Plugins

Market

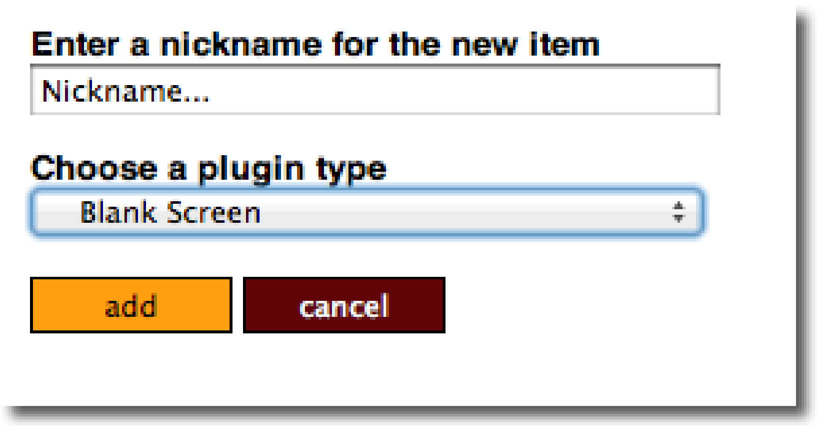
Creating, Sharing, and Selling Buzztouch Plugins

Table of Contents

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 want 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 your control panel. On buzztouch.com control panels, individual users determine which plugins are installed in their control panels. On self hosted servers, the installed plugins are controlled in the Admin Panel of the software. This means that if you want to use a plugin that does not show up in the list of choices in your control panel, you'll need to install that plugin before you can use it in an application. After you install a plugin, it is available for use in all of the applications in your control panel. NOTE: If you install a new plugin after downloading the source code for one of your projects, that project WILL NOT contain the new plugin. This means you will need to re-download the source code package for that application then update the application (re-compile it in Xcode or Eclipse) so it includes the new plugin source code. This is an important concept that many new app owners struggle with.

What plugins do

From an app owners perspective...

1. Plugins allow app owners an almost endless amount of flexibility when creating mobile apps.
2. Plugins allow app owners to efficiently and effectively create compelling native apps without learning complex programming languages.
3. Plugins allow app owners to save an tremendous amount of money when making native mobile apps (compared to hiring a developer to create an app from scratch).

From a developers perspective...

1. Plugins allow developers and programmers to extend the functionality of the Buzztouch platform in an endless number of ways.
2. Plugins allow developers and programmers to earn revenue by selling their creations. This may be in the buzztouch.com Plugin Market or on a third party website or blog.
3. Plugins allow developers and programmers a way to extend their mobile development skills, gain exposure and credibility, and to showcase their professional work.

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 the Buzztouch plugin architecture makes this possible.

Who creates plugins

Anyone can create a plugin. In most cases, new plugins are created using the Plugin Creator tool found in your account control panel at buzztouch.com. The Plugin Creator Tool is only available to Buzztouch Members, not guests. However, it is possible, and common, to create new plugins without using the Plugin

Creator tool at buzztouch.com. In this case, it's a matter of understanding what constitutes a Buzztouch plugin “package” and assembling that package manually. In most cases, people that create compelling plugins identify themselves as developers or aspiring developers.

NOTE: Plugins that are created manually (without using the Plugin Creator tool) cannot be uploaded to a Buzztouch control panel and cannot be distributed in the Buzztouch Plugin Market. This means that a plugin you create manually (without using the Plugin Creator tool) will only be useful to app owners running their own Buzztouch self hosted control panel.

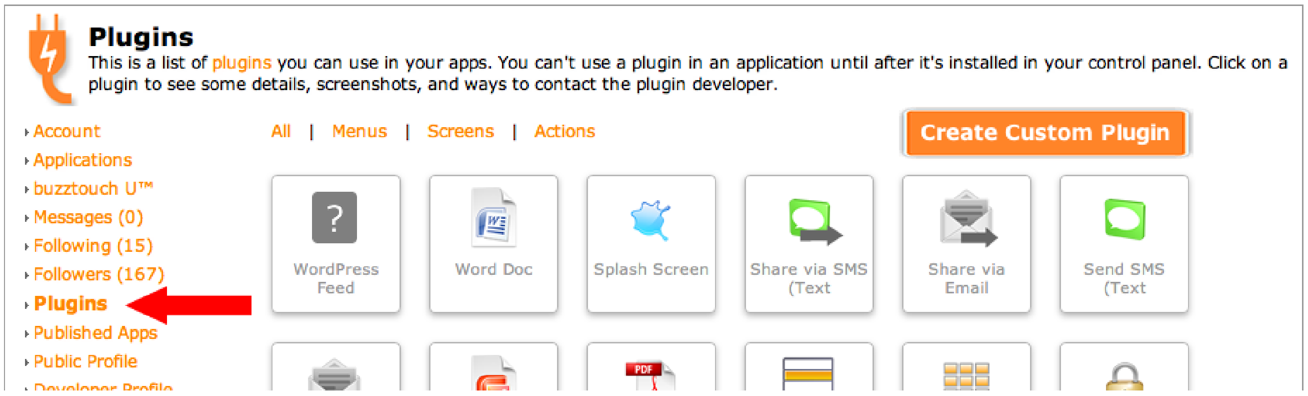
It does take programming skill and experience to create sophisticated plugins. For this reason, it is very common for new or aspiring developers to begin creating plugins only to find that they need help with some programming before they can finish perfecting their idea. This is normal and part of the learning process. In many cases, plugins are created by a non-developer, then completed by a developer hired to help complete a mobile app project.

How do I get new plugins

In most cases plugins are discovered and installed in a Buzztouch control panel by way of the Buzztouch Plugin Market. Plugins in the market can also be downloaded for use in self hosted control panels. Third-party websites and other developers may also distribute plugins.

Managing Plugins in the Buzztouch control panel

The list of plugins available in a Buzztouch control panel is managed using the Plugins screen. Each plugin icon on this screen represents a plugin that is installed.


Plugins

This is a list of plugins you can use in your apps. You can't use a plugin in an application until after it's installed in your control panel. Click on a plugin to see some details, screenshots, and ways to contact the plugin developer.

Installing plugins in a buzztouch.com control panel

There are two ways to add a plugin to a buzztouch.com control panel.

1. By using the "Install" option available for any plugin displayed in the Plugin Market (this may or may not require a fee).
2. By clicking the Create Custom Plugin option and creating a new plugin using the Plugin Creator tool (Buzztouch Members only, Guests cannot do this).

Updating plugins in a buzztouch.com control panel

Plugin updates are common and necessary and two things happen when an update it released.

1. buzztouch.com control panels using the plugin will be updated to the new version. App owners that previously installed the plugin in their control panel will have access to the new version.
2. App owners may or may not need to re-compile their application(s) source code that uses the plugin. This will depend on the nature of the update. In most cases a re-compile will be necessary.

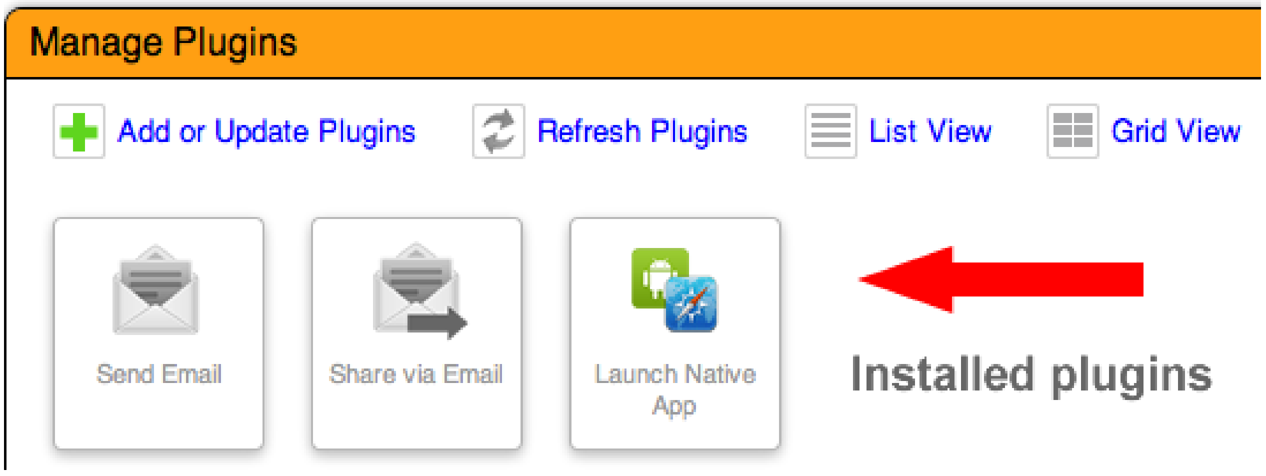
Removing plugins from a buzztouch.com control panel

To remove a plugin from a buzztouch.com control panel.

1. Visit the Plugin screen in the control panel by clicking the plugin's icon.
2. Click "Remove from Control Panel" Note: Apps using this plugin that have already been downloaded and compiled will continue to work.

Managing Plugins in a Self Hosted control panel

The Manage Plugins screen (available to admins) is used to see what plugins are installed and to install new plugins or to remove existing plugins.



Installing plugins in a self hosted control panel

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 in a self hosted control panel

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.

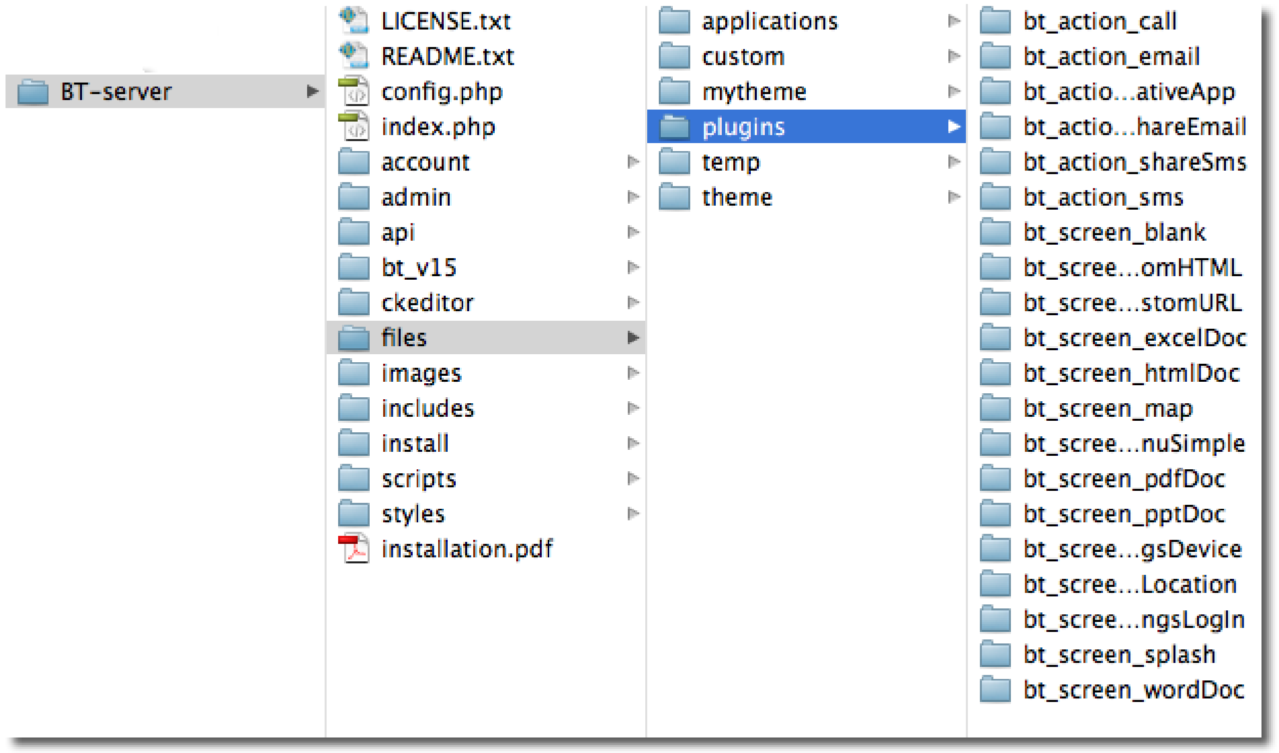
It's important to understand that updating plugins in the online control panel does not change anything in any apps that you have already compiled. Example: You create an app with a Map Screen then publish it and distribute it. If the Map Screen plugin is updated in the control panel, the apps that you 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 from a self hosted control panel

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 self hosted control panel's 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.

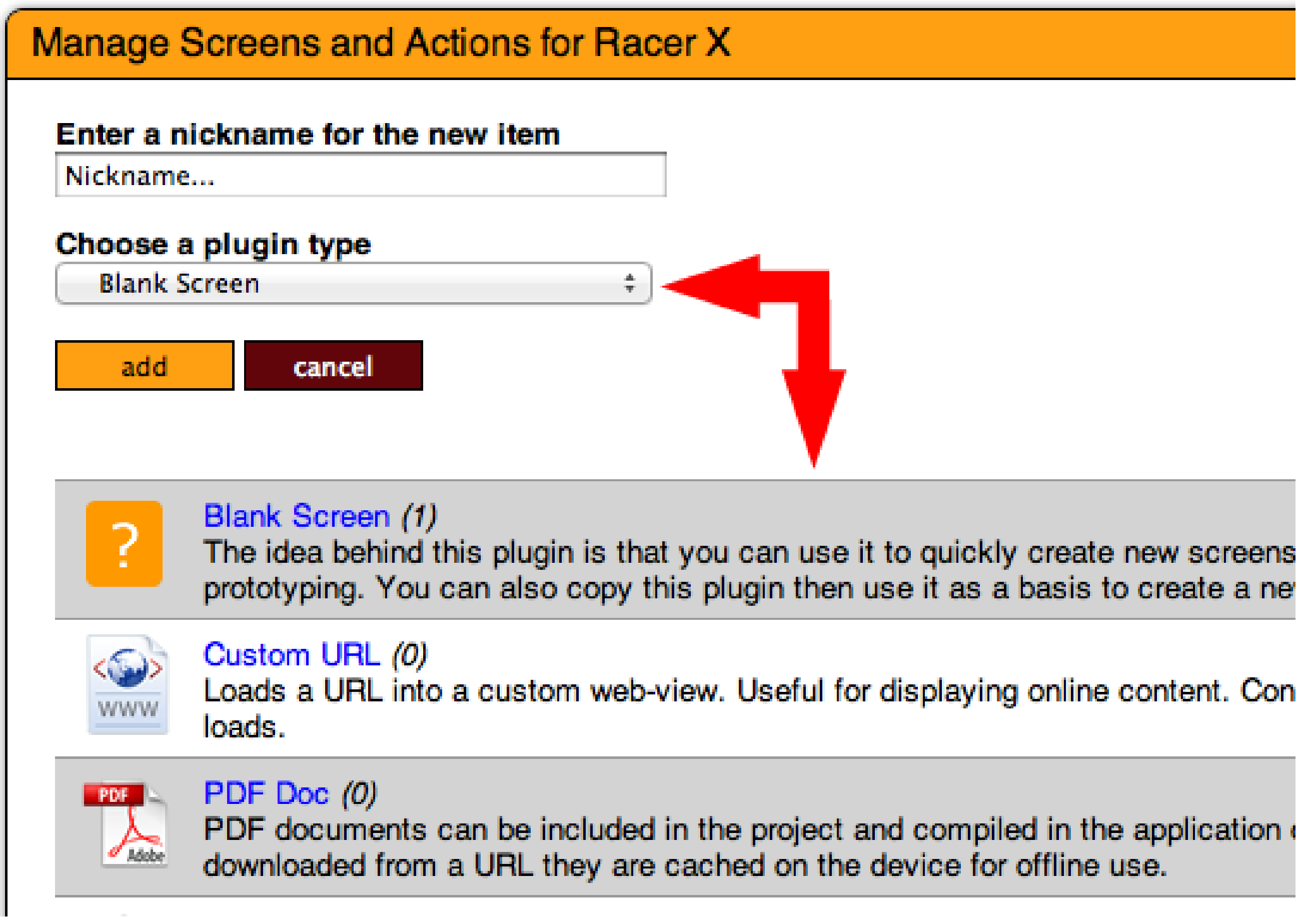
Using Plugins in Applications

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” 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.

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 developer. Some plugins have lots of properties, others have none. It all depends on the purpose of the plugin.


Manage Screens and Actions for Racer X

Enter a nickname for the new item

Nickname...

Choose a plugin type

Blank Screen

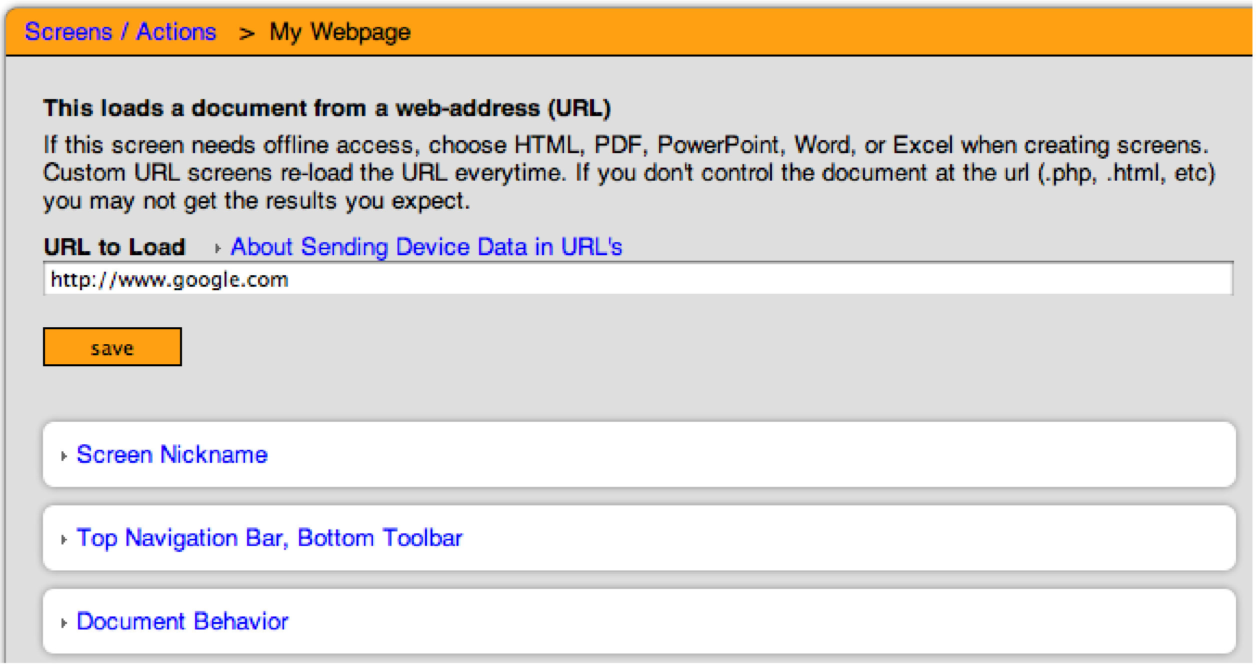
Blank Screen *(1)*
The idea behind this plugin is that you can use it to quickly create new screens
prototyping. You can also copy this plugin then use it as a basis to create a ne

Custom URL *(0)*
Loads a URL into a custom web-view. Useful for displaying online content. Con
loads.

PDF Doc *(0)*
PDF documents can be included in the project and compiled in the application
downloaded from a URL they are cached on the device for offline use.

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 plugin developer determines 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 flexible properties and some plugins will have none.


Screens / Actions > My Webpage

This loads a document from a web-address (URL)

If this screen needs offline access, choose HTML, PDF, PowerPoint, Word, or Excel when creating screens. Custom URL screens re-load the URL everytime. If you don't control the document at the url (.php, .html, etc) you may not get the results you expect.

URL to Load About Sending Device Data in URL's

http://www.google.com

save
Screen Nickname
Top Navigation Bar, Bottom Toolbar
Document Behavior

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 idea of your new plugin justify an entirely new “screen type” or is your idea related to something other than loading a new screen type. Not all app improvement ideas are related to loading a “new screen.”
2. Does the new plugin need to work in multiple apps in the control panel or is it very specific to one app? You may be better off creating a simple one-time screen manually after you download the source code for a project. If you don't plan to re-use the screen type, it may not be a good candidate for an entirely new plugin.
3. Will you be sharing or selling the plugin so other app owners or developers can use it?
4. Do you have the skill to create the new plugin or will you need to recruit or hire some help?
5. Will you be using the online Plugin Creator tool to create the new plugin or will you be creating it from scratch?
6. Will the application(s) using the new plugin require an internet connection to work?
7. Does the new plugin rely on images, audio, video, or file system assets?
8. Does the new plugin need to work on multiple device sizes (small, large)?
9. Does the new plugin need to work on multiple platforms (iOS and Android)?

Asking yourself these questions will help you develop a strategy for taking the appropriate steps towards developing, or hiring someone to develop the idea.

Selling or sharing plugins in the Buzztouch market

Plugin developers may choose to create plugins in an effort to earn money. This is OK as long as the rules and guidelines explained in the Buzztouch Plugin Developers Terms of Service. These terms can be found at the following URL:

http://www.buzztouch.com/pages/terms-plugin-developer.php

Updating a plugin in the Buzztouch market

App owners expect plugin updates and updates are common because things tend to move very quickly in mobile and developers make improvements constantly. Submitting an update to an existing plugin is as easy as re-uploading a newly updated version to the control panel.

Updating a plugin hosted on a third party website or blog

The update process is slightly different for plugins that are not distributed using the Plugin Market (these types of plugin are used only by app owners running a self hosted control panel).

1) Distribute your plugin by way of a download URL. This is normally a web address on your website such as http://www.site.com/myCoolPlugin.zip
2) Include the downloadURL in the config.txt file. Also include the updateURL in the config.txt file.

When an app owner installs the plugin in their self hosted control panel, then opens the plugin details screen to view that plugins information, they are presented with a "check for updates" option if the config.txt file for the plugin includes an updateURL. The updateURL points to a simple text file on a web server somewhere (your website) 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.

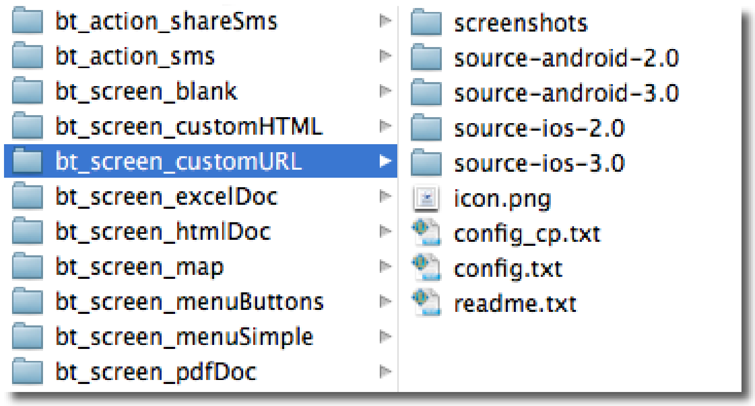
This simple approach allows app owners to download the latest version of your plugin only when necessary and relieves you from having to "tell everyone" that your plugin has been updated.

The plugin package

Every plugin package begins the same way. This is true if the plugin was created using the Plugin Creator tool, or if the plugin was created manually. Either way, the package is the same. A plugin package contains some standard files and several optional sub-directories. It's important that the file names of the standard plugin files match precisely what is displayed in the graphic. File names are case sensitive and must include the file extension. A plugin may optionally contain a number of additional files, media, and resources as determined by the plugin developer.

Standard Plugin files are:

config.txt, config_cp.txt, readme.txt, icon.png, /source-android-VERSION, /source-ios-VERSION, /screenshots



Each standard 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.

config_cp.txt: This is a simple text file that describes how the control panel should render the plugin management screen (used by app owners).

/source-android-VERSION: 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. IT IS IMPORTANT TO UNDERSTAND THAT THE ANDROID FILES ARE BROKEN INTO SEVERAL SUB-DIRECTORIES. EXPLORE AN EXPSITING PLUGIN PACKAGE TO SEE THIS STRUCTURE. The structure is not arbitrary, it matches the structure of a typical Android project.

/source-ios-VERSION: 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.

/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.

Understanding the config_cp.txt file

The webpage that shows in the control panel is configured by way of rules contained in the config_cp.txt file. This file has a list of “sections” that show on the webpage. Each section represents an option or value that the app owner can adjust. Some sections are standard, others are custom. Most plugins contain a mixture of standard and custom sections.

```
{"propertySections":[
	{"fileType":"bt_section", "fileName":"btSection_navBar.html"},
	{"fileType":"bt_section", "fileName":"btSection_login.html"},
	{"fileType":"custom", "fileName":"myCustomSection.html"},
	{"fileType":"bt_section", "fileName":"btSection_backgroundColor.html"},
	{"fileType":"bt_section", "fileName":"btSection_backgroundImage.html"},
	{"fileType":"bt_section", "fileName":"btSection_search.html"},
	{"fileType":"bt_section", "fileName":"btSection_tabBar.html"},
	{"fileType":"bt_section", "fileName":"btSection_screenJson.html"}

	]
}
```

Note the standard sections, along with one custom section. The myCustomSection.html file would be included in the plugin package. This was created by the plugin developer. This is NOT a complete .html web page, it is only a snippet of HTML. The control panel will include this snippet when the app owners lands on the webpage used to control the plugin.

Standard Sections. Include these without writing code.

btSection_ads.html
btSection_backgroundAudio.html
btSection_backgroundColor.html
btSection_backroundImage.html
btSection_childItems.html
btSection_dataURL.html
btSection_documentBehavior.html
btSection_login.html
btSection_menuListLayout.html
btSection_menuListRows.html
btSection_navBar.html
btSection_nickname.html
btSection_screenJson.html
btSection_search.html
btSection_tabBar.html

When creating custom sections it's normally best to copy a standard section and modify it to your needs.

Custom Sections use Javascript to modify control panel data. The landing page for a plugin is hosted by the control panel and includes sections for each plugin. This landing page exposes some Javascript variables and functions that plugin developers use to modify data.

Javascript Variables available to developers when creating custom sections.

userGuid	the unique id of the user using the control panel.
AppGuid	the unique id of the application in the control panel.
BT_itemId	the unique id of the screen in the control panel.
screenDataURL	the URL to this screens data if childItems are used.
JSONString	the complete JSON string of data configured for this screen.

It's necessary to know these values when working with the javascript functions.

fnFillFormValues(JSONString)

This method is used to pre-populate the form field elements on the landing page. This is normally fired when the page first loads so the app owner sees their previously saved settings.

fnExpandCollapse(hideOrExpandElementId)

This method is used to expand or collapse an HTML container. Consider a n html div or paragraph element that expands and collapses when clicked.

function saveJSONProperties(showResultsInElementId)

Triggering this method will save all the users selections to the database. The properties and values that are saved are mapped directly to the form field elements on the landing page. All form elements who's names are prefixed with "json_" will be saved. Example: json_dataURL or json_firstName. The dataURL and the firstName entry would be saved to the backend. In almost all cases the submit button's in each section call this routine to save all the user selections.

fnSetSelectedIndex(theEl, theValue)

A simple helper routine to manually set the selected index of a drop down element. An HTML <select> element.

fnPickColor(formFieldId)

This opens the color picker screen. When a color is selected the value is inserted into the form field on the underling landing page.

fnPickScreen(BT_itemIdElementName, nicknameFormElementName)

This opens the screen picker. This is normally used when app owners select screens to use on button clicks.

fnPickMenu(BT_itemIdElementName, nicknameFormElementName)

This opens the menu picker. This is normally used when app owners select context menus on individual screens.

fnPickFileName(BT_itemIdElementName, nicknameFormElementName)

This opens the file picker. This is normally used when app owners select file names for images and other file based assets.

fnPickFileURL(BT_itemIdElementName, nicknameFormElementName)

This opens the file picker. This is normally used when app owners select file URL's for images and other file based assets.

fnExecuteBackendCommand(theCommand)

This method is used to interact with backend data. The command value you pass can be one of several options. Options include:

getPluginOptions	Returns a list of installed plugins for the user.
getChildItems	Returns a list of JSON objects for childItems.
addChildItem	Used to insert new childItems for a screen.
removeChildItem	Used to remove a child item for a screen.
updateChildItemsOrder	Used to modify the orderIndex of child items.

In most cases it's best to review how other existing plugins use these methods before trying to use them yourself (unless you're an experienced javascript dev). The most complicated set of routines are related to childItems. Screens that display a list of child items are more complex than other screens and as such the plugin developer needs to understand how to add / remove items on the screen.