The Tao of loadScreenObject

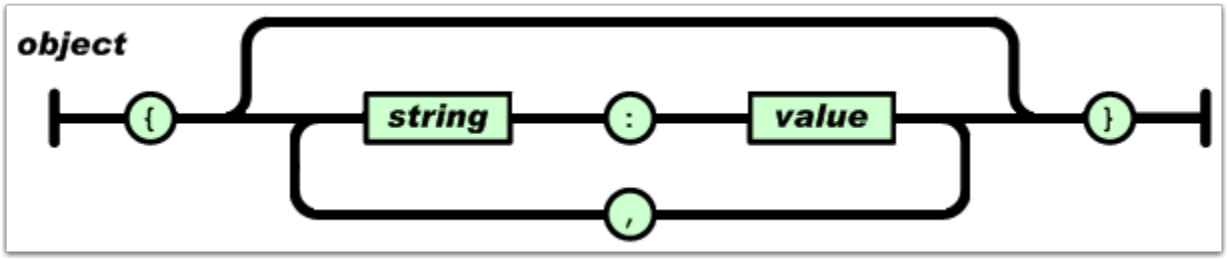
Document Notes

This revision: v1.0

Revision Notes:

1.0 - July 13, 2014 First Draft Created

What you need to know before I can tell you what you need to know

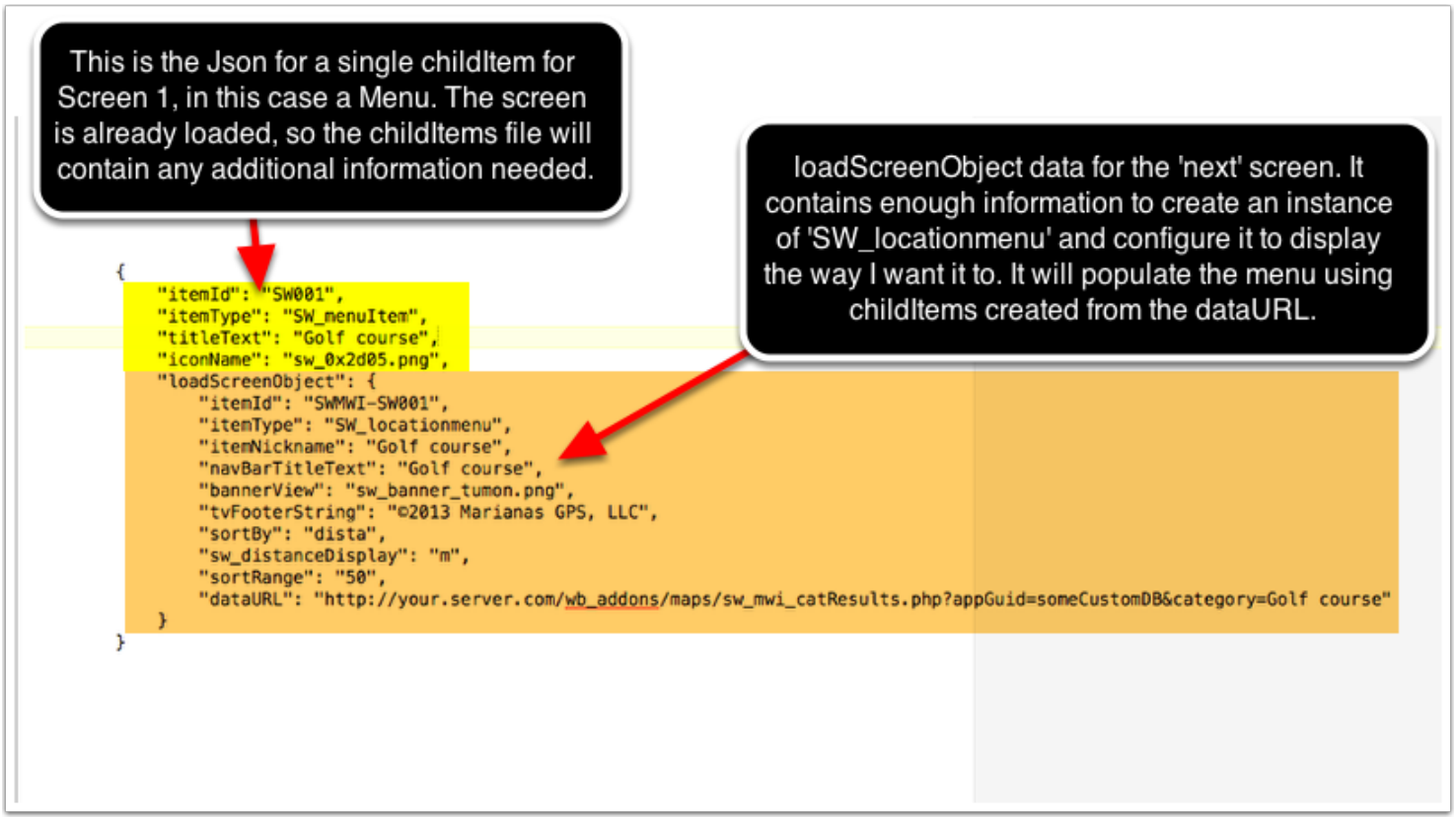


Everything centers around '**objects**'. If you understand the concept of objects, great. If you don't understand objects, this will either help you, or it won't make any sense at all. Hopefully it will help you.

You need to have at least a 'grasp' on what **JSON** is and what it does.
You need to have at least a 'grasp' of **PHP** scripting, and what it does.
You need to have at least a 'grasp' of **MySQL** and how to administer the basics.
You need to have at least a 'grasp' of **HTML** and how to display web pages.
You need to have at least a 'grasp' of **CSS** and it's use within HTML.

You don't need to be an expert, but you need to understand the concepts. Especially if you plan to modify scripts for your own specific needs.

What is loadScreenObject?



In the most simple explanation I can think of:

Load Screen Object is the method in which we embed 'Next Screen' data into the first Screen's childItems.

It provides 'most' of the same information as if it were created in the BT Control panel, however it is not included in the BT_config.txt file, and it is 'usually' created dynamically as part of a PHP/MySQL Script.

Reasons to use loadScreenObject

You might be wondering what are some of the advantages of using 'loadScreenObject' over conventional methods.

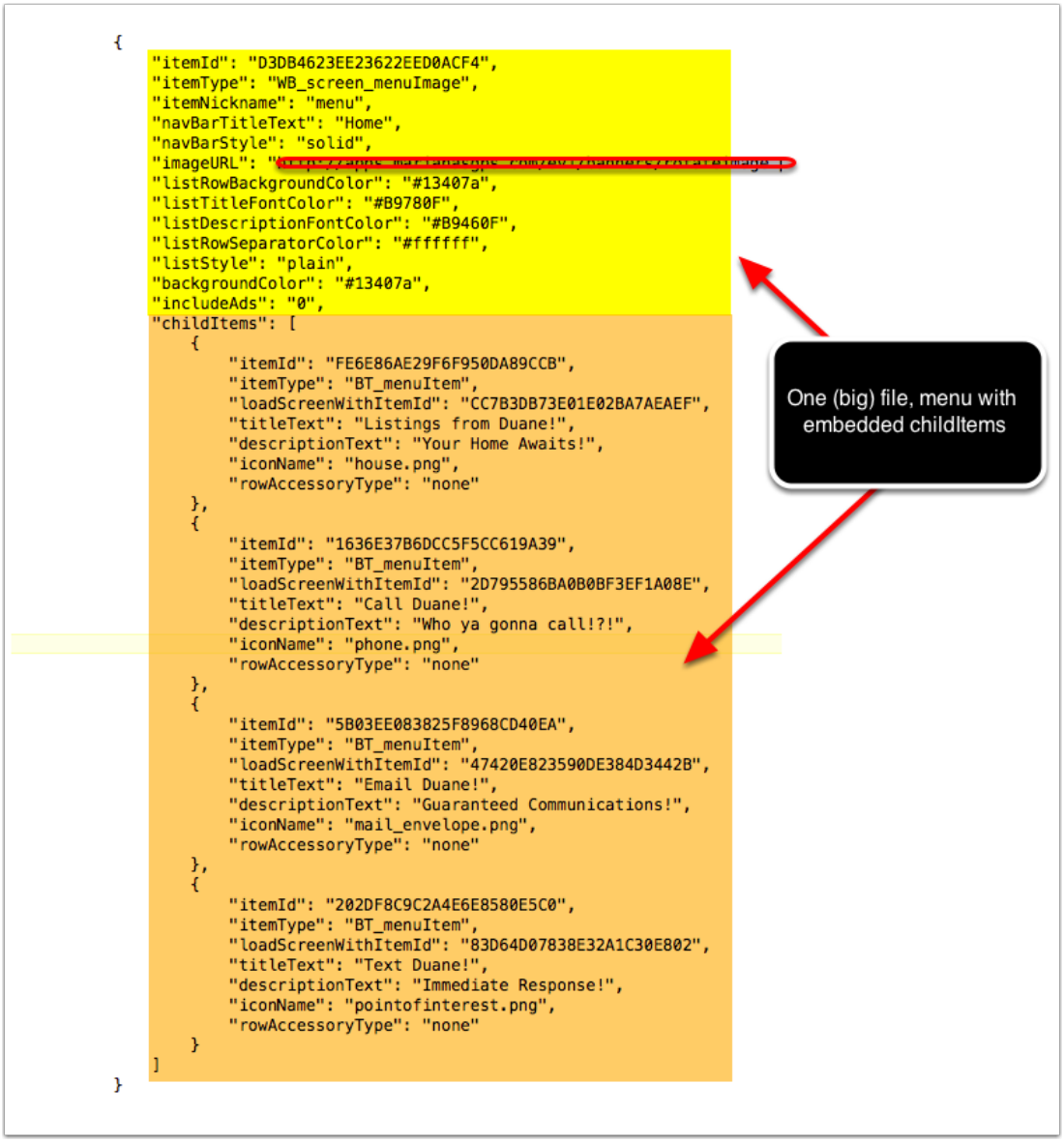
- Dynamic Creation of screens, based on user input.
- Create one template for thousands of similar screens
- Easier 'global' updates for screens. One template edit, rather than one for each screen.
- Smaller BT_config.txt file means a speedier app with faster response than a control panel loaded BT_config.txt file.
- Easier modifications and duplication to create other screens. Why reinvent the wheel?

Consider the app that is location based. Lets say you have 20 locations. Typically you would create one Map screen per location, and enter in the appropriate data for each location in the respective control panels. As well, you probably want or need to create some kind of Screen to display data about each location. We'll say as an example, an htmlDoc Screen. Each would have a unique page created to display information particular to an individual location. So at this moment, you have a menu screen, and at least 40 screens that you create by hand in the control panel.

Now consider the same location based app using loadScreenObject. Coupled with a PHP Script and a MySQL database to either use as a back end, or as a childItem file creator, you can create a script that pulls information from the MySQL Database, and creates your childItem files that is passed to your menu. When the user taps a menu choice, the loadScreenObject data will create the new screen, and allow a dataURL to populate the childItem data for the new screen with additional information. You have effectively created 40 screens dynamically. With the bonus that, any new information added to the database can automatically create new screens for you as you go along.

If you don't have access to a php server with mySQL, you could still create this scenario with a CSV file and a few utilities, albiet a little more manually, but still far more convenient that using the control panel for each and every screen.

A quick word about childItems, and external childItem files...




```
{
    "itemId": "D3DB4623EE23622EED0ACF4",
    "itemType": "WB_screen_menuImage",
    "itemNickname": "menu",
    "navBarTitleText": "Home",
    "navBarStyle": "solid",
    "imageURL": "[redacted]",
    "listRowBackgroundColor": "#13407a",
    "listTitleFontColor": "#B9780F",
    "listDescriptionFontColor": "#B9460F",
    "listRowSeparatorColor": "#ffffff",
    "listStyle": "plain",
    "backgroundColor": "#13407a",
    "includeAds": "0",
    "childItems": [
        {
            "itemId": "FE6E86AE29F6F950DA89CCB",
            "itemType": "BT_menuItem",
            "loadScreenWithItemId": "CC7B3DB73E01E02BA7AEAEF",
            "titleText": "Listings from Duane!",
            "descriptionText": "Your Home Awaits!",
            "iconName": "house.png",
            "rowAccessoryType": "none"
        },
        {
            "itemId": "1636E37B6DCC5F5CC619A39",
            "itemType": "BT_menuItem",
            "loadScreenWithItemId": "2D795586BA0B0BF3EF1A08E",
            "titleText": "Call Duane!",
            "descriptionText": "Who ya gonna call!?!",
            "iconName": "phone.png",
            "rowAccessoryType": "none"
        },
        {
            "itemId": "5B03EE083825F8968CD40EA",
            "itemType": "BT_menuItem",
            "loadScreenWithItemId": "47420E823590DE384D3442B",
            "titleText": "Email Duane!",
            "descriptionText": "Guaranteed Communications!",
            "iconName": "mail_envelope.png",
            "rowAccessoryType": "none"
        },
        {
            "itemId": "202DF8C9C2A4E6E8580E5C0",
            "itemType": "BT_menuItem",
            "loadScreenWithItemId": "83D64D07838E32A1C30E802",
            "titleText": "Text Duane!",
            "descriptionText": "Immediate Response!",
            "iconName": "pointofinterest.png",
            "rowAccessoryType": "none"
        }
    ]
}
```


This is a typical menu from the control panel. It uses childItems that were manually input at the control panel.

The Same Menu using an external childItem file:

This is an example of an external childItem file. Identical to the one above, but the childItems are held in a separate file, which is accessed by either a 'localFileName' JSON key/value pair, or by a 'dataURL' key/value pair, depending if the file is bundled, or accessed via internet.

The fact that you can separate your childItems from the control panel provides a great way to automate your app, and reduce your config filesize.

To know where we're going, we need to know where we came from.

Your typical BT_config.txt file holds all of the JSON data to configure your app. It contains json for the App, Tabs, Menus, Themes and Screens. We're going to focus on the 'Screens' aspect of the BT config file. Along with your 'typical' BT config file are many JSON Screen 'Objects'. They describe Menu Screens, Splash Screens, Map Screens, every kind of Screen you're going to use in your app. When you 'add' a screen in your BT control panel, JSON is created for that screen to be included in your BT_Config.txt file. And it gets that information from your control panel for that Screen Plugin. The workflow usually goes like this:

You 'Add' a new screen. Let's say it's a menu screen. You name it 'Home'.
You 'Add' a new screen. Let's say it's a Map Screen. You name it 'Location 1'.

You 'Add' a new screen. Let's say it's an htmlDoc Screen to display an html file of information about Location 1. You name it 'Location 1 Data'.

You now go back to the Menu, and 'Link' a menu choice (which will be a menu childItem) by ItemId to the 'Location 1' map screen.
You now go back to the Map, and 'Link' a tap choice (which will be a map location childItem) by ItemId to the 'Location 1 Data' display Screen.

And you do this over and over and over again for each of your 20 locations, creating Map Screens, Data Screens, and linking them to each other. That's a lot of work.

A Quick look at what is...

Sample of typical 'BT_Config.txt' file JSON

```
{
    "itemId": "B2B33C9909CE31E7C6FAD9E",
    "itemType": "WB_screen_menuImage",
    "itemNickname": "tools",
    "navBarTitleText": "Tools",
    "navBarStyle": "solid",
    "imageURL": "http://                                    s/rotateimage.php",
    "listRowBackgroundColor": "#13407a",
    "listTitleFontColor": "#b9780f",
    "listDescriptionFontColor": "#b9460f",
    "listRowSeparatorColor": "#FFFFFF",
    "listStyle": "plain",
    "backgroundColor": "#13407a",
    "includeAds": "0",
    "childItems": [
        {
            "itemId": "1165C37DA4A34DD576EA8A6",
            "itemType": "BT_menuItem",
            "loadScreenWithItemId": "060F81DB28EABAF81CF229C",
            "titleText": "Notepad!",
            "descriptionText": "Take notes of your listings!",
            "iconName": "notepad.png",
            "rowAccessoryType": "none"
        },
        {
            "itemId": "43592B602126C941690C679",
            "itemType": "BT_menuItem",
            ....
            .... (And so on and so forth)
            ....
        }
    ]
},
{
    "itemId": "060F81DB28EABAF81CF229C",
    "itemType": "Notepad_feature",
    "itemNickname": "notes",
    "navBarTitleText": "Listing Notes...",
    "navBarStyle": "solid",
    "backgroundColor": "#13407a",
    "backgroundImageURLSmallDevice": "notepadmsg.png",
    "backgroundImageURLLargeDevice": "notepadmsg.png",
    "backgroundImageScale": "bottom"
},
.....
.....
.....
```

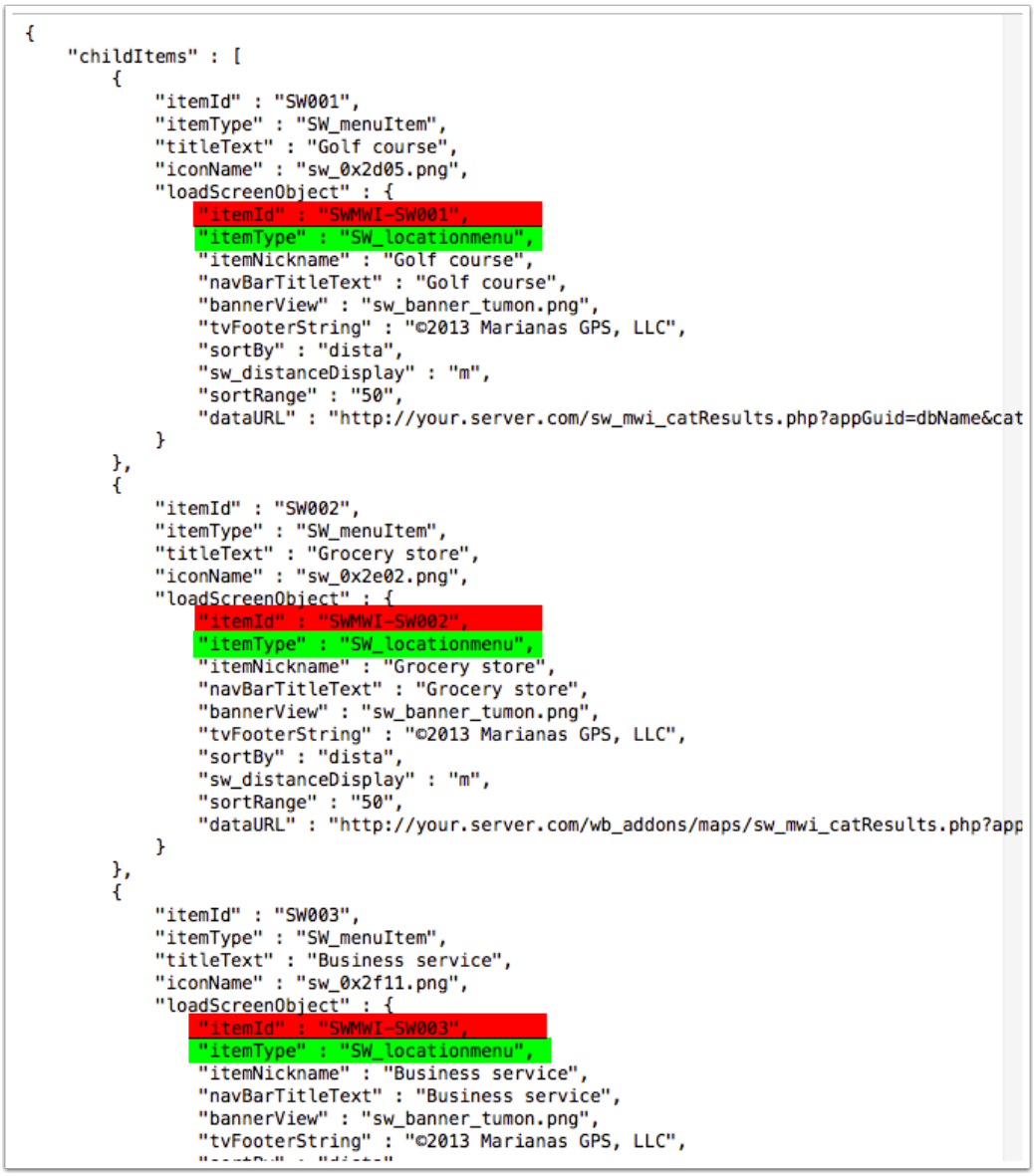
This is a common view of the BT_Config.txt file. You could have a menu screen. The menu childItem lists basic information about the target screen, specifically the 'ItemId' of the target screen, to be used with the 'loadScreenWithItemId' method. The menu childItem entry has just enough information to tell the app 'where' to find the appropriate screen, by listing the ItemId. When the user taps on the menu selection, the app hands off the target screen ItemId to the loadScreenWithItemId method and loads the target screen with the configuration information provided by the 'Screen Json'.

... and what could be.

```
{
    "itemId": "B2B33C9909CE31E7C6FAD9E",
    "itemType": "WB_screen_menuImage",
    "itemNickname": "tools",
    "navBarTitleText": "Tools",
    "navBarStyle": "solid",
    "imageURL": "http://a                                    teimage.php",
    "listRowBackgroundColor": "#13407a",
    "listTitleFontColor": "#b9780f",
    "listDescriptionFontColor": "#b9460f",
    "listRowSeparatorColor": "#FFFFFF",
    "listStyle": "plain",
    "backgroundColor": "#13407a",
    "includeAds": "0",
    "childItems": [
        {
            "itemId": "1165C37DA4A34DD576EA8A6",
            "itemType": "BT_menuItem",
            "titleText": "Notepad!",
            "descriptionText": "Take notes of your listings!",
            "iconName": "notepad.png",
            "rowAccessoryType": "none"
            "loadScreenObject": {
                "itemId": "060F81DB28EABAF81CF229C",
                "itemType": "Notepad_feature",
                "itemNickname": "notes",
                "navBarTitleText": "Listing Notes...",
                "navBarStyle": "solid",
                "backgroundColor": "#13407a",
                "backgroundImageURLSmallDevice": "notepadmsg.png",
                "backgroundImageURLLargeDevice": "notepadmsg.png",
                "backgroundImageScale": "bottom"
            }
        },
        {
            "itemId": "43592B602126C941690C679",
            "itemType": "BT_menuItem",
            ....
            .... (And so on and so forth)
            ....
        }
    ]
},
.....
.....
.....
```

This same situation with loadScreenObject, would have the needed information on the second screen in the loadScreenObject entry of the childItem. This illustrates how users can create second screen information using loadScreebObject.

If the 'conventional' method uses an itemId to trigger the next screen, what does loadScreenObject use?



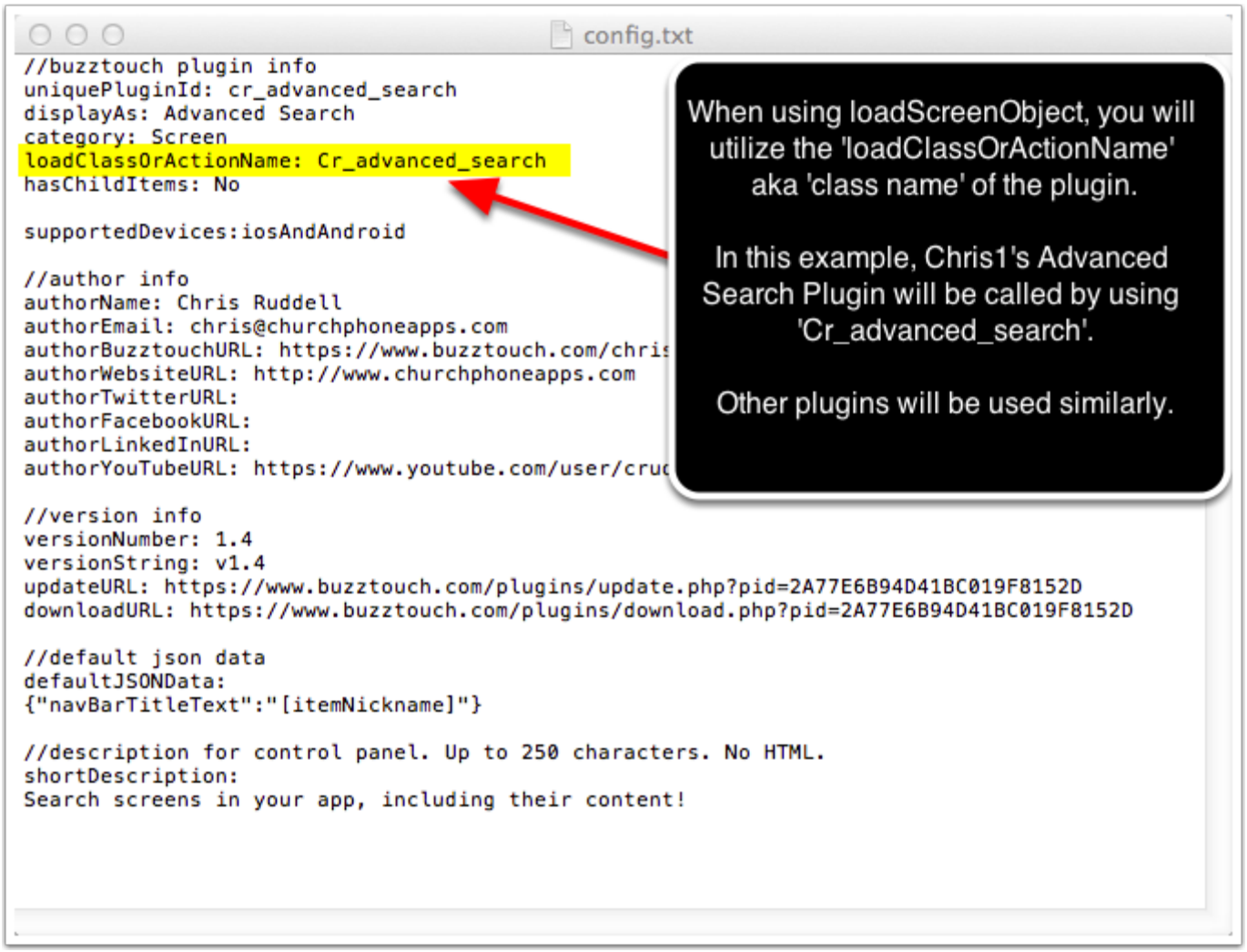
```
{
    "childItems" : [
        {
            "itemId" : "SW001",
            "itemType" : "SW_menuItem",
            "titleText" : "Golf course",
            "iconName" : "sw_0x2d05.png",
            "loadScreenObject" : {
                "itemId" : "SWMWI-SW001",
                "itemType" : "SW_locationmenu",
                "itemNickname" : "Golf course",
                "navBarTitleText" : "Golf course",
                "bannerView" : "sw_banner_tumon.png",
                "tvFooterString" : "©2013 Marianas GPS, LLC",
                "sortBy" : "dista",
                "sw_distanceDisplay" : "m",
                "sortRange" : "50",
                "dataURL" : "http://your.server.com/sw_mwi_catResults.php?appGuid=dbName&cat
            }
        },
        {
            "itemId" : "SW002",
            "itemType" : "SW_menuItem",
            "titleText" : "Grocery store",
            "iconName" : "sw_0x2e02.png",
            "loadScreenObject" : {
                "itemId" : "SWMWI-SW002",
                "itemType" : "SW_locationmenu",
                "itemNickname" : "Grocery store",
                "navBarTitleText" : "Grocery store",
                "bannerView" : "sw_banner_tumon.png",
                "tvFooterString" : "©2013 Marianas GPS, LLC",
                "sortBy" : "dista",
                "sw_distanceDisplay" : "m",
                "sortRange" : "50",
                "dataURL" : "http://your.server.com/wb_addons/maps/sw_mwi_catResults.php?app
            }
        },
        {
            "itemId" : "SW003",
            "itemType" : "SW_menuItem",
            "titleText" : "Business service",
            "iconName" : "sw_0x2f11.png",
            "loadScreenObject" : {
                "itemId" : "SWMWI-SW003",
                "itemType" : "SW_locationmenu",
                "itemNickname" : "Business service",
                "navBarTitleText" : "Business service",
                "bannerView" : "sw_banner_tumon.png",
                "tvFooterString" : "©2013 Marianas GPS, LLC",
```

Hopefully by now you have a basic understanding of how the basic BT app works, using 'loadScreenWithItemId'. Every time you 'add' a screen, that screen is created. Along with screen creation, an 'ItemId' is created to identify *that* particular screen.

However, if you create screens dynamically, there is a good chance you may not know 'what' the ItemId is going to be before it is created. If this is true, how do we identify the screen? Answer: you don't worry about the itemId. You're going to create the screen by the itemType (file class name).

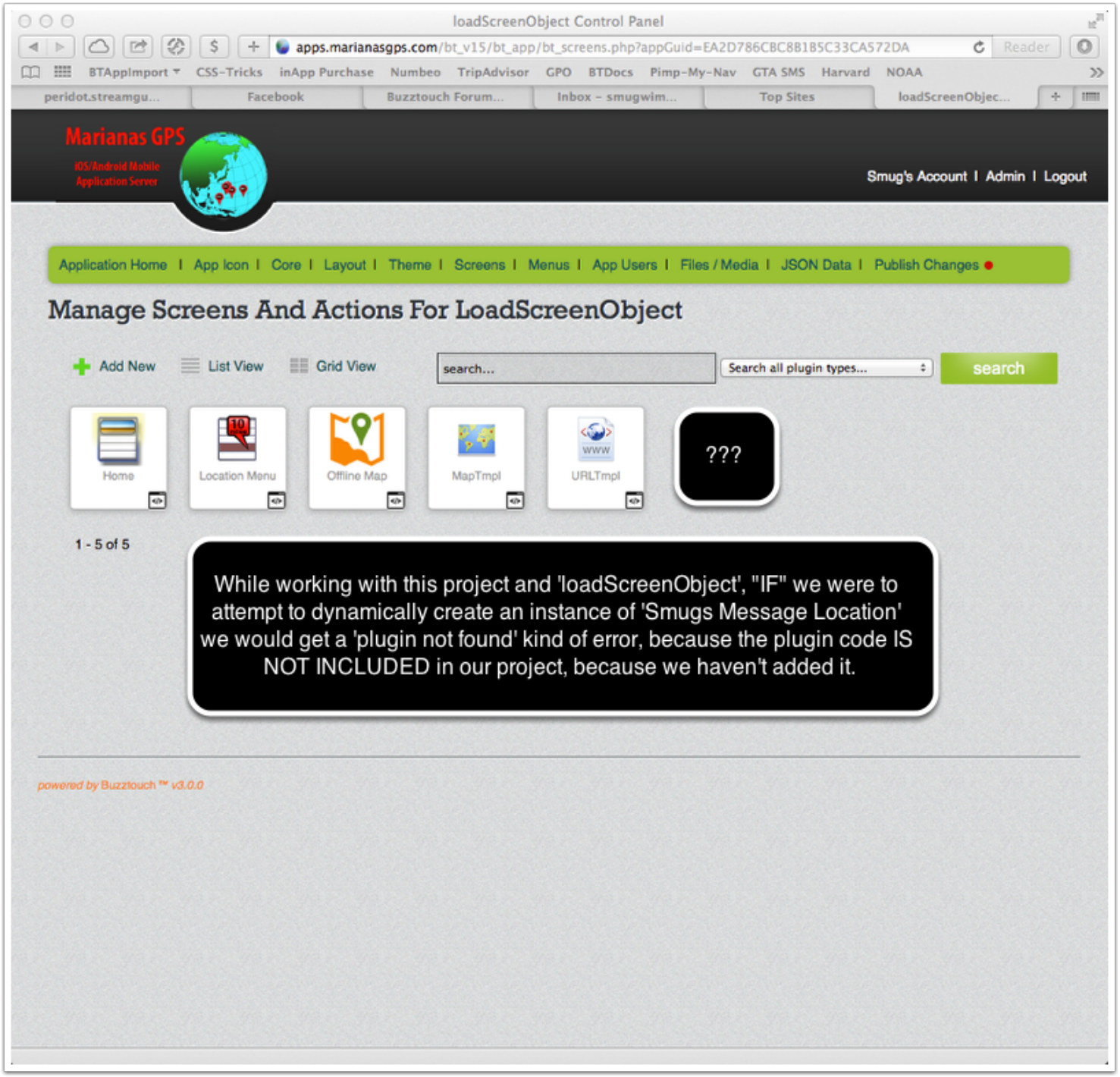
AS LONG AS YOU HAVE THE CORRECT PLUGIN IN YOUR BT PROJECT, LOADSCREENOBJECT WILL CREATE YOUR SCREEN. You *have* to include the code for your plugin in your file. All we're doing is changing the 'configuration' of your app, however it will not create code. When using the BT Control panel, it should automatically 'add' these screens to your download package. If you're using loadScreenObject, you'll have to manually add at least 'one' screen of the correct type.

identify a Screen using it's 'loadClassOrActionName'

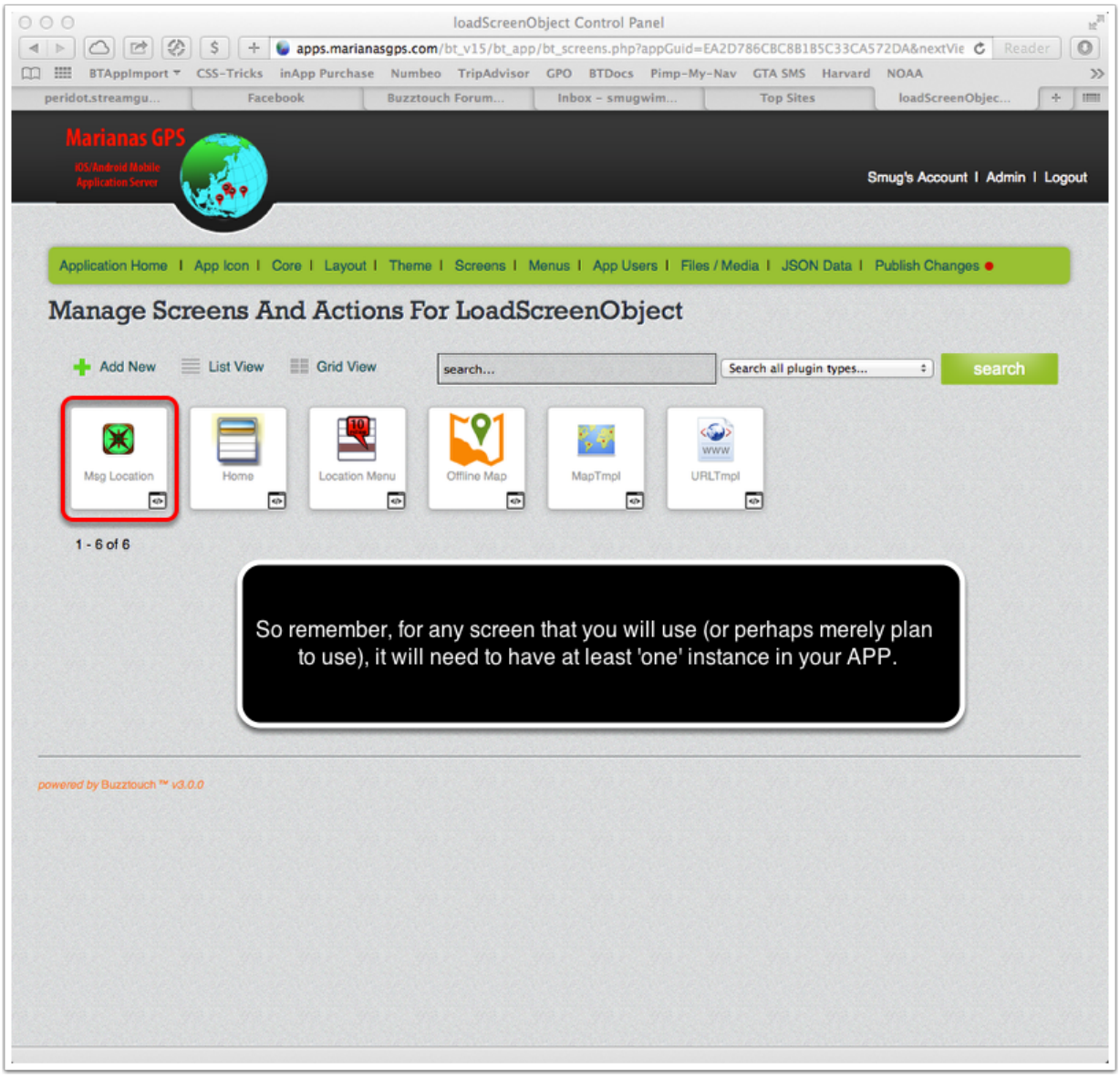


in using loadScreenObject, the importance of the itemId is minimized. As long as it is a unique value, we're happy. We're far more concerned with the 'loadClassOrActionName' of the plugins 'config.txt' file, which is also the 'Class' name for the plugin.

To use or not to use...

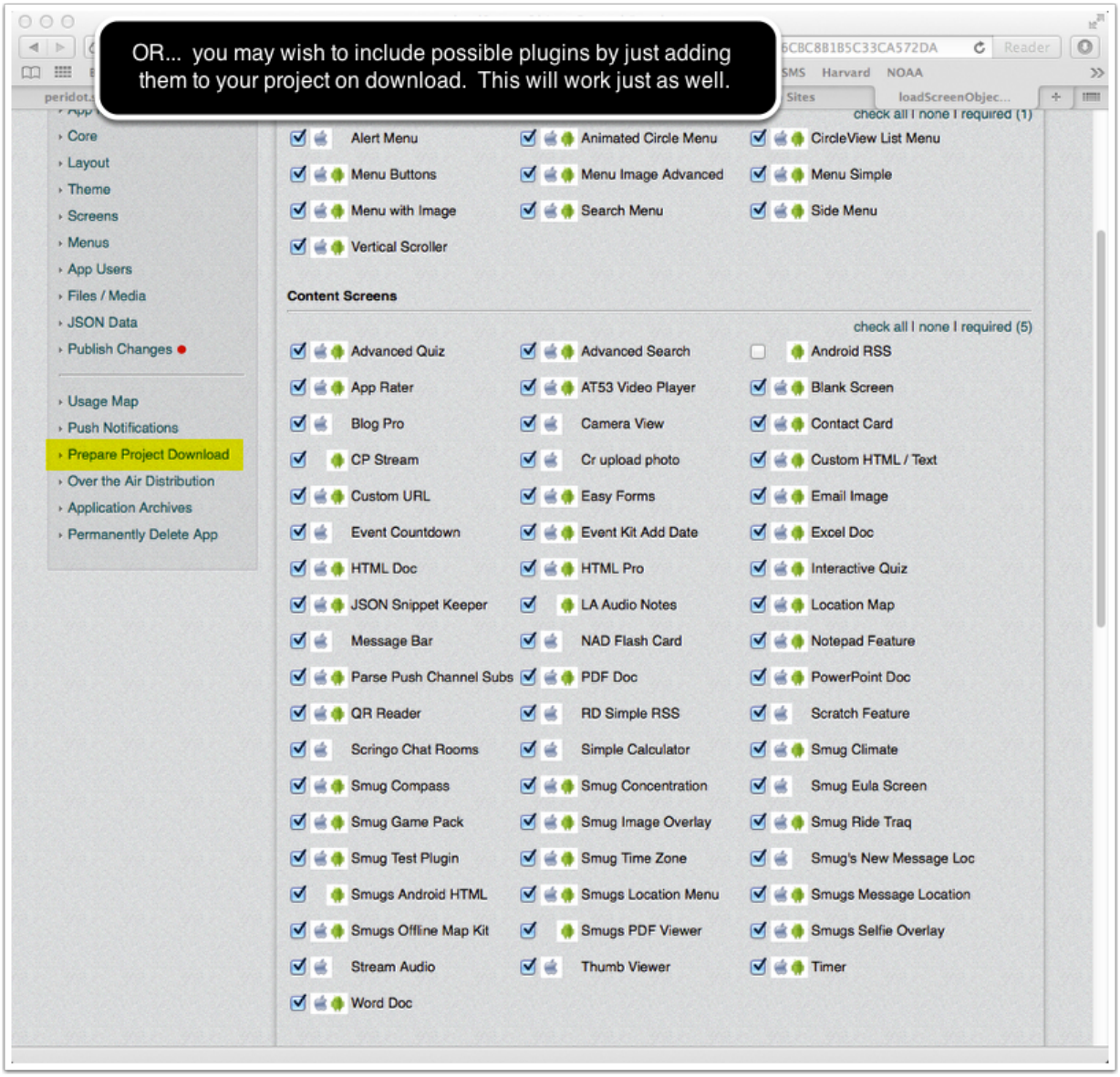


make sure you have code with your project for the dynamic screen you create.



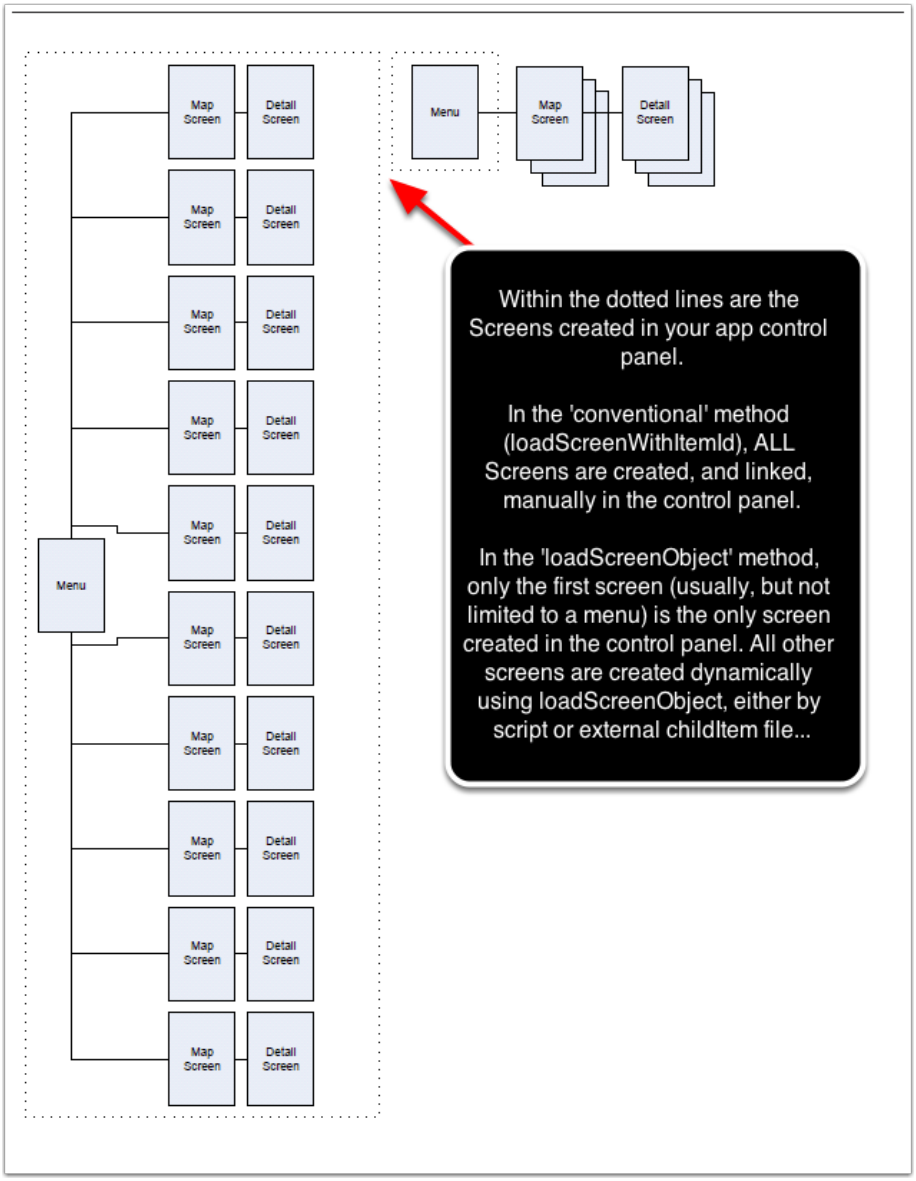
Remember, loadScreenObject is merely a 'way' to configure your BT app. But like any other BT app, you HAVE to have the plugin CODE in your project for it to work.

select on download is a viable option...



You can choose to add screens by the control panel, or you can choose them on download. Either way is fine, as long as the code is included in your project!

Semi visual representation of LSWII and LSO Differences



This is sort of a visual representation of screens in a typical project, and screens in a 'loadScreenObject' project. As you can see, in the conventional method, all screens are created, linked, and configured by hand.

In the loadScreenObject representation, you only need to add 'one' core screen (usually a menu, but not always) and the other screens are created dynamically using LSO.