buzztouch™

Installing, Using, and Creating buzztouch Plugins

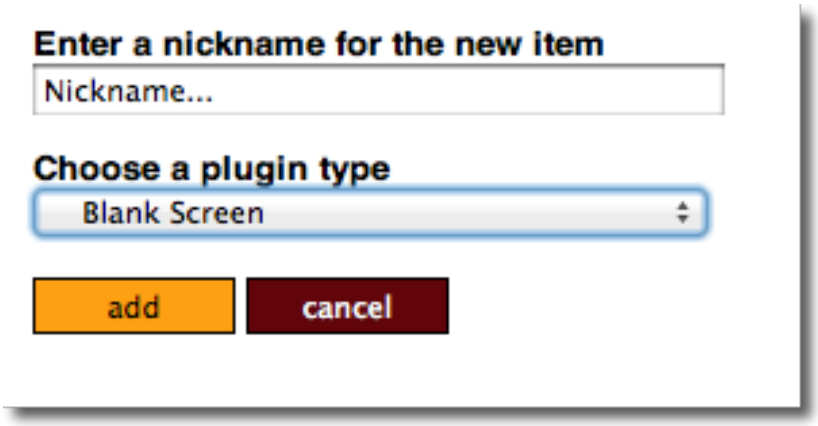
Table of Contents

About Plugins..... 3
  Why plugins are necessary..... 3
  What plugins do..... 4
  Who creates plugins..... 4
  How do I get new plugins..... 4
Managing Plugins in the Control Panel..... 4
  Installing plugins..... 5
  Updating plugins..... 5
  Removing plugins..... 6
  Plugins on the web server..... 6
Using Plugins in Applications..... 7
  Creating a new screen using a plugin..... 7
  Modifying a screen's behavior..... 8
Creating Plugins..... 9
  When to create a new plugin..... 9
  Things to consider when creating new plugins..... 9
  How plugins work, technically..... 9
  The plugin package..... 10
  A simple step-by-step example..... 11
Distributing Plugins 12
  How can I distribute a plugin I created..... 13
  Updating a plugin after it has been distributed..... 13
Conclusion..... 13

About Plugins

Every screen in a buzztouch app is derived from a plugin. For this reason, understanding what plugins do, how to find them, how to install them, and how to use them is important if you're to create anything beyond the most basic app.

Some plugins are simple, some plugins are complex, but all plugins help you extend the usefulness of an application. Using the right plugins and / or creating your own plugins helps you make your application do almost anything you can think of. If you can imagine it, a plugin can do it.



When you add new screens to an app, you start by choosing an existing plugin from a list in the control panel. The choices in the list are determined by the number of plugins installed in the Admin Panel of the software. Administrators determine which plugins can and cannot be added to individual applications. This means that if you want to use a plugin that does not show up in the list of choices you'll need to install that plugin in the Admin > Manage Plugins screen first. If you're not a system administrator you won't be able to do this.

You do not need to install a plugin in the control panel every time you want to use it, only the first time. Also, once the plugin is installed, it becomes available for use in any app in the control panel.

Why plugins are necessary

Plugins are necessary because there is no way to predict in advance what an applications purpose, intention, or audience is. Cookie-cutter apps and template driven approaches can cover lots of the situations but not all of them. The fact is, the best apps, the highest quality apps, and the most successful apps always include unique features that set it apart from all the others. We have spent years at buzztouch.com learning, discovering, and improving our concept of web-based app management. In this time we have gained a deep understanding of what works, what doesn't work, what developers want, and what end-users expect. An organized, efficient, useful, beautiful, and oftentimes fun app is the objective. A plugin architecture is our approach to helping you meet that objective.

Nearly every idea or feature you see in a mobile app is an extension of an existing idea. Very few apps are genuinely unique, most are improvements or

extensions of functionality found in other apps. For non-creative apps, such as a utility app used by a group of employees, necessary functionality becomes more predictable.

It is much more efficient for an app developer to implement customized features and functions by starting with an existing foundation of code. For example, creating a map highlighting local restaurants is not unique. However, implementing a new super-restaurant-locator with a new and exciting feature could help set the app apart from other existing apps. For the developer, it makes sense that the programming of the super-restaurant-locator feature would begin by leveraging code already written in a generic, boring map. Repurposing code is a significant time saver and a plugin architecture makes this possible.

What plugins do

From a technical perspective, plugins do two things.

1. First, they allow an app owner to manipulate certain properties of a screen by way of making choices in an online control panel.
2. Second, they provide the source-code for the app (Objective-C for iOS or Java for Android) when the project is packaged for download.

Who creates plugins

Anyone can create a plugin. It doesn't take much skill to create a new plugin when you start by copying an existing plugin. However, it does take some programming skill and experience to create sophisticated plugins. For this reason, it is very common for new developers to begin creating plugins only to find that they need help with some programming before they can finish perfecting their idea. This is normal and part of the learning process. And, it's a significant time saver for an app owner or inexperienced developer to be able to prototype their concept, complete parts of it, then hire out only the missing pieces. This is very difficult to do without a plugin architecture.

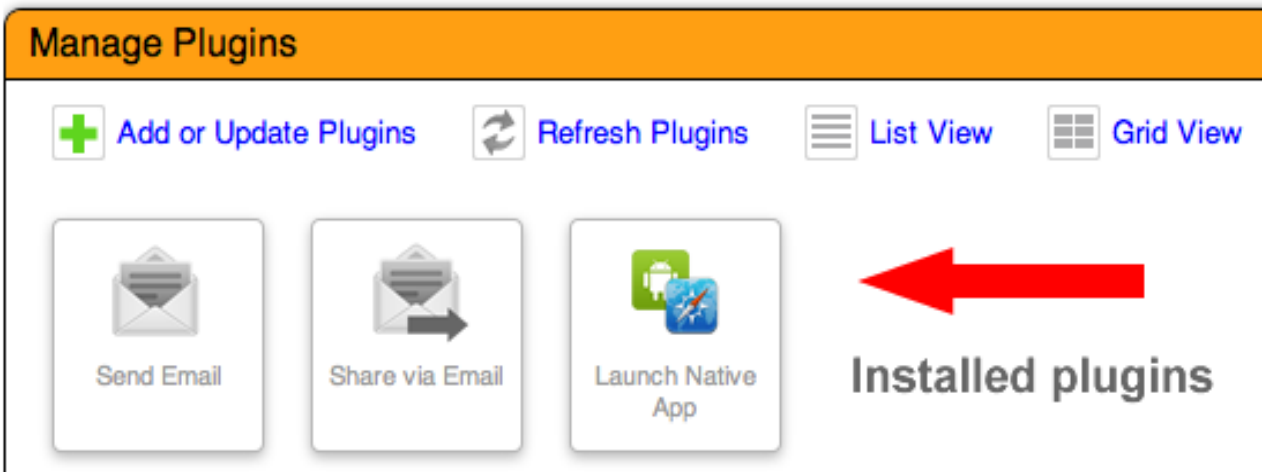
How do I get new plugins

In most cases plugins are discovered and downloaded from buzztouch.com. Third-party websites and other developers may also distribute plugins. It's also common for developers and app owners to share plugins amongst themselves. Where the plugin comes from is not important, how it works and functions is.

Managing Plugins in the Control Panel

The Manage Plugins screen (admin control panel) is used to see what plugins are installed and to install new plugins or to remove existing plugins. Choose “Grid View” or “List View” depending on your preference.

The title and icon for the plugin are provided by the plugin author. Clicking a plugin on the Manage Plugins screen shows additional details about the plugin such as it's purpose, author, screenshots, and other useful information.



Installing plugins

There are three steps to install plugins.

1) Download the plugin package(s) from buzztouch.com (or from any other location). Plugin packages are .zip archives and each .zip package contains multiple files. Do not unzip the archives after downloading them. Each .zip archive represents one plugin and no two .zip archives will have the same name.
2) Click the "Add or Update Plugins" link on the Manage Plugins screen. A screen will display that allows you to upload the .zip package(s). When you upload a .zip package, the control panel software will verify that the package contains all the necessary files required to make the plugin work. Continue this process until you have uploaded all the plugins you need to install. Note: If you upload a package that you previously uploaded you will need to check the "Update Existing Plugin" checkbox.
3) Click the "Refresh Plugins" link. The process syncs the control panel software with its supporting database. If you don't refresh the plugin list after uploading new packages the list of installed plugins will not be updated.

Updating plugins

The steps to updating a plugin are identical to the steps to install a plugin (see previous section). When you upload plugins that you have previously installed, you are overwriting the existing matching plugin with the update.

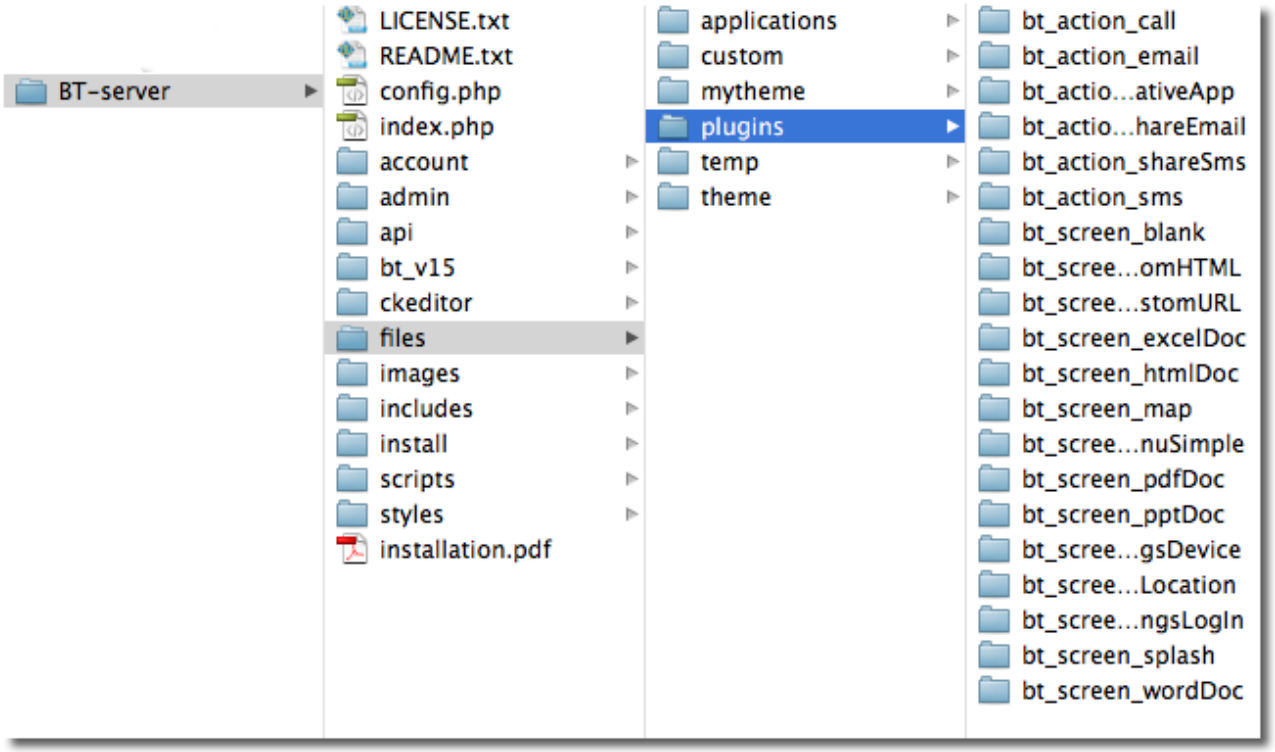
It's important to understand that updating plugins in the online control panel does not change anything in any apps that you have already compiled. Example: You create an app with a Map Screen then publish it and distribute it. If the Map Screen plugin is updated in the control panel, the apps that you have already distributed will still have the "old code" from the original Map Screen plugin. This may or may not matter, depending on the nature of the update. In some cases, it will be necessary to re-compile and re-distribute any apps that rely on the updated plugins code.

Removing plugins

To remove a plugin, click "List View" on the Manage Plugins screen then use the "remove" option. This option does not show in "Grid View".

Plugins on the web server

If you use an FTP program to browse the file system on your web server, the /files/plugins directory looks like this. Note the individual directories for each installed plugin. When you initially install the control panel software, no plugins will exist. This is what it looks like after you've installed a few.



When you click the “Refresh Plugins” link in the control panel, the software inspects the content of each of the folders (see previous image) then updates the supporting database accordingly.

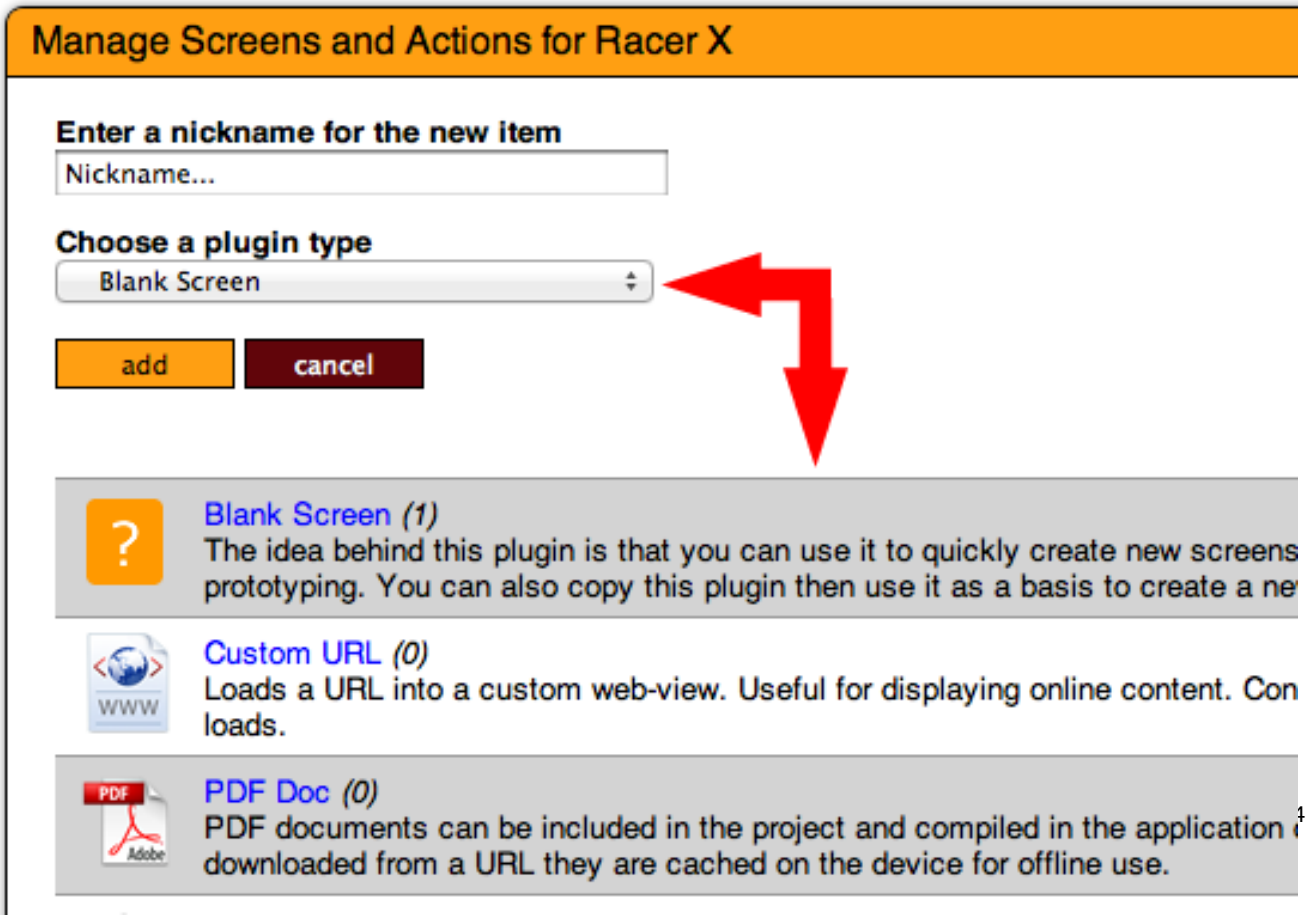
Using Plugins in Applications

Creating a new screen using a plugin

Plugins are used to create new screens in apps. Three steps to use a plugin...

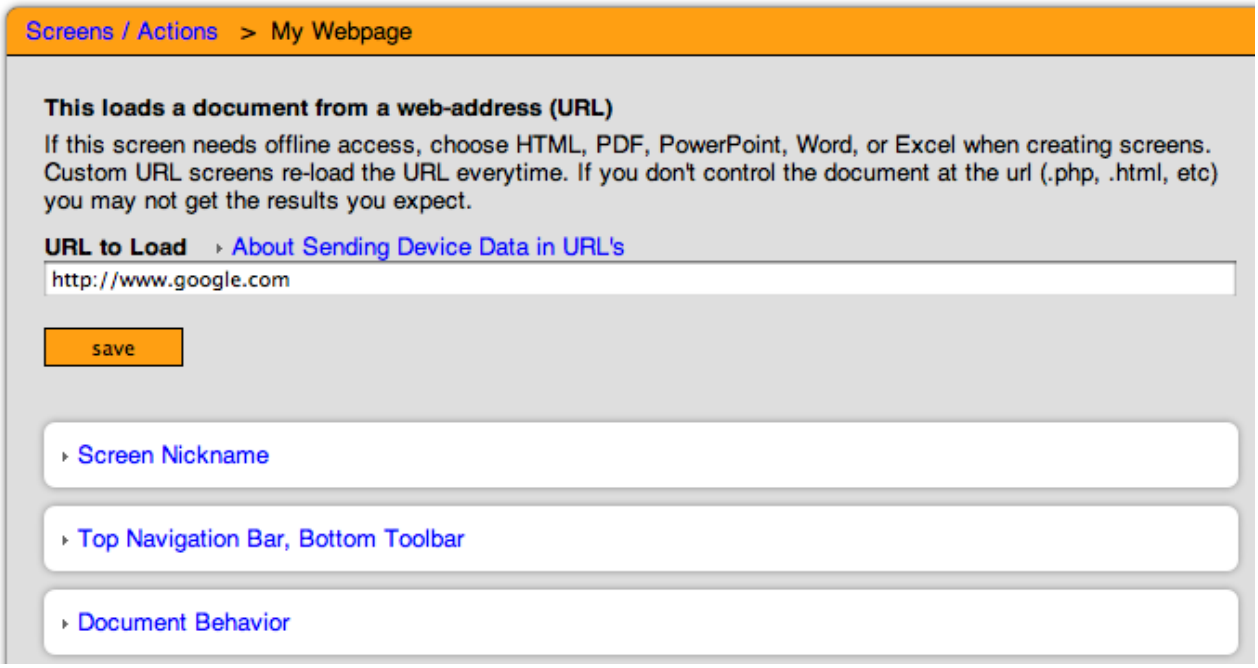
1) Click the “Screens and Actions” link in an apps control panel.
2) Enter a nickname for the screen you're about to create.
3) Choose a plugin type from the drop-down list. The choices in the drop-down list are derived from all the plugins installed in the Manage Plugins area of the admin panel. Information about each plugin type is displayed for reference. The number next to each plugin's name shows you how many screens in this application are based on that plugin type.

It's important to understand that most plugins require additional configuration after adding them to an app. Example: The Custom URL plugin in the example image will require an internet address to function. The number of properties and the types of information required to make a plugin work are determined by the plugin author. Some plugins have lots of properties, others have none. It all depends on the purpose of the plugin.



Modifying a screen's behavior

When you add a new screen to an application, its default behavior is determined by the plugin it is based on. The author of the plugin decides this default behavior. In most cases, the plugin will need additional configuration. Configuration options (properties) of each screen are adjusted using the online control panel. The sample below shows the properties for a screen named "My Webpage." Some plugins will have lots of adjustable properties and some plugins will have none.



When you "save" the properties of any screen and commit the changes to the database, it's likely that the application has changed (at least this screen has). It's also possible that you have already compiled and distributed the application you are now modifying. In this case, devices that have already installed the app will need a way to "learn" about the changes you made in the online control panel. In most cases this is handled automatically by the device and end-users will be prompted to "refresh" the apps data.

However, it's possible that the app owner compiled, then distributed the app without a connection to the online control panel. This could be intentionally or unintentionally. In either case, if the app is not connected to the online control panel, devices will have no way to know about the changes. In this situation, app updates are only possible by manually updating the apps configuration data, recompiling the project, then re-distributing the app to end users.

In some cases it's useful to look at the apps configuration data (use the

“configuration data” link in the apps control panel) after you make changes. This is useful in cases when you want to see how the data changes based on your selections.

Creating Plugins

When to create a new plugin

Creating new plugins is necessary when the available plugins will not accomplish what you need to do. Generally speaking, if you need a new screen type, you need a new plugin. In many cases a new plugin is not necessary because an existing plugin could be configured to behave in the desired manner.

Things to consider when creating new plugins

There are a few things to think about when you're considering creating a new plugin. Among other things, consider this...

1) Does the new plugin need to work in multiple apps in the control panel or is it very specific to one app?
2) Will you be sharing or distributing the plugin so other developers can use it?
3) Do you have the skill to create the new plugin or will you need to recruit or hire some help?
4) Should the new plugin be based off an existing plugin or should it be created entirely from scratch?
5) Will the new plugin require database or backend support to function?
6) Will the new plugin require an internet connection?
7) Does the new plugin rely on images, audio, video, or file system assets?

How plugins work, technically

Plugins are not as sophisticated as some folks want to believe. For an experienced developer, the concept makes perfect sense. Aspiring developers tend to confuse the idea that because there are so many files in a plugin package, they must be very complex.

The easiest way to think about it is to consider what's happening when a user adds a new screen to an app then downloads the source code. Fundamentally, two things happen.

1) An app owner adds a new screen to an application and the new screen is added to the database. The newly added screen uses the default configuration data provided by the plugin. App owners may or may not provide additional configuration data for the plugin. If they do, that too is

saved to the database.

2) When the source code for an application is downloaded, each screen in the database is inspected and processed. Because each screen is derived from a plugin, that plugin is responsible for providing the iOS and Android source code that is included in the Xcode or Eclipse project.

For sure it's a sophisticated process and lots of time went into engineering it. But, it's not that complicated once you understand how it works. The key is that each plugin must contain some required files in its directory so the control-panel can understand how to work with it.

The plugin package

Every plugin package (directory) contains 6 required files and 2 required sub-directories. It's important that the file names of these required files match precisely what is displayed in the graphic. File names are case sensitive and must include the file extension.

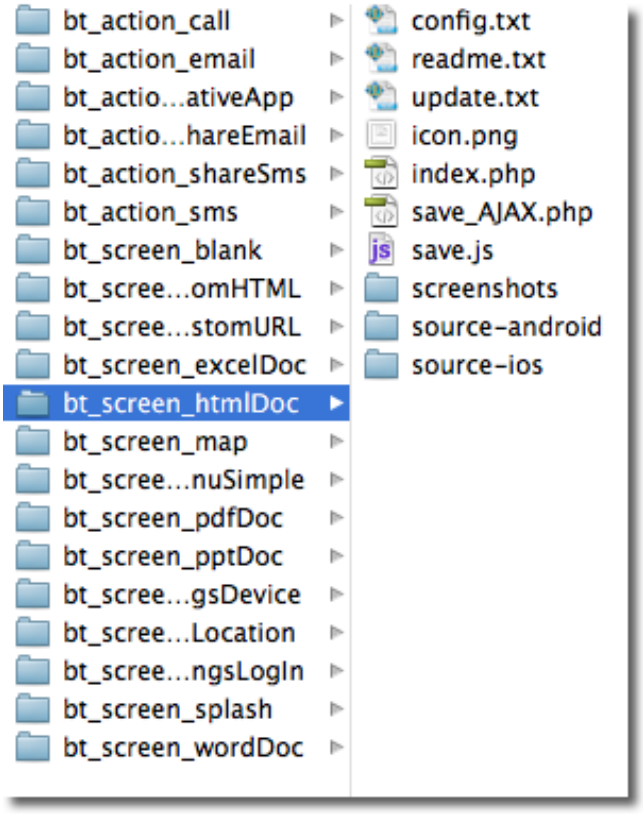
The required files are:

config.txt, readme.txt, icon.png, index.php, save_AJAX.php, save.js, /source-android, /source-ios

Optional files are:

update.txt, /screenshots

Side note: As mentioned previously, it's usually best to begin making a new plugin by copying the entire directory of an existing plugin. Next, rename the new directory so it's unique. This approach ensures that you have the required files named properly.



Each required file or sub-directory in the plugin package has a specific purpose.

config.txt: This file contains important information about the plugin used by the control panel for several things. The config.txt file serves as a sort of manifest for

the plugin and it's essential that its information be accurate.

readme.txt: This file contains information intended for humans. Its purpose is to explain in plain-language how the plugin works, what its purpose is and why somebody would want to add it to their control panel. In essence, the readme.txt file serves as the instruction manual for the plugin.

icon.png: This is a 50 x 50 .png image that displays in the online control panel.

index.php: This is a web page that app-owners use to configure the plugins properties. When an app owner "clicks a screen" to configure it, the index.php page loads and presents them with configuration options in an HTML form.

save.js: This is a javascript file that is included in the HTML markup in the index.php web page. "Saving" plugin properties triggers a method in this file that POSTS the values in the FORM element in index.php. It posts these values to the save_AJAX.php script.

save_AJAX.php: This file is responsible for receiving the app owners inputs (from the FORM included in index.php) and committing them to the database.

/source-android: This sub-directory contains all the required resources for the Android version of the plugin to work. This is usually only two files but may be more. The two required files in this folder are the Android Activity Class (a .java class file) and the Android Layout file (a .xml layout file). Additional files in this folder may be graphics, audio, and other supporting .java code or .xml layout code.

/source-ios: This sub-directory contains all the required resources for the iOS version of the plugin to work This is usually only two files but may be more. The two required files in this folder are the UIViewController's .m and .h files. Additional files in this folder may be graphics, audio, and other supporting Objective-C objects necessary to support the UIViewController.

Optional files

/screenshots: This optional sub-directory contains screenshots displayed in the control panel. When creating screenshots, export them as lightweight .png files. If you screen-capture the simulator running you'll end up with images around 396 x 744 which is appropriate. Do not include any thumbnail versions of the screenshots, the control panel will create these automatically.

update.txt: This optional file is used when a user clicks the "Check for updates" option when viewing a plugins details. This concept will be discussed later in this document.

A simple step-by-step example

Because the best way to create a new plugin is to begin by copying an existing plugin, the process can be broken down into a few simple steps. In this simple

example we will create a new plugin called “db_screen_sample” to illustrate how easy this can be. Keep in mind that this plugin won't do much but it will demonstrate the process. The following 6 steps can be completed in :05 minutes or so with some practice.

1) Copy the bt_screen_blank plugin (the entire folder) and paste it in your documents directory. Rename the folder my_first_plugin.
2) Expand the my_first_plugin folder. Open the config.txt file in your favorite text editor. Change the uniquePluginId to: emptyScreen. Change the displayAs to: Empty Screen. Change the loadClassOrActionName to: my_first_plugin. Change all the author information to your information. Remove the updateURL and the downloadURL. All of the elements still exist but have new values. updateURL and downloadURL are blank. Save and close this file.
3) Open the readme.txt file in your favorite text editor. Replace all the occurrences of BT_screen_blank with my_first_plugin then save and close this file.
4) Open the icon.png in your favorite image editor. Modify this image so it's unique and meaningful. For this example a simple question mark is appropriate. Save and close your image editor.
5) Expand the source-android folder. Rename the screen_blank.xml file to my_first_plugin.xml, rename the BT_screen_blank.java to my_first_plugin.java. Open the my_first_plugin file in your favorite text editor. Replace all occurrences of BT_screen_blank with my_first_plugin. Replace R.layout.screen_blank with R.layout.my_first_plugin then save and close this file.
6) Expand the source-ios folder. Do the same thing with the Objective-C code that we did to the Android code. Rename BT_screen_blank.h to my_first_plugin.h, rename BT_screen_blank.m to my_first_plugin.m. Open both of these files in your favorite text editor and replace all occurrences of BT_screen_blank with my_first_plugin.

That's all there is to it, you've created an entirely new plugin based on an existing plugin. In a real-world situation you would then create unique functionality in the Objective-C and .java files, some new screenshots, and modify the index.php to accurately reflect the plugins purpose. But, the idea is the same, copy an existing plugin, rename some files and change some values, then go to work creating the unique aspects of your idea without starting from scratch.

Distributing Plugins

You may or may not wish to distribute or share plugins that you create, it's

entirely up to you. If you do plan to distribute a plugin, it's important to consider two concepts. Distribution and updates.

How can I distribute a plugin I created

You can distribute your cool plugin in a few ways. You can email it as a .zip archive to other developers, you can send a .zip archive to buzztouch.com and ask that it be included in the growing list seen by hundreds of thousands of site visitors, or you can upload a .zip archive to a web enabled directory (like DropBox or your website) then promote the URL to others so they can download it. The best method depends on your motive. Note: If you ask us to display your plugin on buzztouch.com we may or may not do it. It depends on the quality, usefulness, and nature of the plugin. We'll do what we can but make no promises.

Updating a plugin after it has been distributed

App owners need a way to update plugins in their control panels. This means they need a way to learn that an update is necessary. Plugin updates are common because things tend to move very quickly in mobile and developers make improvements constantly. The best way to manage this is:

1) Distribute your plugin by way of a download from a URL.
2) Include the downloadURL in the config.txt file. Also include the update URL in the config.txt file. We removed these in the step-by-step example (earlier in this document) for simplicity.

Here's the idea. When an app owner installs the plugin, then opens the plugin details screen to view that plugins info, they are presented with a “check for updates” option IF the config.txt file for the plugin includes an updateURL. The updateURL generally points to a simple text file on a web enabled folder somewhere (DropBox again) and this text file includes a simple “version string.” When the app owner checks for updates, this URL returns the most current version string. If the string is different than the version string in the already installed plugin, an update is necessary. The “download latest package” link in the control panel opens the downloadURL in the config.txt file. This leads to the plugins .zip archive.

This simple approach allows app owners to download the latest version of your plugin only when necessary and relieves you from having to “tell everyone” that your plugin has been updated.

Conclusion

Congratulations for taking the time to get through this lengthy document. We believe strongly that usefulness, flexibility, and creativity provided by learning

how to manage plugins is well worth the read. We encourage you to begin making as many plugins as you need, sharing them with the buzztouch community, and continuing to expand your mobile development arsenal.