device to full power so that it can transmit data.

Sending even one extra packet that could have been reduced means the wireless radio was turned on needlessly and this overhead was incurred. When an application requires periodic transfers, each transfer potentially incurs a long tail when closing the radio transmission, and that wastes energy and increases network congestion.

Over time, pinging the network repeatedly for analytics or advertising, for example, uses more power than delivering the actual content your users want. By reducing the number of these pings, you can greatly reduce the battery drain of your application.

The Best Practice Recommendation for using periodic transfers effectively in a wireless application is to experiment with different scenarios to determine which combination of periodic transfer timing, piggybacking and batching can improve your application the most.

- Determine the best timing for analytics and keep-alive so that your application uses as long a period as possible between transfers.

- Use piggybacking to combine analytics with other data and remove standalone pings. TCP piggybacking is the process of sending data along with an acknowledgement. Piggybacking lets both sides of the TCP conversation send data at the same time.
- Use batching to bundle as much data as possible into a single message. Batching or bundling data refers to using a single TCP segment to send all the data from a specified sequence number up to the maximum permitted by the segment window. When used appropriately, sending multiple messages down the networking stack in one batch can reduce latency.

AT&T's firewall keep-alive is 30 minutes, and the shortest TCP timeout found among major carriers is 4 minutes. This indicates that sending a ping every minute or so just to keep the connection alive is not necessary. A period of several minutes rather than one of a few seconds would be more appropriate for most applications.

Opening Connections

Many applications handle the opening of connections inefficiently. A typical application start-up consists of an initial TCP burst, followed by a series of bursts spread out over time. Unfortunately, this approach can dramatically slow down the application's response time and waste energy on the device, because every time an application causes a new packet burst, it increases latency across the board.

A better approach is to download as much content as quickly as possible when opening a connection.

On the wired web, opening connections is not as big of an issue because network bandwidth is broad enough to smooth out inefficiency. The wireless environment, with constrained radio networks and greater latency, however, requires efficiencies to be found wherever possible, including the opening of connections between a server and a mobile device.

The Best Practice Recommendation is to open network connections efficiently. You can do this by following some simple practices:

- Group TCP packets closely together when opening the connection.
- Download user requests for content as quickly as possible.
- Prefetch content. Prefetching refers to downloading any content that you anticipate your users will request next. For best practice recommendations about prefetching, see Deep Dive: Content Prefetching.

Closing Connections

Mobile applications often leave a connection open long after data has been transmitted and the connection is no longer required. This typically happens when a connection is not deliberately closed as part of the transmission and is terminated later when the connection times out. This is an inefficient way to close a TCP connection.

It is good practice to close a connection as soon as possible after data is transmitted, to prevent radio channels from being kept open needlessly.

Offloading to Wi-Fi

Wi-Fi connections are much more efficient from a battery and network perspective, because there is little latency for connection setup, and therefore no need for the radio state machine.

The Best Practice Recommendation is to offer your users the opportunity to connect to Wi-Fi when it is available.

Screen Rotations

Many applications are written so that the device pings the server and reloads the page every time the user rotates the screen. This is often done even when there is no change to the contents of the page. In such a situation, pages do not need to be reloaded.

The Best Practice Recommendation is to track the orientation in the application locally, and send information at scheduled intervals, bundled with another activity. This approach would save energy for each unnecessary transfer of data that is eliminated.

Duplicate Content

Each time a client requests content from a server, the server returns it as an Entity (the payload of an HTTP Response Message).

Since every content request results in a Full Response from the server, eliminating the Full Responses that result in duplicate content also eliminates the associated overhead for the Full Response.

The Best Practice Recommendation is to incorporate a caching strategy into your client application design. The HTTP 1.1 protocol supports Response Entity caching mechanisms, and you can put them to use by incorporating them into your caching strategy.

The cache is a process that runs locally, as a service. It behaves transparently, and operates as a middle-man standing in between the client and server processes. The cache serves locally-stored copies of Response Entities, so you can refer to it as a Response Entity Cache. There are several ways to implement caching. There are libraries available for this, and some operating systems have functions for it, but the most direct approach is to incorporate Response Entity Cache functionality into your client software code.

This may be done by implementing a class that wraps a collection of Response Entities. This class could encapsulate a searchable container of Response Entity objects and include methods for:

- Transparently intercepting Inbound Response Messages.
- Determining if a Response Entity is Cacheable.
- Adding Cacheable Response Entities to the collection.
- Transparently intercepting Outbound Request Messages.
- Checking to see if the associated Response Entity is a cached item.
- If not (a cache miss), relaying the Request Message to the Origin Server.
- If yes (a cache hit), determine if it is appropriate to serve the cached response.
- If it is, then serve the cached response.

Cache Control

By implementing a cache, and making full use of the caching mechanisms and Cache-Control directives that are available to your application, you can improve speed, energy usage, and user experience.

When a file is cached, it is available immediately for reuse, which makes an application appear faster. Enabling a cache, and making full use of Cache-Control directives, reduces the amount of data and data connections that are sent needlessly. These savings can help keep a user beneath their data cap, keep their battery from draining as quickly, and improve the responsiveness of wireless networks that have limited capacity.

Some applications do not implement a cache, and many do not make full use of the caching mechanisms and Cache-Control directives in HTTP/1.1. This leads to unnecessary downloads that waste energy and make an application appear slower.

While it’s true that downloading one 4KB image unnecessarily, may not waste much energy. That amount can grow exponentially as your application gains exposure and that image is downloaded thousands of times.

If your application requests a 4 KB image twice per session with just 5,000 users, you're sending 19 MB of "extra" data to your users. The radio power usage on these 5,000 additional downloads is equivalent to draining 35% of a sample smart phone battery.

The Best Practice Recommendation is to implement a cache in your application, and to make full use of the caching mechanisms and Cache-Control directives in HTTP/1.1.

A cache is a process that runs locally, as a service. It behaves transparently, and operates as a middle-man standing in between the client and server processes. A cache serves locally-stored copies of Response Entities, and can be referred to as a Response Entity Cache.

Cache Expiration

The Best Practice Recommendation is to ensure that caching is enabled on the server-side, and that a response entity cache is implemented in your application that makes the best use of the HTTP 1.1 expiration model.

Content Pre-fetching

There are two conflicting facts when it comes to delivering content for mobile applications:

- There is greater latency when joining wireless networks and wired networks together, than there is in wired networks alone. This often makes content delivery slower.
- Users expect wireless networks to behave just as quickly as wired ones.

To balance these two contradictory facts, developers need to devise a content management plan. Such a plan includes the following choice:

- Should content be delivered only as it is requested by the user?
- Should the expected behaviour of the user be anticipated and additional content be download before the user requests it?

Prefetching is the process of retrieving and caching content before it has been requested by the user, and when used intelligently, it can speed up the user's experience with your mobile app.

The Best Practice Recommendation is to use prefetching. In other words, you should select content to download in anticipation of what your user will want to see next.

Prefetching data can be overdone, and a balanced approach should be taken. For instance, prefetching should be stopped when the user clicks on a link, loads different content, or makes the app do any sort of networking activity.

It all comes down to understanding your content and understanding your users.

Accessing Peripherals

As today's phones are being loaded up with cool features, the inclination of developers is to use many of these features and peripherals as they can. This is not a problem if the app actually needs to utilize those peripherals, but many apps access peripherals that they never use.

When accessing device hardware, you should ensure that your software has failover parameters that are set so that they turn off the peripheral after period of inactivity. Turning off the Bluetooth, GPS and other features when they are not in use is an easy way to save battery life in your app.

Sometimes, there are more efficient alternatives to using peripherals. Using alternative location information rather than enabling GPS for example.

If your app uses Bluetooth, camera, GPS, or another device feature, code a standby mode. That way, when the peripherals aren't in use for a pre-determined period of time, the accessory turns off or goes into standby mode. As you'd expect, reducing the power consumed on the device preserves the battery life, and allows users to access your application for longer sessions.

Another technique for saving battery life is to turn GPS off when the device is not moving. You can use the device's accelerometer to verify that motion has started, and then re-engage the GPS.

Blocked Connectivity

Ensure that the app can handle a network connection being invalid or unusable. For example the data connection may not be properly set up, or the user is roaming, or connectivity is simply unavailable. The device may even be switched into offline or flight mode.

The user should be notified with an appropriate message if this occurs: ideally the application should indicate that the device may be in flight mode (or equivalent) stating that the application cannot run successfully.

Sending and receiving data

Ensure your application can connect via a valid data session setup and send/receive data via an HTTP network session. You need to make sure that the application data is properly sent / received over the network. Check it for each application screen or feature that uses data services.

Network delays and loss of connection
When the application uses network capabilities, it should be able to handle network delays and any loss of connection, giving a clear error message to the user indicating there was an error with the connection.

Messaging & calls

Sending or receiving SMS and / or MMS
For an application that sends and/or receives SMS or MMS messages as part of its function, ensure that it can send messages successfully and that:

a) a notification of a new message is given where enabled on the receiving handset
b) the message is in the correct format, and, where MMS is involved, it contains the correct payload.

Handling telephone call when application is in use
The user must be able to accept an incoming phone call while the application is running.

It should then be possible to resume from the same point in the application at the end of the call, or a logical re-starting point.

The application should not block the user from making emergency calls on a cellular network.

External Influences

Memory card insertion
For an application on a device that supports removing and replacing memory cards while applications are running, the application should handle this gracefully and continue to operate as designed.

User Interface

Navigation - don't hijack the native experience
Only hide the status bar where the application's user experience is better without it.

Use native icons consistently. The application should not replace a system icon with a completely different icon if it triggers the standard UI behaviour.

Don't override the platform menu button. Instead, put menu options behind the menu button.

Respect user expectations for navigation. Where the platform provides a Back button, it should always navigate back through previously seen screens. Support the standard Back button navigation and don't seek to replace this with on-screen navigation.

Always support trackball navigation if the platform supports it. Understand your navigation flow when the platform permits the entry point to be a notification or widget.

Navigating between application elements should be easy and intuitive.

The application must support the standard system navigation, and must not make use of any custom on-screen prompts or buttons that attempt to replace system navigation functions.

On Android all dialogs must be dismissible using the Back button.

If the platform supports a Home button, pressing this at any point must navigate to the expected default or home screen on the device.

Discrimination

Don’t make assumptions about screen size, resolution, orientation or input. Never hard-code string values in code. Use Relative Layouts and device independent pixels. Optimize assets for different screen resolutions, and use reflection to determine what APIs are available.

Read time and readability

There should be a comfortable amount of time for content reading. Each screen should be visible for the time necessary to comfortably read all its information. Everything in the application should be in a font size and type that is readable by the user.

Touch screen use

For applications used in a touch screen device without stylus, on-screen elements should be of sufficient size and responsiveness to provide a good user experience.

Screen repainting

The application screens should be correctly repainted, including cases when edit boxes and dialog boxes are dismissed. Also, there should be no blinking of moving objects and background. If the application objects overlap they must still render correctly.

Consistency

The application UI should be consistent and understandable throughout, e.g. displaying a common series of actions, action sequences, terms, layouts, soft button definitions and sounds that are clear and understandable.

The application should not redefine the expected function of a system icon (such as the Back button on those platforms that support it) or replace a system icon with a completely different icon if it triggers the standard UI behaviour

If the application provides a customised version of a standard system icon, it must strongly resemble the system icon, and must trigger the standard system behaviour.

The application must not redefine or misuse platform UI patterns, such that icons or behaviours could be misleading or confusing to users.

UI & Graphics

The application should support both landscape and portrait orientations if possible.

The application should expose largely the same features and actions in both orientations, and preserve functional parity. Minor changes in content or views, or UI changes to provide greater functionality in a landscape widescreen tablet display, are acceptable.

The application should use the whole screen in both orientations and not letterbox to account for orientation changes (but minor letterboxing to compensate for small variations in screen geometry is acceptable).

The application must handle rapid transitions between display orientations without rendering problems.

The application should cope with “flooding” with touch inputs, e.g. by the user rapidly and repeatedly tapping the screen.

Visual Quality

The application should display graphics, text and other UI elements without noticeable distortion, blurring or pixilation.

The application should provide high-quality graphics for all targeted screen sizes and form factors, including for larger-screen devices such as tablets.

There should be no visible aliasing at the edges of menus, buttons and other UI elements.

The application should display text and text blocks in an acceptable manner.

Composition should be acceptable in all supported form factors, including for larger-screen devices such as tablets.

There should be sufficient spacing between text and surrounding elements.

Application speed

The application should work in the device it was targeted for, and should be usable on the device: the speed of the application should be acceptable to the purpose of the application and must not alter the user experience by being uncontrollable.

The speed of the application should be good enough for the application usage (i.e. the frame rate or response to user input should remain adequate, and not compromise the application usage, or prevent the user from progressing normally). For games, to ensure a smooth looking experience for your user, a minimum generally accepted frame rate is 15 frames per second. Also, make sure not to base animations on the frame rate. In other words, you may not want to paint and adjust drawing locations every time the drawing loop is cycled through. You should introduce timers into your code to ensure that animation is consistent across devices. On some devices, you may need to adjust drawing locations less frequently than on others.

Notifications and Error messages

Notifications should follow the design guidelines for the platform.

Multiple notifications should be stacked into a single notification object where the platform supports this.

Notifications should only be persistent if related to on-going events (such as music playback or a phone call).

Notifications must not contain advertising or content unrelated to the core function of the application, unless the user has made a conscious choice to accept such data via an opt-in option offered by the application.

The application should only use notifications to

- Indicate a change in context relating to the user personally (such as an incoming message)
- Expose information or controls relating to an on-going event (such as music playback or a phone call).

Any error messages in the application should be clearly understandable. Error messages should clearly explain to a user the nature of the problem, and indicate what action needs to be taken (where appropriate).

Function progress

Any function selected in the Application should give evidence of activity within five seconds. There should be some visual indication that the function is being performed. The visual indication can be anything that the user would understand as a response, e.g.

- prompting for user input;
- displaying splash screens or progress bars;
- displaying text such as "Please wait...", etc.

Actions while performing calculations or image rendering processes

The user display and application activity should remain consistent and stable when calculations or image-rendering processes are being performed.

Multiple display format handling

Where the device and application can display in multiple formats (e.g. portrait / landscape, internal / external display), the elements of the application should be correctly formatted in all display environments, with the application displaying correctly without obvious errors in all formats.

If the application is supposed to be used ONLY in landscape mode, then the application should be forced so that it cannot be rotated to portrait mode. The same applies to applications intended to be run in ONLY portrait mode.

Differing screen sizes and device formats

Where the application is designed to work on multiple devices it must be able to display correctly on differing screen sizes. In particular, tablet devices now offer a significantly larger range of screen sizes and also the likelihood of the user predominantly using the device in landscape mode, so developers should make use of the following detailed recommendations.

- Optimise layouts and other UI components for each targeted screen configuration. Take full advantage of the additional screen area available on tablets.

- Initially adjust layout, fonts and spacing to help your existing layouts work on smaller tablets.

- Redesign the UI with a multi-pane layout, improved navigation and additional content for larger tablet displays.
- Provide custom layouts for larger screens, detect the screen size where possible and load them based on detecting the screen size or shortest dimension.
- For tablet devices with larger screens used in landscape orientation, adjust the positioning of UI controls to be near to the sides of the screen within easy reach for the user (for example; consider thumb radius of movement for controls when a tablet is held in both hands).
- Padding of UI elements needs to be larger on tablet screens for optimum appearance.
- Make sure content is adequately padded to keep it away from the edges of the display on tablet size displays.
- Avoid layouts that look stretched on widescreen format landscape displays:
  - Work to an absolute maximum of 100 characters per line, but bear in mind that best results will come from targeting 50 – 75 characters per line.
  - Lists and menus should not use the full screen width.
  - If padding of onscreen elements does not prevent a stretched appearance, switch to a multi-pane UI for large landscape screens.
- When adopting a multi-pane UI, look at information you might have displayed in sequential screens on a smaller display, and see whether that information would be suited to displaying in separate panes of a new UI on a large landscape display.
- Use alternative bitmap assets (e.g. icons) to give the best results on different screen densities. Icons that will appear in OS areas outside the app itself, such as launchers and notification areas, should:
  - Meet any recommended design guidelines for the platform.
  - Be of a size matching other icons deployed in these areas on the platform.
  - Appear similar in size and design whether the icon is viewed on a large or small screen.
- Adjust font sizes and touch targets in your UI for large screens.
  - Adjust font sizes through styleable attributes or dimension resources.
- Text should not be excessively large or small on any of the devices you target.
- Labels should always be appropriately sized for their corresponding UI elements.
- There should be no improper line breaks in labels, titles and other elements.
- To meet the accessibility needs of certain users it may be appropriate to use larger touch targets than recommended standards.

- Where there is a design requirement for the use of a small icon on a touch control, look at techniques for expanding the touchable area associated with the icon.
  - When a platform allows the touchable area of a smaller icon to be expanded, use this where appropriate; otherwise consider centring a smaller icon within a larger transparent button.
- Where the platform supports the use of home screen widgets, ensure the widget's default height and width, and minimum / maximum resize height and width are appropriately set for all the display sizes you support.
- Design your app to offer at least the same set of features to tablet users as phone users.
- Be aware that not all phone features will be available on tablet devices, or their use may not be appropriate or compelling on a tablet.
  - Telephony functions may not be available or easily usable.
  - Vibration alert may not be available.
  - GPS location may not be available, or may not be practical to use in the intended way on a tablet (e.g. in an app for the live recording of running tracks).
- If you have to design for the absence of a function on a tablet device, make sure that your app does not offer functionality that cannot be used on the device.
  - Consider whether there is any opportunity for "graceful degradation" to an alternative functionality (e.g. when checking for connectivity, accept WiFi as a substitute for unavailable 3G / 4G connectivity where possible).
  - Consider whether a user-input location (like a street address or a postal / zip code) would allow the app to be usable on a device without GPS functionality.
- Where the platform supports declaration of supported screen sizes, make sure that you have declared support for al relevant sizes.
- Follow the platform recommendations for declaring support for multiple screen sizes when publishing your app through distribution channels.
- Where possible, release a single executable that adapts to all the screen sizes you support, because this:
  - Ensures that users can easily find your app and that they can continue to use your app if they acquire a different format device.
  - Maintains consistency of stats and ratings and avoids dilution of ratings in any distribution channel across multiple versions.
- When testing an app for a platform that has multiple tablet sizes (e.g. 7-inch, 10-inch), make sure to test using devices or emulator simulations for all the possible size ranges that your app targets.

Multiple format input handling

Where the device and application can accept input in multiple formats (e.g. external touch screen / external keypad / internal touch screen / internal keypad / QWERTY layout / 12-key layout and others), it should work correctly with all supported input methods.

Accelerometer / motion sensor responses

Where both the device and the application use accelerometer / motion sensor support, the response of the application to movement or change of alignment of the device should not impair use of the application, nor be likely to confuse the user. The application should change between portrait and landscape modes without confusing errors being displayed to user.

Spelling errors

The Application should be free of spelling or language errors unless they are part of a deliberate design concept.

Technical text errors

The text in the application should be clear and readable. The application should be free of technical text display issues such as: Text cut off / Text overlapping, and all text in each target language should be displayed without corruption, distortion or other display problems.

Problems to avoid are:
-Menu item text labels incorrectly aligned with cursor
-Button text label over-running the button area or truncated such that its meaning is not clear
-Text over-running or being truncated in other bounded text display areas (e.g. speech bubbles, user interface elements etc.)
-Text not wrapping at the edge of the screen resulting in words being cut off
-Multiple pieces of text overlapping each other, or text overlapping user interface elements
-Text being cut horizontally.

Language

Correct operation

Ensure that the application works correctly with all appropriate languages and allows the user to select languages if appropriate, with the correct rendering.

Supported formats

Verify that date, time, time zone, week start, numeric separators and currency, are formatted appropriately for the implemented language's target country and supported throughout the application.

International characters and units of measurement

Ensure that the application accepts and displays all appropriate international characters and units of measurements correctly, according to the local market needs.

Performance

Battery Life

Consider battery management, and allow the power-saving features of the device to be used, including sleep functions for the screen and the device itself.

The management of connectivity can have a huge impact on battery life. See AT&T ARO and GSMA guidelines on network and radio efficiency to assist in this area.

Responsiveness

Always update the user on progress. Render the main view and fill in data as it arrives. Always respond to the user input within 5 seconds. Users perceive a lag longer than 110-200ms adversely.

AT&T and GSMA guidelines provide advice on caching that can significantly improve responsiveness.

In Android with Strict Mode enabled, no red flashes (performance warnings) should be visible when exercising the app, including during game play, animations and UI transitions, or any other part of the app.

Suspend / resume

Where an OS environment supports 'suspend / resume', ensure that the application suspends and resumes at a point that does not impair the user experience.

Multi-tasking and effect on other functions

In a multi-tasking environment, remember to release used resources or functionality for other applications to use when not in use by that application.

An application should correctly handle situations where - following user input, or some external event (e.g. a phone call) - it is switched to the background by the terminal. Upon returning to the foreground the application should resume its execution correctly.

While in the background the application should not intrude in any way on the operation of other applications or handset functions, unless its explicit design is to do so and that is clearly explained in an accessible Help file.

When pauseApp()/hideNotify() is called by the system, the MIDlet should do the following:

1. Pause the application.

2. Save application state.

3. Release any resources the app won't need while paused (like sound resources).

Upon they system calling startApp() should reallocate the resources it needs and renew the app from its saved state.

For apps which are meant to run in the background, it's not necessary to do all of the above, but apps should at least stop the drawing loop as drawing while in the background is a waste of system resources.

User State / Application State

The application should not leave any services running when it is in the background (unless needed for a core capability of the app). Generally speaking, the application should not leave services running to maintain a network connection, or to maintain a Bluetooth connection, or to keep GPS location powered-on.

The application should correctly preserve and restore the user or app state; particularly, it should preserve the user or app state when leaving the foreground, and prevent accidental data loss due to back-navigation or other state changes.

When returning to the foreground, the application must restore the preserved state, and any significant stateful transaction that was pending (such as changes to editable fields, game progress, menus, videos, and other sections of the application or game.

When the application is resumed from the background or from sleep, it should return the user to the exact state in which it was last used, unless that state has been invalidated by the passage of time (e.g. the expiry of a limit related to actual calendar date / time rather than elapsed time in the application).

When the application is re-launched from an app launcher or home screen, it should restore its state as closely as possible to the previous state (e.g. if the application is already running and the user navigates away to another application or system function, then re-launches the app, it should aim to continue the existing session rather than start a completely new instance, if this is possible).

If the platform provides a dedicated Back key or function (and this is selected), the application should then give the user the option of saving any app- or user-state that would be otherwise lost on back navigation.

Resource sharing – database

Where there are multiple applications that make use of a common database (for example, calendar synchronisation and calendar viewing), the database should be properly shared between those applications.

Other programming dos and don’ts

Don’t update widgets too frequently, or update your location unnecessarily or use Services to try to override users or the system as these adversely impact data usage and battery life.

But do share data to minimize duplication and let users manage how often they update, and whether or not they want to allow updates to happen at all.

Media

Audio Behaviour

For applications with sound settings, there should be a Mute or Sound On / Off setting, unless the Application does not have Application mute facility by design or it respects the settings of the handset volume buttons.

Audio must not play when the screen is off, or behind the lock screen, or on the home screen, or over another application, unless it is a core feature (e.g. the application is a music player).

Audio should resume when the app returns to the foreground, or should clearly indicate to the user that playback is in a paused state.

Media Performance

Music and video playback must be smooth, without stutter, crackle or other artefacts, during normal application usage and load.

Settings statuses understandable

Where the application has settings options, ensure that the settings statuses are easily understandable at any stage in the application’s journey.

Saving settings

Where the application has settings options, ensure that the application saves all settings on exit and that restarting the application will restore the saved settings.

Specific functions

For applications with sound, ensure application sounds have specific functions and should not be over utilised - e.g. a game completing with a minute of random noise should be avoided.

For media transmission guidelines, see GSMA Smarter apps for smarter phones section 3.4.

Menu

Help and About

An application with user interface capable of displaying information to the user should contain standard Menu items Help & About (or equivalent information in a format easily found and understood by the user) to explain to the user how the Application works.

If it is clear that the application’s purpose requires network coverage to operate, then it would be sufficient for the Help to be provided through a browser connection rather than being contained in the application. In the opposite case, where most functions of the application can be used while the device is offline, then the application should have Help that can be accessed without needing a data connection.

Menu items like Help and About should be presented on the main menu or other easily-found screen of the application.

About functions should contain the application name, application version number and author information.

Help should include the aim of the application, usage of the keys (e.g. for games) and other instructions. If the text of the help is too long, it should be divided into smaller sections and/or organized differently.

Help must be accurate and consistent with the application functionality and the handset specifics.

Valid actions

All application items that can be selected and/or changed by user should do what the application is intended to do.

Functionality

Functionality sanity check

All specific application functionality such as algorithms, calculations, measurements, scoring, etc. should be implemented correctly.

Application hidden features

The application should not introduce any hidden features: its functionality set should be consistent with the help and it should not harm the data on the device. However, the following hidden functions are OK: Cheat codes and the unlocking of an application to upgrade from a demo version to a full version.

Keys

Scrolling in menus

For an application with user interaction, when the keypad or other navigation device is used to scroll vertically and (if applicable) horizontally in the Main menu item list, there should be no adverse effect on the application. In addition, an application should be able to lock itself in a vertical or horizontal view if seen as important from application-use point of view.

Selection key

For an application with user interaction, pressing the primary selection key or device equivalent in the main menu item list should select the menu item with no unwanted effects on the application.

Text field scrolling

For an application with user interaction, the scrolling functions of the keypad or other navigation device in a text dialog (for example: About and Help) should scroll vertically and (if applicable) horizontally in the dialog.

Pause

For an application where time-sensitive immediate user intervention is needed, the application should support a pause feature in the areas of the application where this is applicable (for example in-game).

The user should - where necessary - be able to easily pause and resume the application.

All time-specific features of the application should be disabled at the time of the pause.

There should be a clear indication that the application is in a paused state.

There should be a clear indication of how the user can return from the paused state.

(You should use the potentially-available APIs for pause and continue methods if possible.)

Simultaneous key presses

For an application with user interaction, ensure that it copes with simultaneous key presses by not putting the application into an unusable or incomprehensible state by simultaneous key presses. Any error messages generated should be meaningful.

Multi key presses

For an application that supports multi key press actions (on a device that also supports this), they should perform as predicted and should not leave the application in an unusable state.

Device Specific Tests

Device opening and closing

For an application on a device with open / close functionality, ensure that it handles opening and closing of the device correctly while launching and returns to the same state before the interruption.

Stability

Application stability

The application should not crash, unexpectedly close, freeze or otherwise behave abnormally at any time while running on any targeted device.

Application behaviour after forced close by system

The application should preserve sufficient state information to cope with forcible close by the system. It should not lose any information that it implies would be preserved, nor become difficult to use subsequently, as a result of a forcible closure by the system.

Data Handling

Save game state

For an application where the user may exit part completed game or where a player high score value is identified, ensure that it can save its game state/high score table information into persistent memory.

Data deletion

Where an application has a function to delete data, it should indicate whether data will be permanently deleted or offer easy reversal of the deletion.
The user should always be required to confirm deletion of data, or have an option to undo deletion, to reduce risk of accidental loss of information through user error.

Security

Encryption

All sensitive information (personal data, credit card & banking information etc.) must be encrypted during transmission over any network or communication link.

Passwords

If an application uses passwords or other sensitive data, the passwords or other sensitive data should not be stored in the device and not echoed when entered into the application. Sensitive data should always protected by password. If possible, all stored password should be encrypted With passwords, the desired approach is that the application shows which character the user selected and then changes that to an asterisk (*).

If the user is explicitly asked for permission, a password can be stored to the device memory.

It's important to minimise the risk of access to sensitive information should the device be lost, by ensuring that no authentication data can be re-used by simply re-opening the application.

Once sensitive data has been entered, it should not be displayed in plain text anywhere in the application. However it is generally acceptable to have no more than 25% of a sensitive value displayed in plain text (e.g. 4 of the 16 digits of a card number) where this assists the user to distinguish between multiple cards or accounts.

Misleading security

An application:

- Must not override system or virtual machine generated security prompts or notifications or deceive the user by displaying misleading information just before a security prompt is shown to the user of the application.

- Must not simulate security warnings to mislead the user of the application.

- Must not simulate key-press events to mislead the user of the application.

- Must run in the sandbox environment and must not exploit any malicious means of exiting the sandbox environment.

Privacy, Content & Policies

Privacy Legislation

Content must comply with local privacy legislation including privacy laws regulating the processing of personal data in all markets in which the content is published.

User's rights

If the application collects personal data, and as further defined in applicable laws, the developer has an obligation to users to:

- Disclose what personal data is held about them.

- Update or delete incomplete, incorrect, unnecessary or outdated personal data.

- Enable them to unsubscribe from marketing messages and to request that you stop using their personal data for direct marketing or market research purposes.

The application must also provide users with contact information or online tools to exercise their rights and must contain an option to delete personal information (deletion during un-installation at the minimum).

Privacy Notice

Applications that collect personal data should provide a clear, understandable privacy notice so that users can make informed decisions about whether or not to use the application or certain features of the application.

The requirements within 'privacy' and 'user's rights' plus some others have been included in this section where we provide an outline of what needs to be included within any Privacy Notice association with an application.

Applications must make the Privacy Notice available to users prior to the initial collection of personal data through the application, and viewable anytime thereafter within or through the application (for example, through the application's Help menu, a link to a website hosted by you, or a similar effective method).

The notice may be in the form of a privacy policy, either embedded in the application or linked to a website, and should include the following information:

o What personal data is collected, how it is used and shared, and how long it is retained.

o When personal data is shared with third parties:

- detail the type of third party with whom this information is shared (for example, service providers, other users, or governmental agencies), for what purpose, and how the user can restrict the disclosure of personal data to such third parties.

o When tracking technologies are used:

- detail the type of technology used, for example, cookies or analytics software (irrespective of whether the technology is used by you or named third parties who may be collecting information from the application).

- detail what information the technologies track, with whom the information is shared, the purpose for using the technologies, and how they can be disabled.

o What means users have to manage their privacy preferences and instructions on how to use these means.

o That users have the following rights with respect to their personal data:

- the right to know what personal data you have about them.

- the right to request the deletion or modification of unnecessary or outdated personal data

- the right to unsubscribe from direct marketing messages or use of personal data for marketing or market research purposes.

o Instructions on how to exercise such rights, for example, through online tools or otherwise through contact details provided by you.

o What security measures the application uses to protect personal data against unauthorized access, use, modification or loss.

o How the user can contact you, including company name, email address or a link to an online form as well as a physical address and a phone number, if available.

Location Data

Applications using location data should:

o Obtain voluntary, informed, express, and revocable permission (also known as active consent) to use location data from the user. Active consent must be obtained separately from approval of service terms or Privacy Notice / Policy.

o Provide the ability to opt out, use defined APIs, not override or circumvent a user's choice, and not restrict access to user information on the device.

o Periodically remind users or provide a visual indicator when location data (GPS, IP address, cell tower, Wi-Fi based location data) or user information is being sent to any other service, service provider, user or other third party continuously or periodically on an on-going basis, if this is done without the user's separate active consent.

Safeguards

Applications must process personal data only for justified purposes that are relevant to the features and functionalities of the application.

Applications require user registration only when it is needed to use the application, for example, to log into an existing user account.

Passwords or other sensitive data, such as credit card details must not be stored in the device and the passwords may not be echoed when entered into the application.

Information stored by the application on the device, or on servers, must be deleted when storage is no longer necessary for the purposes for which the information was originally collected or when storage is not required.

Users must be able to delete information stored by the application on the device (for example in caches, search histories, keyboard logs and other such storage areas) and any user-generated content stored on servers.

Information security and accountability

You must have appropriate information security measures to protect personal data against unauthorized access, use, modification or loss when stored on the device or transmitted to and stored in other repositories.

All information lifecycle schedules for the data you control must be defined.

There must be appropriate processes in place for information accountability and for assigning responsibility for information security.

Content & Policies

The application should adhere to any platform or distribution channel policies regarding inappropriate content and infringement of brands or intellectual property belonging to others.

Where the distribution channel requires a maturity level to be assigned to an application, this should be appropriate and in accordance with any platform or distribution channel guidelines.

Applications that request permission to use the device location must not have a maturity rating that makes them available to all ages without restriction (where maturity ratings are implemented).