SMASHING MAGAZINE

A Guide To Designing Touch Keyboards (With Cheat Sheet)

By Christian Holst

Published on August 13th, 2013 in Mobile, Usability, User Experience with 19 Comments

Touch devices have rightfully been praised for generally being much more intuitive than the decades-old computer mouse and keyboard. Users interact directly with touch interfaces, which narrows the gap between human act and software response. Yet typing on mobile devices — in particular on smartphones — is quite the horror story. It's **slow, painful and error-prone.**

The obvious culprits are keyboard character size and proximity of the keys, but there are many other important aspects to consider, including:

- using auto-correct dictionaries appropriately,
- auto-capitalizing relevantly,
- hinting at the input type,
- honoring the tab sequence,
- invoking custom keyboards consistently.

During a recent large-scale 1:1 mobile usability study of 18 of the largest mobile commerce websites, we observed how certain features and limitations of modern touch keyboards can collide with the user's expectations of how to fill out a form. When this happens, users quickly grow frustrated, as one form-field validation error pops up after another or, worse yet, the user gets stuck and ultimately abandons the website.

When faced with a suboptimal touch keyboard implementation, users lose confidence in the website, and some even doubt their own ability to fill out a form on a smartphone. Clearly, a good mobile experience requires good form usability, and implementing touch keyboards is a key part of that.

In this article, we will look a bit deeper into the usability issues surrounding touch keyboards, including **five design guidelines** that will alleviate some of these pains. The guidelines are an excerpt from the 147 guidelines in the M-Commerce Usability Report. We previously looked into 10 guidelines for mobile e-commerce here on Smashing Magazine; these 5 guidelines on touch keyboards are more generic and apply to any mobile website on which the user interacts with a touch keyboard.

Furthermore, we've also benchmarked the mobile websites of the top 50 online retailers against these five guidelines and found that an astounding 98% get one or more of these wrong, and 70% of the top mobile websites get at least two wrong (as of 31 July 2013). While some of the guidelines might seem obvious at first, clearly we all need to pay better attention when so many multi-million dollar e-commerce stores get them wrong.

1. Disable Auto-Correction When The Dictionary Is Weak (92% Get It Wrong)

Issue: Poor auto-correction is frustrating when users actually notice it, and can be detrimental when they do not.

Auto-correction often works very poorly for abbreviations, street names, email addresses and other words that are not in the dictionary. This caused significant issues throughout testing and resulted in a great deal of erroneous data being submitted as test subjects completed their purchases.

As this subject typed in the street name "westheimer" on the website for Toys'R'Us, the phone incorrectly auto-corrected to "weathermen" (left). However, the subject did not notice this, submitted the form and received a validation error (right). (Large view)

One major problem with auto-correction is that **users often fail to notice the correction** (because they are focused on what they are typing instead of what they have typed). This is fine if the correction is correct, but it can be detrimental if it is wrong. For example, in multiple instances during testing, valid addresses were auto-corrected to invalid ones and submitted because the subject didn't notice the auto-correction.

On websites without address validators, this resulted in orders being shipped to wrong addresses, unless the subject was particularly attentive on the order-review page (after all, auto-corrected data often looks very similar to the intended input, making users less likely to notice the error). Of course, auto-correction fails miserably in the address field not just in edge cases (such as with "weatherman"), but with common (and standardized) abbreviations, such as "Rd" being auto-corrected to "Ed."

That being said, **auto-correction did prove helpful in other scenarios** when it corrected invalid data to valid data. Disabling auto-correction on all fields (such as comment fields), therefore, is not recommended. Instead, use discretion, and disable it on fields for which the dictionaries are weak. This typically includes proper names of various sorts (streets, cities, persons) and other identifiers (email addresses, coupon codes, etc.).

While seemingly simple, in practice this is by far the most neglected part of touch keyboard usability; almost every single top mobile commerce website gets this one wrong. The benchmark reveals that 92% of them haven't disabled auto-correction on the address field. Given the severity of the problems caused by auto-correction on address and email fields, it's astonishing how few actually disable auto-correction here.

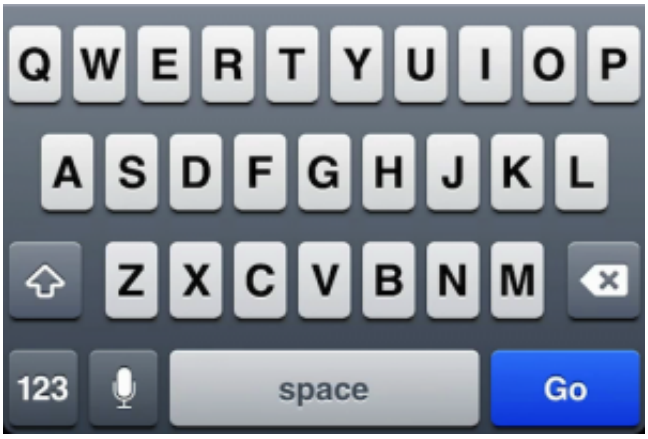
You can disable auto-correction by adding the `autocorrect` attribute to the input tag and setting it to `off`, like so:

```
<input type="text" autocorrect="off" />
```

2. Show The Appropriate Keyboard Layout (60% Get It Wrong)

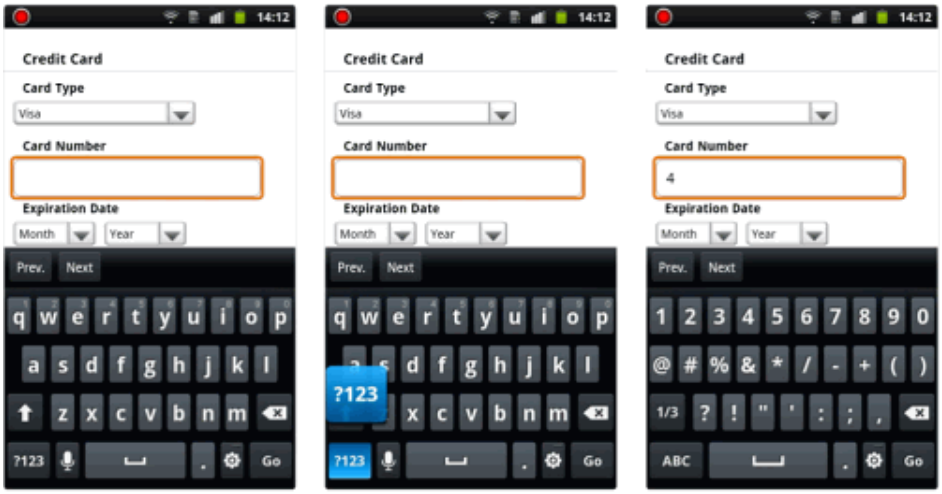
Issue: Inappropriate keyboard layouts slow down typing, and users generally mistype long number sequences on standard keyboards because of the small hit area and the close proximity of the numeric keys.

One of the main limitations of touch keyboards on smartphones is their size. The letters themselves are minuscule. In fact, a character on an iPhone 4 in portrait mode measures 4 × 5.9 millimeters.



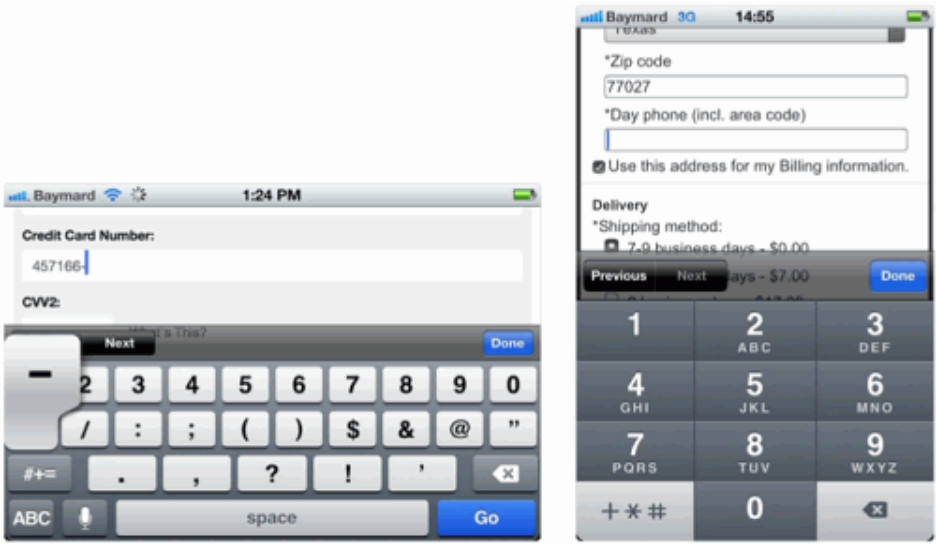
Compare this to Apple's own design guidelines, which recommend that all clickable interface elements be of least 6.85 × 6.85 millimeters because anything below that would yield very poor click accuracy. (Microsoft and Nokia also recommend a minimum hit area of approximately 7 × 7 millimeters). Predictably, this results in misspellings.

But by changing an attribute or two in the code of your input fields, you can instruct the user's phone to automatically show a keyboard optimized for the requested input. For example, you can invoke a numeric keyboard for a credit-card field, a phone keyboard for a telephone field, and an email keyboard for an email address. This saves the user from having to switch from the traditional keyboard layout, and, in the case of numeric inputs, **minimizes typos** because these dedicated keyboards have much larger keys, thus reducing the chance of accidental taps.



The credit-card input on Best Buy invokes the standard keyboard layout, so the user has to first switch to the numbers and special characters view (middle) and then type out the 16 digits without a single typo. This was a difficult task for many subjects, who looked to and from their card and phone while trying to hit the miniature buttons stacked against one another. (Large view)

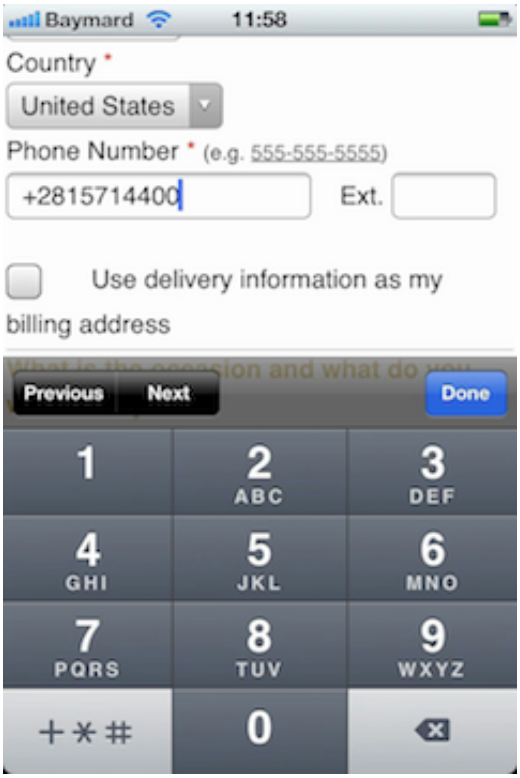
Throughout testing, multiple subjects noticed these dedicated keyboards and commented on them approvingly. In fact, on iOS, the hit area of a key is 471% larger on the numeric keyboard than on the traditional keyboard (209 × 108 pixels versus 52 × 76 pixels). More importantly, we recorded significantly fewer typos in numeric inputs when a numeric keyboard layout was displayed. This led to **fewer validation errors**, which, in turn, resulted in a better and more seamless shopping experience on those websites. This was especially true of long sequences, such as phone and credit-card numbers.



On the left, a subject accidentally hits the dash button instead of the "1" button due to the small size and proximity of buttons on the standard keyboard layout. A number-optimized keyboard layout would have been more appropriate. On the right, when the user fills out the "Day phone" field on GAP's website, a special phone-optimized keyboard appears, showing buttons that are 471% larger than those on the traditional keyboard. (Large view)

Another benefit of dedicated keyboard layouts is that they indicate the required input, which is helpful if the label is out of sight or the user is unsure of what to enter. However, note that numeric keyboard layouts can be limiting because they do not allow the user to enter alphabetic characters and allow only a few, if any,

special characters or separators. Invoking these keyboards on fields where they are the best match, therefore, is important, which includes **phone numbers, ZIP codes, credit-card numbers and credit-card security codes**. Similarly, make sure your formatting examples are actually possible to replicate using the invoked keyboard.



Typing a phone number according to the example format given here ("555-555-5555") is not possible on iOS because the keyboard layout does not include the dash character. (Interestingly, entering it on Android is possible, which goes to show why testing on multiple platforms helps to ensure that you don't require formatting that is only possible on some.) (Large view)

Given these substantial usability benefits, one would think that these dedicated keyboards are widely used. Yet, 60% of the top mobile commerce websites do not invoke them for one or more of the layouts for email addresses (email keyboard), phone numbers (phone keyboard) or credit-card numbers (numeric keyboard).

Technically, there are a few ways to invoke the numeric keyboard layouts, and also slight distinctions between them (i.e. phone versus number), with slightly different behaviors across platforms (iOS, Android, etc.). In general, two HTML attributes will invoke a numeric keyboard layout, namely the `type` and `pattern` attributes.

The `type` attribute carries semantic meaning and should be used only when an appropriate type is available for your input, which is the case for phone numbers and email addresses. For numeric inputs, however, providing a `pattern` attribute instead is recommended. (Note that you might want to add a `novalidate` attribute if you only want the browser to invoke the keyboard and not enforce this format.)

For any phone fields, use this:

```
<input type="tel" />
```

For any other fields where you want to invoke a numeric keyboard, use this:

```
<input type="text" pattern="d*" novalidate />
```

For any email fields, use this:

```
<input type="email" />
```

As mentioned, there are some distinctions between the types of numeric keyboard layouts, as well as some **differences between mobile platforms**. For example, on iOS, the code provided above to invoke the phone layout would produce a keyboard that allows the user to enter digits and a small set of phone-related special characters and separators, whereas the code for invoking the numeric keyboard would allow the user to enter only digits. Meanwhile, on Android, the phone keyboard layout is also invoked but with vastly more special characters, allowing for richer formatting of the phone number.

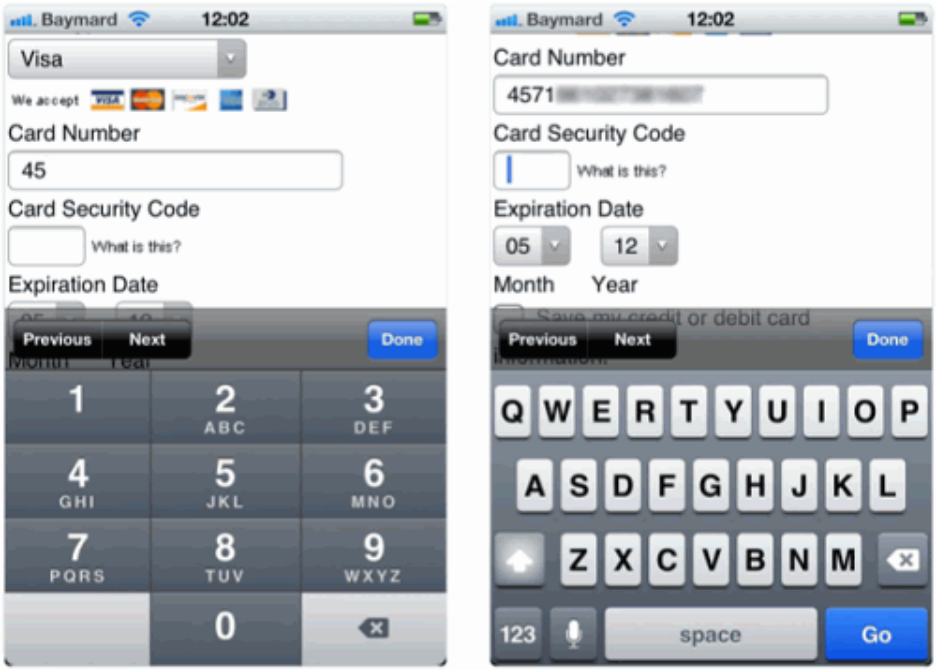
Yet, the numeric keyboard invoked by the `pattern` attribute isn't support by Android at all; instead, it simply invokes the regular alphanumeric keyboard. While you could use `type="number"` to invoke a numeric keyboard on both iOS and Android, setting the type to `number` carries semantic meaning that in many cases wouldn't be appropriate (for example, a credit-card number is a numeric sequence, not a number).

Therefore, we recommend the more defensive strategy of using `pattern="d*"`, which produces an enhanced experience on iOS while having no implications on other platforms that do not yet support this behavior. (Of course, if the field does represent a number, such as a price or quantity, then `type="number"` should obviously be used.)

3. Invoke Keyboard Layouts Consistently (54% Get It Wrong)

Issue: When one field invokes a dedicated keyboard layout but other similar fields do not, users are confused and begin to question the type of input requested by the field without the dedicated keyboard.

Invoking the appropriate keyboard layout for a particular input field is great (see the previous recommendation), but be sure to do it consistently throughout your website, or else you could greatly confuse the user. In other words, if a ZIP code field invokes a numeric keyboard, then similar input fields should have the same behavior.



(Large view)

While this might sound obvious, **many websites fail to invoke dedicated keyboard layouts consistently**. For instance, the flower store FTD (pictured above) invokes a numeric keyboard for the credit-card number but not for the very next security-code field, even though both values are always numeric.

Of the top 50 grossing online retailers, 54% get this wrong on their mobile websites, where one or more of the telephone, credit-card or CVV fields don't invoke a numeric touch keyboard. These 54% break down as follows (in absolute numbers): 24% invoke the numeric inputs for none of these three numeric inputs (which, although consistent, is consistently bad), and the remaining 30% (including FTD) are inconsistent, with the numeric keyboard layout being invoked on only some of the fields.

Even more surprising is just how confused some of the test subjects were by this during the usability tests. They began questioning their initial interpretation of individual fields, thinking that perhaps something else was required. For example, upon seeing the standard keyboard layout for the "Card security code" (pictured on FTD's website above), the subjects began wondering whether this was the three-digit code on the back of their credit card or one of the many other strings printed on the card.

4. Honor The Behavior Of The "Next" and "Previous" Buttons (4% Get It Wrong)

Issue: Users are either vexed or confused when "Next" and "Previous" buttons take them to fields that are illogically sequenced.

During testing, the subjects struggled with websites that failed to honor the behavior of the "Next" and "Previous" buttons. The expected behavior is straightforward: When the user clicks the "Next" button, they expect to be taken to the next logical field in the form, without any changes and without the form being

submitted. The same goes for the “Previous” button, just in the opposite direction, of course.

This goes beyond just having the right tab sequence (although that is a good start). Things often go awry when dealing with dynamic fields that depend on the user’s prior selection. In these instances, we’ve seen the data of users get deleted or the tab sequence violated. One must be particularly careful with the interfaces of custom forms, too. For example, in the Disney Store, a custom-designed state selector isn’t part of the tabbing sequence (because it technically isn’t an input element), and so users are sent right past the state field.

After filling in her ZIP code here (left), the subject hit the “Next” button, which correctly took her to the “Location Type” drop-down menu (right). But, as shown, the website cleared the subject’s previously inputted data. Obviously, data should persist when “Next” and “Previous” buttons are used. (Large view)

These buttons essentially function as the mobile version of keyboard tabbing; therefore, they should adopt the same sequential principles as desktop tabbing. They should provide a fast way to get from one field to the next without having to use a pointer (whether a mouse or finger). This is particularly important on mobile because screen space is so limited when the keyboard is open that the next field might be partially obstructed, making the “Next” button even more convenient to use. So, while the “Next” and “Previous” buttons might not be used by all users, the consequence of dishonoring the behavior of these buttons is significant.

Luckily, most websites get this one right. As long as the code is clean, mobile browsers will, by default, set the tabbing sequence according to the order of appearance of the fields. Of the top mobile websites, only 4% get this wrong.

5. Disable Auto-Capitalization Where Appropriate (38% Get It Wrong)

Issue: Nearly all subjects believed their email address had to be in lowercase, so auto-capitalizing this data adds needless friction to the process.

The default behavior of smartphones is to auto-capitalize the first letter in standard text fields, which is usually desirable. However, disabling this auto-capitalization is preferable in a few cases, especially for email addresses, which most test subjects wanted to be in lowercase.

This subject noticed the capital "J" and went back to replace it with a lowercase "j" because he was unsure whether the capitalized version would work. (Large view)

Multiple times during testing, subjects noticed an uppercase letter and made an explicit effort to replace it with the lowercase equivalent. Most explained that they were unsure whether uppercase characters were allowed or whether email addresses in general were case-sensitive. On websites that had disabled auto-capitalization in the email field, no subject ever actively capitalized the first character. Disabling auto-capitalization for email and other appropriate fields (such as URLs) is recommended, then.

Among the top mobile commerce websites, 38% don't disable auto-capitalization on email address fields, leaving them as plain-text input fields, and leaving less technically inclined users in doubt.

Auto-capitalization can be disabled by adding the `autocapitalize` attribute to the `<input>` tag and setting it to `off`, like so:

```
<input type="text" autocapitalize="off" />
```

Of course, for email fields, you should set the `type` attribute to `email`:

```
<input type="email" autocapitalize="off" />
```

On iOS, setting the `type` to `email` will automatically disable auto-capitalization. However, still set the `autocapitalize` attribute because that will also work on iOS and might be needed on other platforms that do not yet support the `email` input type or that implement it differently.

Testing And The Cheat Sheet

While these fundamentals might seem obvious at first, remember that 98% of the world's largest mobile commerce websites violate at least one of these (see the complete list). And 70% get two or more of these "basic" touch keyboard guidelines wrong. In fact, 24% haven't optimized their inputs for touch keyboards at all, whether by omitting basic keyboard layouts (phone, email, numeric), invoking them inconsistently (or consistently poorly), not disabling auto-correction where appropriate, or not disabling auto-capitalization for email fields.

One reason for this lack of compliance might be that very thorough testing would be required to spot all of the pitfalls across a large website — hence, the third recommendation of invoking keyboard layouts consistently, which in an ideal world shouldn't even need to be mentioned. Another reason, mentioned in a prior Smashing Magazine article, is that **mobile and touch interfaces represent a relatively new platform**, with an entirely new interaction method that requires attention to a myriad of small details — details that we as Web designers and developers are not yet accustomed to actively looking and designing for.

For this reason, we'll end this article with a cheat sheet of the most common pitfalls when working with input fields for touch interfaces, along with copy-and-pastable code and a mobile touch-optimized demo of fields that invoke the correct keyboards, which you can use as a checklist when designing and developing mobile- and tablet-optimized websites.

- Interactive cheat sheet for touch keyboards, with mobile-optimized demo

These fields are typically included in the following types of forms: account registration, account sign-in, search, surveys, the entire checkout process, comment forms, and contact forms. We recommend searching your entire code repository to catch every single instance of these.

(al) (ea)

Christian Holst

Christian Holst is co-founder of Baymard Institute where he writes bi-weekly articles on web usability and e-commerce optimization. He's also the author of the E-Commerce Checkout Usability and M-Commerce Usability research reports.