fpxEngine

Table of contents

Lets Get Started

Lets Get Started

Let's Get Started

The fpxEngine is designed for all levels of pinball developers, from the advanced coder to people who have never scripted before. A powerful and full featured pinball engine, fpx was built around the concept of presets and using each of the presets with just one line of code. The entire structure of fpx was carefully written and arranged to make ease of use as simple as possible

This is a fully functional script engine, with a special tutorial included, structured and aimed at beginners (though intermediate scripters would like this as well) It is a very different way of doing things, a lot of things were not added (like polish to the script, lack of details in the example table) but that was to not overwhelm beginners with too much high end concepts. Everything is heavily commented, even the hit code is structured to add and expand scripting methods and concepts as the user goes through it.

The engine includes 4 player scoring, 9 digit A.N.D., with routines that handle everything from Table start, to the match and light Attract modes. The table design has examples and fully functioning code, including bonus and bonus countdown routines, simple background sound system, and examples of extra ball, Jackpot among others.

Beginners Guide

Because the main engine contains code that only advanced scripters should change, the fpxBeginners Template is designed to give absolute beginners the ability to customize their tables in as easy a manner as possible without having to dig through very confusing and easy breakable advanced code.
Learn all the basics of the fpxEngine, with the step by step tutorial that shows and explains everything as you go along. There is not a easier template engine out there, you don't even need previous coding experience!

Link

fpxEngine System Features

System Features details some of the special features and settings that you and a player can use to modify the table, like the menu system built into every fpxEngine table to adjust Balls per game, as well as unique settings to adjust the overall table lighting and shadows.

Link

fpxEngine Presets

The magic behind the fpxEngine. No more coding complex routines, no more months of learning how to code, no more months of debugging and pulling your hair out trying to figure things out. fpx includes a powerful suite of preset code that's just 1 or 2 lines of code to use powerful and common pinball features like Extra Balls, Jackpots, and many others. Soon, fpx will add even more with the new Vault system, to give you even more advanced features such as Advance Value, drop target routines, and multiball, complete with objects you can copy and paste right into your table, and it works!

Link

AddScoringEvent

One of the most powerful and simplest to use features with fpxEngine are the built in scoring features, such as a "Extra Ball" or adding a bonus Multiplier. In other engines, these are very tough to code and debug, but fpxEngine contains a suite of special features, all written in such a way that you can use them anywhere you want, and all with a simple copy and paste of one line! This section contains all the information you would need, and has a full in-depth explination of what the code does, and even the code used for advanced coders who wish to modify the code. There are already multiple special features, with more being planned to be added.

Link

The Vault

Don't have any coding or design experience? No problem, you are using the fpxEngine, you can have a fully designed and coded table in a matter of minutes. The Vault is a suite of small prebuilt templates that you can copy and paste directly into your table, and when you press play, It works! Unlike other engines, you are not limited to one design, or with the most basic coding, the vault items have full lighting and music effects, the ability to have "pinsettings", and many more features built in, all in a highly professional presentation that will wow you and the people who play it. Fast, simple, easy, and the flexibility to produce a design the way you want it, not forced like the other templates do.

Link

Better Arcade Mode (BAM)

Without Better Arcade Mode (or BAM for short) Future Pinball is pretty useless. fpx contains basic support for BAM physics, in a incredibly easy manner. You can adjust the physics from the old EM era to the modern era with just one change of a number, and even set the bounce of the flipper just as fast. As time goes on, the fpxEngine will include even more preset BAM code to help make your game even better.

Link

About fpx

Lets Get Started »

About

About The fpxEngine

The fpxEngine is a direct replacement for the new-table template for Future Pinball, with bug fixes, and missing code added, as well as additional code. This uses all stock objects and textures from Future Pinball (we call it FP), with the exception of the custom Flippers models

This is intended for the absolute beginner, and was developed to make it as easy as possible for the beginner to script his own games. This engine template was made as simple as possible, so there are some things that lack polish or additional coding. This is also a learning template, as it describes each line of code, and was written and structured to teach basic scripting concepts to people who may never have scripted before.

In fact, there is a lot less scripting than any other template or table, nearly all of it has already been done and instead just has presets where all you have to do is just change a number or type in different text. The actual scoring code for things like scoring points from the ball hitting a target or a bumper (the big fat things you see in all actual pinball games) are all ready coded in for you. Any code a beginner needs to use them can (usually, with the basic ones) be done by one line of code, that usually you can just "copy" them and then "paste" them. Even when the engine supports more and more advanced features and scoring, it will be done in a manner that is as simple to use as possible.

We all hate scripting, it is not easy, even the best of us makes mistakes and it can take hours to find and fix a mistake.
The best way for you to look at this is in writing all this code for you, I also made all the mistakes for you, and I corrected those (I hope)

So open up the script, start from the top, work your way down till you get to the engine code and I tell you to stop. If you want to learn more about FP, the built in FP help file is a very good resource to learn more from. Though FP is far more simple to use than the other editor, it still has lots of settings and options. The Future Pinball manual does a very good job of detailing those, so I recommend you give it a good look before you start this. Just click on help at the very top text menu in the FP editor and then click "open Manual"

About Better Arcade Mode (BAM)

FP has a secondary program (or a plugin as it is usually called) called Better Arcade Mode (or BAM for short)

BAM greatly extends the capabilities of FP, there's lots of new features, and it's a real pain to use because it is very very complex stuff. Instead, the fpxEngine supports the basic BAM code for physics and is set up for you in as simplest manner as possible. The biggest thing with BAM is it finally fixes the play ability of FP which is all added here. I have also included some other helpful features of BAM, and then rewrote some things to make it as easy as possible.

The fpxEngine is all I can handle at the moment, so I wont be able to talk much about BAM here. If you have a problem or want to learn more about BAM, you should go to www.gopinball.com.By the way, say hi to me there as well. :)

Build History

Features

The FPx template is designed for those who want to get their feet wet with scripting, and yet has all the power and features for Intermediate coders to use.

Here's a list of features:

Build 10

' 4 players support
' Bonus count up to 159, bonus multiplier up to 15x
' Replaced tilt system with a custom plumb bob tilt setup
' Added Variables for easy settings. balls per game, bonus, replay scoring etc
' 2 ball multiball
' Player memory system that requires typing in one line of code per light
' ball save, extra ball, special and jackpot support
' Complete routines included for all aspects of arcade tables, from power up, light Attract mode to a match feature
' Initialization, Light Attract, Match routines all ready included
' easy control of lights for Light Attract and game play, no typing required
' Pause key plays tune
' 9 element x4 translight display, HUD display x2 A.N.D. (Alpha Numeric Display) to support up to Williams System 11
' HUD display can be hidden or shown using the default HUD key set in the editor (Usually the tithe key, just below the esc key)
' Pre-set unified Bally type scoring
' Debug system variable for code testing in place. press F9 key at editor. Also writes to text file during debug mode to track code
' Special HUD displays for debug mode, shows bonus etc
' Multiple sound system capability with background music support
' Solid State and A.N.D. support HUD display
' Preset feature scoring Extra ball/Special etc with just one line of code
' Easy Beginners Fail safe system. All access needed for easy modification of the engine in ONE place without needing to touch the engine code
' You can add/remove/replace lights, sounds, change the display, adjust timers etc through

this one setup
' Code is heavily remarked, and teaches you new methods as you work your way down the hit code
' Engine is fully automated as possible, so no need to touch the engine code, no rocket science degree needed.

Build 11

- fpxEngine is now so simple even a person with no coding experience can use it (scriptwise that is)
- fpxEngine has been completely restructured and rewritten. It is now the easiest template engine to use since Pinball Construction Kit
- fpxEngine now has a manual, that describes a lot more than you would think
- Preset section added. all accessed with one line of code to multiple scoring routines. Just point and shoot
- Bug fixes etc.
- Light Attract routine expanded, now displays high score list, table name. This is now fully automated, so no need to add light bulb interval/on/off code
- BAM support added with a simple system for the user to select among multiple xml physics files and a up to 10 different flipper bounce settings.
- Select-able Debug mode (press f9 key in editor) now shows general table info including physics package or engine debug information
- New Display and Light Seq routines added
- New Control routine added, which adds multiple music set support, with the ability to customize all displays/lightSeq/music per music set. Support added for Williams System 7 and early 1979 music sets.
- Users should be able to play any music set/company style with any fpx table
- All display and lightseq code is also usable in code as well (e.g. AddDisplay "BlinkMessageFast":AddLightFX "BlinkFast")
- HUD displays expanded from 2 to 4 displays, to match the translight displays. This is for possible future support of Williams/Bally system11
' and Data East tables
- all music/sound code has been (or will be by the time the next build is released) moved to the main control routine (AddMusicSet) so we can have multiple styles without the user having to do anything
- Multiple background sound support (up to 29) using increasing frequency/pitch common with Solid State games.
- There are about 15 display routines now coded, with 6 new routines added. (More will come)
- AddEvent has been stripped of any code completely. Now it is just for typing in messages for the display to use, and your custom code hooks like what to do with a new ball etc.

Build 1.2

- Fixed: Bam "fix" breaking nvHighscore. From now on, you need to set the highscore and names in the script. You will find how to do that right here
- Fixed: textdebug now shows proper information in debug mode. This appears to happen with slower computers, or the debug mode just isn't fast enough to write too much text
- Removed music/light/display routines not used.
- Drain code fixed to allow main music feature to finish playing before bonus count started. (TimerDrain)
- Added some mechanical sounds
- constAddDebug has different settings based on number
    1 = shows engine subroutines and addEngineEvent

2 = shows AddMusicSet subroutines and the music case it is playing. This also now shows AddLightFX and AddDisplay subroutine calls
- Renamed some things, deleted unused routines. LightAttract scroll effect slowed, replaced table name showing with replay goal instead
- Added Mystery Feature. Player can set minimum and maximum values for a random score
- AddScoringEvent sections included in the manual with copies of the code used by the engine and complete description for coders
- AddDisplay new additions: "Radar"
- HUD key added (show HUD/ Hide HUD) Because BAM uses the default HUD key for it's own menu, a second HUD key is added set to the "H" key. nvR and nvS support is added, the table will now save menu settings.
- Table Menu system added. Use "M" key and then flipper buttons to change options
You have to exit the table and restart the table again for the changes to take place.
- Save/Load system added. This saves number of Balls per Game, scroll/no scroll, high scores plus high score names, plunger type (enter key or arrow keys) and 3 difficulty levels
- Manual has been updated, and almost completely rewritten in spots. quite a few new pages have been added.

BAM Support

- Support added for flipper and ball shadows
- Custom ball, including blinking ball based off example by ravacade (turned off by default)
- Some small improvements made, and some additional support. None of the fancy stuff nor is it up to standard with the newer tables.

Textures now set color as well as graphics, so all objects set to a light white/grey color for easy modification. A new manual page added just for the template objects, and to explain all the new features. (fpx Table Build)
- Beginning the new overhauled Debug and script generator. Press F9 in editor, a option list appears. Numberpad 1 generates a copy of the User section code to fpDebugTextLog. This is script ready. Numberpad 5 shows BAM settings used by fpx
constAddDebug = 0 shows variable info at the start and at the loss of a new ball.
- First Vault item - fpxAdvanceScore- About halfway done, this is a test of the main master code and template for the first Vault Item and the basic structure of the code. The code will be folded into the engine when completed, this is placed in the Hit Section for now for review and testing. This code features 2 pin settings to help customize your game.
fpxAdvanceScore advances the score by 10k and awards extra ball and special when lit, but is fully adjustable to score any fpx AddScoringEvent calls.
There is a alternate code in the manual page to score features instead based on the features presets in AddScoringEvent
The manual page explains this vault item, and has a complete description of the code, lists of the variables and objects, and also has alternate code you can include for additional scoring features. . This also includes full information and the complete code to use in non fpx tables.
- AddScoringEvent - Mystery and Add Multiplier will now override music already playing. A new AddScoring Event "SpecialIsLit" has been added to the script and is demoed in the Vault fpxAdvanceScore as well.
- PhysicsXML=2 replaced to original Jungle Girl code, Just wasn't Bally with others xml settings. Plays a bit slower. An old experiment also included, code to prevent the ball shooting out like a cannon at the very tip. Still too fast, but flipper speed animation gets too slow. Only way to improve this is with a possible future new script feature added by BAM.
- Manual updated and some pages changed or added to reflect new features.

Build 1.4
- Manual rewritten and up to date

Build 1.41
- The fpxEngine overhaul has begun, with a rewrite and overhaul of the code to make things even easier for Beginners and non-coders to be able to use the engine and design their own tables from scratch
- shivaFlippers (version 3.1) added.
- Plastics in the Vault are now lighted objects, with special code added for BAM support and blinking effects.
- Menu key changed to the Special 2 key
- removed music sets, only Bally81 set in. This was to save time and get a master named music set in. The removed music sets will be added back in and renamed in the future.
- Messages removed from subroutines and placed in AddMusicSet
- New Vault items (basic) added. These are fully coded, but are simplified scoring. There are 7 vault items added for this version.
  - StandupTargets
  - Drop Targets
  - Kickers
  - triggers
  - Plastics and spare parts
- New AddScoringEvents
  - (AddScoringEvent "25kAward") 25kAward
  - (AddScoringEvent "AddAlternatingLanes") Alternating Lanes (for both Inlane and Outlanes) with pinsettings in the User Define Section
  - (AddScoringEvent "OutLaneSpecial") Special code to handle a Outlane special, that also turns off the outlane lights
  - (AddScoringEvent "SpecialIsLit") a prompt informing the player that a special has been lit
- Ball Shadows added with settings (On/Off) in the User Define Section
- Playfield lighting added, with custom lens and bulb lighting code. With 3 settings in the User Define Section
- Dynamic Shadows added. With settings (On/Off) in the User Define Section
- Specail Bam Lighting code has been added for all lights, bulbs, and plastics, using Brightness, GlowRadius, and GlowBrightness. 1.14 only has these tested for the playfield lighting settings at it's brightest. Medium and Dark settings are there, but will be adjusted in the next versions.
- Manual rewritten but more rewrites are needed. A new section in the manual has been added for the Vault.

What you need first

What you need first

What You Need First

Guide to Future Pinball (by GeorgeH) http://gopinball.com/forum/viewtopic.php?f=84&t=6054

Playing the table

Playing the table

Lets Get Started »

Playing The Table

By now, you should have a good idea about the FP editor, so now it's time to start working with the fpxTemplate.And of course you are going to want to play it to test it out. Open up the template from the location you stored it in. As you can see, it's not exactly a lot there, it pretty much looks like the basic "new_table" that fp has, but it's not the looks that matter, it's what's under the hood that counts.

Still, press play and have a go at what's there. You will notice some pretty high end features, like the display text and the lights, that just isn't there in the new_table template. In fact, the fpxEngine is the third most powerful engine released, only shivaEngine2 (for old vp) and propst (for FP) are more powerful, but then, fpx also happens to be the easiest to use not matter what template you look at, or what skill level as a coder you are. In fact, the entire object of the fpxTemplate is to make it as easy to use for people who don't even know how to code, and there really isn't anything like it (as far as I know at least)

Have a play or 2 and then come back here. If you look in the actual editor, there are a group of triggers (the funny looking star shaped objects) to the left of the table. (You may have to increase the magnification, as FP likes to just show the table dimensions, and not what is outside the table in the editor)

You will notice a menu system comes up. This is where you can change balls per game, skill levels, set to use the arrow keys before you hit enter etc . This is a built in feature of the engine, more options will be added soon.

The Beginners Guide

fpx comes with a bunch of examples and code already done for you, and these are the ones you can play with. You can find a complete walkthrough in the Beginners Guide in this manual. These examples uses the trigger objects, and have a different feature assigned to each trigger (one adds a extra ball, one scores the jackpot etc) so just select one, look at the name of the trigger at the top left OPTIONS bar, hold the left mouse button down, and then just drag it onto the table, press play and it will work. Once you are done, drag it back to where it was in the editor, and drag another in it's place.

to adjust the plunger
These are the preset features built right into fpx. There's not a lot (at the moment) but that will change in time. When you do open the script editor, you will be able to change the text that is displayed when a ball rolls over the trigger (you don't have to if you don't want to) or change a number (you don't have to do that either, only if you want to) and maybe just copy and paste the code if you want to duplicate a feature and have 2 of the same feature in your game. These triggers are coded in the "_Hit" section, I go through each one of them in the beginners guide, as you move down the script and tell you what they do, but you actually don't have to change them.

The Vault

Of all the unique features that fpxEngine has, nothing out there is comparable to the Vault system. The vault is a collection of premade templates complete with code that you can copy and paste right into your design (or use multiple vault items to make a complete design) and within minutes, have a fully working, complete with code, table. In fact, there is no engine, template system, or sample tables out there there have the level of sophistication and polish that the vault items have. Most templates are very basic, knock some targets down, get a basic score, and then pop them back up. The target vault items in fpx do far more, they have complex lighting and flashing routines, multiple levels of scoring, and even has built in player memory that will carry over to the players next ball. And it's all adjustable. Powerful routines built in such as advancing score, jackpots, and alternating lanes, with multiball and lane change coming among others. All with a simple copy and paste that should take you no more than a minute to add.

The vault is only just beginning, already there are vault items for several Future Pinball table objects. Soon, new vault items will be added that will be based on the actual arcade pinball tables, and you can mix and match vault items however you see fit. Consider it a simple jigsaw puzzle, but for pinball table design.

There will be more examples and code added, but ultimately, that's all you would really have to do. And even if you are unsure, a lot of help is written right in the script, and usually a far more long winded version will be here in this manual.

Credits

Credits
Lets Get Started »

Credits

fpx Engine by shiva (C) 2019
p.d.Sanderson

fpx Engine Template by p.d.sanderson, is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

Beginner's Guide

Beginner's Guide

Beginner's Guide

Because the main engine contains code that only advanced scripters should change, the FPX Template is designed to give absolute beginners the ability to customize their tables in as easy a manner as possible without having to dig through very confusing and easy breakable advanced code
Instead, this template is split into 2 main sections, the user define section, and the main

engine code.

The **USER DEFINE** section was written to give access to all the main things you would want to change, like sounds/lights/ display messages, without having to search through thousands of lines of code or accidentally modifying engine code that could break the script.

The **MAIN ENGINE** code you can leave alone, all of the code you may even want to modify is pointed at the top of this script, and done as simple and as easy to understand way as possible. These are "Hooks" within the main code, and give you access without causing errors (even placing a line of code the wrong order can cause errors, you don't have to worry about it now, just keep your code here within this subroutine)

With everything in one place, and set and called properly from within the engine, all you have to do is add your unique table code (Or just leave it the way it is) All Display/LightSync/ music is run with presets, so you don't even have to worry about that.

From the very startup to the end, all the grunt work and engine stuff is done for you, just add anything special you want to it. No matter what, you are still going to have to learn some scripting, but the Beginners Guide even explains that for you.

Most of the code is the same, and any changes here you can make are usually text messages, music, and timer intervals to set the amount of total time you want each case to play before it automatically switches off. It really is the easy way to do things.

YOU DON'T HAVE TO CHANGE ANY OF THIS IF YOU DON'T WANT TO!

These are just presets, if you like it the way it is, leave it alone. If you do want to experiment then this is a great place to do so, as since this is outside the main engine code, you have a safe place to learn without causing errors to the engine.
(And if you do, let me know)

The Basic Structure

Before you open the script, you should understand the basic structure of the script, or for you beginners out there, the top 10 percent of the script actually. If you have already opened the script for fpx, and scrolled down, you will see just how big it is (and that's nothing, if fully developed, the fpxtemplate could be in the 10's of thousands of lines. Aren't you glad I'm doing it for you?) but really, it's only the top part of the script you ever need to do. Everything else you can just leave alone, no need to have to wander around in there, it took me months to write that, and even more time to get it to work properly, and it's very very easy to mess it up (I would also know that one)

Basically, fpx is divided into 2 main parts, the top part that is intended to be modified or used by the table developer (that would be you) and the second part, which is the engine code and should not be touched by anyone, unless you really really know what you are doing. There will be a engine section in the manual down the road, but the entire concept of fpx is to make it so you never have to touch the engine code.

So lets talk about the top, where you will actually work within, called the user input section.

The fpx user define is divided into sections, 4 in fact.

User Input Section

These are the very basic settings. They allow you to change the common pinball table settings, like how many balls per game you want, how much you want a jackpot to score, and others. For those of you that own actual pinball tables, consider this is very simple but very flexible pin settings.
Link

The Hit Section

This is the actual code for the objects of a pinball game (triggers, bumpers flippers etc) You have to tell fp what to do when a ball strikes one of these objects (like add points to the players score, or kick the ball out, play the music etc)
Everything is actually done for you, but you should review it if you want to add your own custom code down the road. It also starts teaching you the basic principles of code (but as little as possible)
Link

Engine Event Section

It may have the word "Engine" in it, but all this section does is allow you to change some things in the engine, mostly just a different text message you want displayed instead of the default. People who do code will use this section to add their own custom code instead of directly in the engine itself.

These Events are "hooks" in the engine scripts, to allow easy access to key areas without having to go through thousands of lines of script. Even elite coders place things in the wrong spot, so these are
a pretty fool proof way to add your own code without then spending hours debugging it if it throws up a error. You will see examples in there, more than likely just lights switching off that was turned on when
a new ball in play is started.
Link

fpxEngine Presets

The fpxEngine presets section is for those who wish to add their own custom routines based around the presets already included with fpx. It's pretty easy to figure out how I did it, and set it up, everything is the same way, and it's just basically "point and shoot" with the code. You will find the main controlling subroutine, and the other subroutines for displaying the text and score (with scrolling), the special effects for lights and bulbs, and others. There is even a small subroutine so you can easily add more flippers or redefine a key. There will be of course a very detailed section in this manual down the road.
Link

The rest is the dreaded Engine code, but if you payed attention, I already told you not to touch unless you are Randy Davis.

If you have never coded before

If you have never coded before
Beginner's Guide »

If You Have Never Coded Before

Scripting is hard. It's incredibility boring, takes forever to get to work properly, and even longer than forever to debug the code and test it out hundreds of times. Anyone that sniffs at you and tells you scripting is easy you should grab their pocket protector from their front pocket and beat them to death with it.

If you are brand new to scripting (or coding as it is also called) the first thing you are going to say is

"There is no way I can do this"

Well you can. Most of the writing (75% of it in fact) in this script is in fact me telling you how to do things. The entire concept of fpx is to take as much coding off your hands, instead, I have done all that for you, all you need to do is make some very simple changes to what I have done, but only if you want to. The engine is automated, it will work all by itself if that is what you want and decide to not touch it at all.

Script Code

As you may figure out right away once you open the script for the fpxTemplate, lines that start with a ' are called remarks, and every thing after them on the same line is colored green.
These are used to explain the code or comments within a script, as messages or the computer version of Sticky Notes if you look at it that way.

The program when it is running ignores remarks, they are not part of the code, but for our purposes, they are a great way to teach you and show you examples every step of the way (Just like I am doing this now)

If you remove the ' at the front of any green text, then that becomes part of the code to run the engine (and it will turn into different colors). We will have a example of that shortly, but here is a quick example of a remark...

```
code:  ' This is a remark, it is just instructions that you
       should read in the script
       ' Notice the ' in the front? If you delete it, it is
       considered a part of the script code.
```

So, easy enough right? Just read those remarks as you go down the script, as it will help you change and modify the code to fit your table. The fpx engine is very powerful, even at this build, but it's actually so simple to use that everyone can use it, even those that have very little coding experience, or who have never scripted before. There are even remarks on the same line as computer code, but remarks there has to be AFTER any code as any code after a remark will be ignored and not run.

No Really, is it that Easy?

Actually it is. First off, you will have to learn some coding (but down the road) but for the most part, all you will be doing is changing a number, or replacing some text between the quotes. Others are just one or two lines of code, and of course, I will not only explain it to you, but even provide the code so you can just copy and paste it right in.

You should be able to type it in yourself, unless you are a horrible typist like I am...

User Input Section

User Input Section

Beginner's Guide »

The User Input Section

So, done playing? Ready to go? Let's start...

Open the script editor (right side menu in FP, says script) and the script editor appears. This is a very primitive version of notepad, and contains the code for the fpxEngine. If you have
never scripted before, a quick review of the "If you have never coded before" section may be in order. It just explains the multiple green lines are in fact just instructions to help you along
the way.

Pay Attention to This when you start!!

With coding there has to be certain things that has to be included at the top of the script. These can not be changed, they must be included no matter what, and can't be moved somewhere else. THEY HAVE TO BE INCLUDED WITH EVERY VERSION AND TABLE THAT USES THE fpxTemplate. These are marked as noticeable as possible, just ignore them and scroll down till you see

```
code: ' *** END OF DO NOT CHANGE ***
        '
        '
      *****************************************************
      *****************
        ' **
```

```
                         **
  ' **                           User Define Section
                         **
  ' **
                         **
  '
*****************************************************
*****************
  ' Everything in one place, what a concept....
  ' Lets begin.
  ' *** START HERE ***
  ' OKAY, NOW YOU CAN CHANGE THINGS, TILL YOU GET TO
THE OPTIONAL SECTION
```

I guess it's pretty obvious where you start...

About the User Input Section

This is all the main settings so you can customize your table. Things like the amount of balls per game, etc. Think of this (in actual arcade pinball terms) as far more advanced "pin settings".
For the most part, all you need to change is the number at the end of the code (not these remarks, the multicolored part, or type in a name or text between the quotes (again in the multicolored part)

But this is also a computer program, and requires a bit more. I will explain everything as we go along, but if you are unsure about how to change a setting, then don't change it till someone
writes the manual that can go in far more detail.

Some settings require either a 0 or a 1 at the end. 0 means that feature is turned off, while 1 means that feature is turned off
Some features require a lot higher number as well. Usually that either means a time to set that afterwards runs another portion of the script, or scores the point total you want.

I will explain each one as we go along, but if you are still unsure, just skip to the next line, Everything will still work fine
Change the settings to how you want, the engine was designed to be as adjustable as possible.
Just work from top to bottom, till you get to the part where I tell you to stop, as there is some more essential code there you can not change. I've divided the user input section into little groups
to make it easier for you. I will write the default code out, and then tell you what it does and how to modify or change it.

The First Things to know

Each line is actual computer code. The only things you can change are text WITHIN the quotes (These things " ") Do not change anything else in that line, and make sure you do not
delete the quotes. Just change the text inside the 2 quotes. Don't add any more quotes either, or the script will have a fit, and hold it's breath and turn blue until you change it back

to the way it wants to. Sort of like a few people you might know, some things you just cant change.

As a example, the very first time you change the text. If you want to change "fpxTemplate" to "My TableName" then you would change this line:

```
code: Const  constTableName = "fpxTemplate"      ' Name of
      the Table. Make this unique
```

To This Line:

```
code: Const  constTableName = "MyTableName"      ' Name of
      the Author. Make this unique
```

Just the text within the quotes has been changed, nothing else. you will notice the remark at the end, you can do that after code, and remarks also help you keep track of the code.

How to change a number

Well in code at least. Lets use nvBallsPerGame as a example. This is part of the main Future Pinball code, it tells the script how many balls per game a player has. As you can see, it's set to 3. As with the text explanation above, just change the number, nothing else. So, if you want instead to have 5 balls instead, then change this line

```
code: nvBallsPerGame  = 3      ' Set Balls per game
      (overrides the Table info, which is very flaky and
      usually doesn't work anyway)
```

with this line:

```
code: nvBallsPerGame  = 5      ' Set Balls per game
      (overrides the Table info, which is very flaky and
      usually doesn't work anyway)
```

All you do is change the 3 to a 5, and now the fpxEngine will be a 5 ball per game table. Like above, don't change anything else, just the number!!!!

The lack of commas

Some lines have a bunch of numbers there. For example " Const fpxReplay1 = 320000 " which means 320,000 to us (it's the first score in points a player has to reach to earn a free game) You can change that to what you want to, say even 25000000, just don't add any commas as the script won't like it. We also use this to define the amount of time we want for a feature to be active (like the Ball Saver) but this time the number is in Milliseconds and not points (scripts can be confusing sometimes) 1000 milliseconds is one second as translated to you and me, I discuss this a bit farther down when you reach the ball saver portion.

Everything is marked in the script, and if you just read what I have said just above this, then you should be fine. In case you do mess up somehow, I've included a copy of the user input section at the very bottom, so you can delete the section in the script editor, and just copy and paste the defaults back in.

A Note

Because fpx uses a menu system, and has different "levels" and adjustments built in, the changes you make here should be considered the default settings when your table is loaded in for the first time. fpx also has a save/load routine that stores the settings from the built in table menu automatically, so you don't have to worry about that.

If you change the settings in the menu, you need to EXIT the table and restart the table to use the new settings. This is required by the engine, and helps prevent errors. (or people cheating as well)

*** Table adjustments**

This controls the main table settings, replays and balls per game

code:
```
Const constTableName = "fpxTemplate"
```

This is the actual name of your table for the config file for Load/save values. (unless you want your table name to be forever called fpxTemplate, which is fine by me.)
Do not add spaces or special characters, keep this one single uninterupted word for now as this may be used down the road by fpx. Really it just tells people what the name of the table is.
Remember what I said about changing the text, change ONLY the text, not anything else, including the quotes. Make sure your text is BETWEEN the 2 quotes, and there are NO OTHER QUOTES. There should only be 2 quotes, and only text between the quotes. Leave everything else alone. Do not touch, modify or even stare at it for too long, it doesn't like it.

code:
```
Const constAuthorName = "by shiva"
```

Who made it, you can always just give me the credit if you want, but you might just want to put in your name instead. Again, Text between the quotes only

code:
```
Const constMaxPlayers = 4
```

Ah, we can change a number now, but we don't need to with this one. This is the maximum number of players per game (between 1 and 4), and nearly every pinball game in the last 60 years are 4 player games

code:
```
fpxReplay1 = 500000
```

The first point total a player has to reach to score a free game. If your game scores over this setting (500000 points) then it will automatically award a free game and give you a nice display and light show. I used this as the example above, remember, just the numbers, no commas. You can actually set this as high as you want, the engine shows up to 1 billion points, but don't make the replay too high or people will find it too hard.

```
code:  fpxReplay2  =  75500000
```

Second Replay award, same as the first really, the default here is set to 750000 points

```
code:  fpxHighScore  =  1250000
```

This is the High Score award default value. If the High Score is made, the new high score is saved. Like a actual pinball table, making the high score will award 3 credits (3 free games)

```
code:  fpxMaxCredits  =9
```

This is the maximum amount of credits a game will allow you to have, usually 15 maximum with Bally's standard settings, but you can set it to any value. A lot of operators changed this to 9 so people couldn't rack up 40 plus credits and then leave then there so everyone else can then use and have 40 free games without paying a cent. Wasn't good for the coin box revenue.

A Special Note

fpx supports 3 ball and 5 ball play, as well as high scores for both 3 and 5 balls.

*** Music/System sets**

Used to select from one of the multiple music/sound sets. Note the code here matches the beginning of the sound files name. The engine can support a unlimited amount of music sets, fpx has included 3 music sets based on era and company. (more are coming soon, just takes time to make the sounds/code them/test them etc)

Each music set also contains preset pointers to the display for special effects like scrolling text, and also special routines for the lights using the LightSeq feature within Future pinball.

A Note To Coders

You can also use the code within your hit event as well, so you can change music sets right in game. In fact, if a player gets bored with one music set, he can just change it to a completely different one any time he wishes. All supported music sets within fpx can be used by any table at anytime, so you can make a more modern table sound like a em table.

If you wish to use a different music set, then comment out the active line and un comment the line you wish to use. In this example, wsys7 is active, just put a apostrophe (') in the front to turn the line to a remark, and remove the apostrophe from one of the other lines. Setting the MusicSet to "off" means no music/display text/or lightseg. This is for coders who wish to put in their own custom code or use their own combination of presets.

```
code:   ' MusicSet="Off_"      ' This switches OFF ALL
       MUSIC/DISPLAY/LIGHTSEQ routines so you can code all
       that yourself
        ' MusicSet="Bally81_"     ' Set the
       music/lightseq/display for Bally 1982 era tables
        ' MusicSet="w79_"     ' Williams system 6 era
        MusicSet="wsys7_" ' Williams System 7 era
```

You can find out more in the AddMusicSet Page

*** BAM settings**

note: These are used by the BAM program, so I will explain these in a bit more detail but the default settings usually work great for most tables so you don't have to change these at all.

```
code:  xPhysicsXML=1
```

This sets the xml physics package you want here. Future Pinball has horrible gameplay and bad physics built in, so a few years back, the community started to come up with a way to improve the gameplay physics. When BAM (or Better Arcade Mode) came out, it added a lot more capabilities and far better physics so FP now plays a lot better. There are tons of Physics XML files out there, what I did was choose a few that seem to be the best to match certain types and eras of the actual pinball games. I will be adding more based on support for new styles, but these are very good general physics setups.

0= No XML(default xml in BAM (NOT TESTED YET))
1=wpc (1990) (BAM Team)
2=Bally 1981 (shiva)
3=DataEast(1985) (GeorgeH)

```
code:  fpxSetBounce=7
```

Forced Maximum Ball Speed
Though the just above xml files sets the general physics, some people prefer different settings for how the ball "bounces" off the flipper. Future Pinball has always been pretty bad at that, the ball really "dies" once it hits the flipper, and it's not very realistic to say the least. fpxSetBounce sets the amount of "flipper Bounce" from a simple value of 1 to 10, with 1

being the least amount of bounce and 10 being very bouncy. I usually find that 7 is a good default value for the Bally and Data East games in the 1980's. EM (electromechanical) games from the 1970 era should be a bit lower, around 4 or 5. Adjust this to your tastes, if you go over 10 then it won't work at all, but 10 is quite bouncy.

```
code: xMaxBallSpeed = 3500
```

Set the maximum speed a ball will play. This adds the value to xBAM.BallSpeedLimit that is part of the BAM code. Sometimes the FP ball just moves too fast, so this stops it from "speeding". 3500 is a good starting point, a higher value (like 4000) will mean the ball will go faster as a max speed, while a lower value (like 3000) means the ball will so slower as a maximum speed
I recommend that you do not go below 2500 (which is quite slow for a modern game, though good for a EM game) or higher than 5000 (which is very very fast)

Ball Shadows
New to BAM are a shadowing effect under the ball as it moves around the table. This feature is built into BAM itself, but is switchable. You can set it to "0" to not have ball shadows at all, "1" to use the default settings that BAM has for a ball shadow, or "2" which is a custom set up for fpxEngine.

```
code: fpxBallShadows=2                                        ' 0=Ball shadows Off:1=BAM
      Default:2 = custom setting
```

fpxUseShadowMaps

```
code: fpxUseShadowMaps=1                                      ' Turns On (1) or Off (0) Dynamic
      BAM shadow maps
```

fpxEngine now supports dynamic lighting and shadow effects. This effect adds shadows based on the light source for posts, targets etc. Because this effect is very "heavy" on a computer,fpx only uses 4 bulbs (the 4 bulbs used with the slingshots) to generate. People with old or very slow computers MAY find that even using just 4 bulbs crashes their table, so this switches the effect on or off. Because of this, we only use the 4 bulbs, 2 at the bottom (slingshots) and 2 at the top (Header bulbs). Future Vault items MAY add a additional "bulb" at the top so the effect is better, as well as any flashers also found in a advanced Vault item. This will be noted in the script and also here in the manual for that vault item.
A Note to coders:

A Note To Coders

xBAM.RemoveFlippersFromShadowmaps, xBAM.Lights.EnableNewRenderer, and xBAM.Lights.EnablePostprocessing is set to TRUE. You can find the code by searching for it in the BAM section.
The defined Bulb Lights for Dynamic Shadows are LeftSlingshotBulb1, BulbLHead2", rightslingshotbulb1, and BulbRHead2

Flipper Omega (Strength of Flippers)

```
code: const MaxOmega  = 50          ' Default Max force
      for All flippers. Must be > MinOmega.
      const MinOmega  = 18          ' Default Min force
      for All flippers Must be < MaxOmega.
```

For coders familiar with BAM, this sets how strong and how weak the flippers are when it shoots a ball. fpx has 2 settings, MinOmega for the flipper to handle slow balls, and MaxOmega for flippers to handle balls that move faster
The higher the number (value), the stronger the flippers are, The lower the number (value), the weaker the flippers are
If you don't like how weak the flippers are then change MinOmega to a value higher than the default setting of 18. (as a example MinOmega = 22) The same with MaxOmega, lower the default (50) means the shots are weaker and may have problems with the ball going up ramps, while setting the value higher means the ball moves faster off the flipper.
Some people prefer their own settings, a common one is const MaxOmega = 46, const MinOmega = 24.

The settings as default are pretty much how I like it for most general tables. The MaxOmega is good for most ramps, without making the ball move to fast for other areas, and the MinOmega is the weakest a flipper can fire the ball, and that is pretty good for lower side items and overall appearance if you "cradle" a ball with your flipper, and then drop the flipper to shoot the ball somewhere.

More detailed information can be found at the BAM page.

*** Debug and menu**

There's a built in Debug menu in FP, if you press the F9 key in the editor, it will load the table and a special debug menu as well. Unfortunately, the help file in FP is not very helpful about this, and it's really not needed all that much

```
code: Const constAddDebug = 1
```

This is a switchable debug text to track things (0 = off 1 = On) You need to press the F9 key in the editor to start debug table view. This is for my development of the engine. This creates a text file called fpDebugTextLog in the same place as where your table is, and lists any debug code that I want in the script. constAddDebug is a special statement, so I can "divide" the debug system into 2 separate parts and just have displayed the part I want and not the other. Leave this at 1 to track the subroutine calls and how the script is being executed. 0 is used more for the reporting side of things, like the version of bam, xml used etc.

More detailed information can be found at the Debug Page

*** Ball Saver**

this sets the time for the ball Saver feature to be active when a new ball is fired from the plunger and enters the playfield to start play. By default, this is set to 10000 milliseconds or 10 seconds (1000 milliseconds = 1 second, 2500 milliseconds = 2.5 seconds) If you put in "0", the BallSaver feature is switched off.

```
code:  fpxBallSaverTime  = 10000
```

More detailed information can be found at the Ball Saver page

*** Extra ball adjustments**

```
code:  fpxMaxExtraBalls=1
```

This sets the limit to amount of extra balls a player can earn per ball. It's set to one, a higher number may not actually work yet as I haven't even tested it so don't bother changing it for the time being

```
code:  fpxMaxExtraBallAlternateScore=25000
```

In case a player already has made a Extra Ball, and scores a second extra ball on the same ball in play, this instead will give a alternate score in points. Default is 25 thousand points added to the score, you can change it if you want.

More detailed information can be found at the Extra Ball Page

*** Audio settings**

You can have background music playing in fpx, there's background music added with each game style using the era you type for *`MusicSet`*. If you do not want background music to play, set `PlayBackgroundMusic` to 0
 A lot of solid state games had their background sounds increase by frequency (or pitch) as the player collected bonus (most notably william's games like Black Knight).
If you set BackgroundIncreaseFrequency to 1, this will play the background sound at a slightly higher frequency every time a bonus is made. The frequency will reset back to the original starting background after every 29 bonus. If you wish to turn off this feature, set `BackgroundIncreaseFrequency=0`

```
code:  PlayBackgroundMusic  = 1 ' 0 = off 1=on You can skip
       this one, most people want a background sound.
       BackgroundIncreaseFrequency=1 ' increases background
       music frequency as bonus is collected
```

This sets the master volume. It's script able for more powerful amplifiers in some computers, or if the music and sound is just too loud for the table. FP has it's own settings, these settings will override them. You can adjust the numbers to your own tastes, the stock fpx sounds and music were adjusted to play at the same volume levels, but if your computer plays music very loudly, you can just set the volume here once you like and then never have to touch it again. You will find (and quite surprising at that) that you do have to change the volume with a few tables, and most do not have these settings.

For some reason, it's not 1 to 10 to change the volume, its 0.0 to 1.0

Don't ask me why, I have no idea, it was something the FP dev decided to do, so we are stuck with it.

```
code: fpxMaxMusicVolume  = 1.           ' Changes the master
      volume level for music (0.0 to 1.0)
      fpxMaxBackGroundVolume  = .7      ' Changes the master
      volume level for background music (0.0 to 1.0)
      fpxSoundVolume  = .6              ' Changes the volume
      level of the mechanical sounds (0.0 to 1.0)
```

*** Jackpot**

A lot of pinball games have a jackpot feature, usually for Multiball so fpx has one too. You only need to set the minimum and maximum values for the jackpot, and there is a second Jackpot (super jackpot) that takes the present value of the jackpot award and adds a multiplier to it. To turn of the jackpot, you can set the fpxJackpotmin and fpxJackpotmax values to 0 (zero) other wise, it's the amount of points you want.

```
code: fpxJackpotmin  = 10000                 ' minimum
      Jackpot a player can score (In points)
      fpxJackpotmax  = 250000       ' Maximum Jackpot a
      player can score (In points)
      fpxSuperJackpotMultiplier  = 2      ' multiples the
      existing Jackpot score by this multiplier
```

More detailed information can be found at the Jackpot page

*** Mystery**

Commonly found on Williams games (Like Black Knight), the mystery feature would give a random score when made. You can set a minimum and maximum value for the mystery here.

```
code:  ' Random score between these two values
       MinMysteryAward=5000                ' Min Mystery
      award
       MysteryAwardMaxium=25000               ' Max Mystery
      award
```

More detailed information can be found at the Mystery page

*** Bonus**

This handles the bonus count. You set the Interval (in milliseconds) between each count. If you have a lot of bonus, or you want the count to speed up between each count, set the SetTimerCountSpeedUp = 1 and the amount of time you want to decrease the time between each count, and watch her fly...

If you want the time between each bonus to be the same (like nearly every EM game and all early solid state games, then just set SetTimerCountSpeedUp = 0 and the engine will ignore everything else, and just use the same time between each bonus count.

This default settings sets the time between counts at 400 ms (miliseconds) (SetTimerIntervalDecreaseby), and then decreases each time by 20 ms. (SetTimerIntervalDecreaseby)
There's code in the engine bonus timer that prevents the TimerBonus.Interval from going below 30, or it looks bad.

```
code: BonusAmount = 1000
```

This is the score for each bonus count (usually 1000 pts for modern tables)

```
code: BonusCountMax = 39
```

The maximum amount of bonus you want to have in your game. fpx will support up to 159 bonus, and even has custom routines for that, but most EM & Early Solid State pinball tables had 29, while the games from the 1980's mostly had either 39 or 59. By default, fpx is set to 39. In the Editor, you have Bonus Lights created for you, do not delete any of them as this will cause errors during game play. If you keep the setting at 39, then the 40k and 50k light is not used, so it is best you left click on them and drag them to underneath the apron so they are not visible. If you set the bonus at 40 or over, you will need the 40k light, and same if you want 50 or more bonus, then leave the 50k light alone.

```
code: MaximumBonusMultiplier  = 5
```

Like BonusCountMax, but used for the Bonus Multiplier system. This is the maximum amount of multipliers you want for your game. The default is 5 (or 5x as you may know it) but fpx can handle up to 15x, and has it's own automated routines to handle and display it. If you are at 5x or below (like 4x) you can drag the 1x button to underneath the apron to hide the light as it is not used, but if you want a bonus multiplier over 5 then you will need it with the other multiplier light

```
code: SetTimerInterval  = 400
```

The time you want between each count, or the initial time ...by default, this is set for 400 milliseconds as the stock time between each count. It's a good general speed, not too slow, not too fast

```
code: SetTimerCountSpeedUp  = 1
```

If you want the bonus to increase speed with each bonus count and decrease the amount of time between each count as set by SetTimerIntervalDecreaseby just below.(1 = On, 0 = Off) Setting this to 0 is "EM Style" so it's the same time for each count

```
code: SetTimerIntervalDecreaseby  = 20
```

And how much in milliseconds you want to decrease the amount of time for each bonus count if SetTimerCountSpeedUp = 1. SetTimerInterval sets the initial starting time, and SetTimerIntervalDecreaseby will decrease it by the amount in milliseconds you set. For example, using the default examples, when your ball is lost the bonus Count routine will start up and start counting out the bonus. the first bonus plays, then there is a 400 millisecond delay(SetTimerInterval) then the second bonus starts, with a 380 ms (millisecond) delay, the 3rd bonus is a 360 delay, the 4th bonus is a 340 delay and so on.

*** Delay at Drain**

this sets a delay (in ms) at the drain before the bonus count starts. Just leave this like it is for the moment

```
code:  fpxBonusDelayTime  = 20
```

And you are done the first section. The next section is the Hit Code Section, where we tell the script what to do when a game is in progress, what happens when the ball hits a object (like a trigger or bumper), the game rules.

User Input Code Copy

User Input Code Copy

```
'
' This is a copy of the User Input section, just in case you
need it. Just remove the entire user input section from the
script, and then copy and paste this entire code to restore
the User Input
' Back to it's default settings
"
'
***************************************************************************
' **
       **
' **                         User Define Section
       **
' **
       **
'
***************************************************************************
```

```
' Everything in one place, what a concept....
' Lets begin.
' *** START HERE ***
' OKAY, NOW YOU CAN CHANGE THINGS, TILL YOU GET TO THE
OPTIONAL  SECTION
' This is all the main settings so you can customize your
table. Things like the amount of balls per game, etc. Think of
this (in actual arcade pinball terms)
' as far more advanced "pin settings". For the most part, all
you need to change is the number at the end of the code (not
these remarks, the multicolored
' part, or type in a name or text between the quotes (again in
the multicolored part)
' NOTE: To try and prevent a lot of lines of text here
explaining everything, which can be rather excessive and a bit
of a slog to go through (which
' defeats the entire concept of fpx) I have gone into far
greater detail in the manual. Look for the User input section
in the Getting Started chapter, I go
' Everything step by step, and even teach you non-coders some
very basic, minor things about coding.
' * Table adjustments
 ' name of your table for the config file for Load/save
values.
 Const constTableName = "fpxTemplate"                    '
Name of the Table. Make this unique
 Const constAuthorName = "by blue"                     ' who
made it, you can always just give me the credit if you want
 Const constMaxPlayers = 4                          ' Maximum
number of players per game (between 1 and 4)
 Const fpxReplay1 = 320000                         ' First
replay award
 Const fpxReplay2 = 500000                         ' Second
Replay award
 Const fpxHighScore = 750000                        ' High
Score award default value. If the High Score is made, the new
high score is saved.
 Const fpxMaxCredits =15                          ' Maximum
amount of credits a game will set. usually 15 with Bally, but
you can set it to any value
  nvBallsPerGame = 2                          ' Set Balls per
game (overrides the Table info, which is very flaky and
usually doesn't work anyway)
' * Engine Style
 ' Used to select from one of the multiple music/sound sets.
Note the code here matchs the beginning of the sound files
name. The engine
 ' can support as many different soundsets, but fpx has just
the one so far.
 ' NO NEED TO CHANGE THIS AT THE MOMENT
 ' MusicSet="Off_"      ' This switches OFF ALL
MUSIC/DISPLAY/LIGHTSEQ routines so you can code all that
yourself
 MusicSet="Bally81_"    ' Set the music/lightseq/display
' * BAM Physics settings
 ' note: These are used by the BAM program, so I will explain
these in a bit more detail but the default settings usually
work great for most tables
 ' so you don't have to change these at all.
 ' Set the xml physics package you want here. 0= No
XML(default xml in BAM (NOT TESTED YET)) 1=wpc(1990)
2=Bally(1982)  3=DataEast(1985)
 xPhysicsXML=1
 fpxSetBounce=7                ' set the amount of "flipper
Bounce". From 1 to 10, with 1 being the slowest and 10 the
fastest.
  xMaxBallSpeed = 3500           ' Set the maximum speed a
ball will play.
 ' For coders familiar with BAM, this sets how strong and how
weak the flippers are when it shoots a ball.
 ' fpx has 2 settings, MinOmega for the flipper to handle slow
balls, and MaxOmega for flippers to handle balls that move
faster
 const MaxOmega  = 50            ' Default Max force for All
flippers. Must be > MinOmega.
 const MinOmega  = 18            ' Default Min force for All
flippers Must be < MaxOmega.
' * Debug and menu
 ' Switchable debug text to track things (0 = off 1 = On) You
need to press the F9 key in the editor to start debug table
view.

 Const  constAddDebug = 1
' * Ball saver adjustments
 ' In Modern arcade pinball games, if the ball should drain
before a certain time, then the game will give you a freebie
ball. This is called BallSaver
 ' Set in ms (miliseconds} for ballsaver time (1000 ms = 1
sec)
 Const  fpxBallSaverTime = 10000          ' this is set to
10000 (10 seconds) by default

' * extra ball adjustments
 ' You do know what a extra ball is don't you? You can skip
this one
 Const constMaxExtraBalls=1           ' sets limit to amount
of extra balls a player can earn per ball
 Const fpxMaxExtraBallAlternateScore=25000 ' Alternate score
in points in case you max out the amount of extra balls with
```

```
one ball
' * Audio settings
 ' You can play background music, you can skip this one, most
people want a background sound.
  PlayBackgroundMusic = 1             ' 0 = off 1=on
 ' This sets the master volume. It's scriptable for more
powerful amplifiers in some computers. FP has it's own
settings, these will override them
 ' This is put in because a lot of people don't adjust their
volume, so sometimes it can be way too loud.
 '
 Const  fpxMaxMusicVolume = 1.              ' Changes the master
volume level for music (0.0 to 1.0)
 Const  fpxMaxBackGroundVolume = .7           ' Changes the
master volume level for the background music (0.0 to 1.0)
 Const  fpxSoundVolume = .6                ' Changes the volume
level of the mechanical sounds (0.0 to 1.0)

' * Jackpot
 Const  fpxJackpotmin = 10000              ' minimum Jackpot a
player can score (In points) To turn off Jackpot, set both min
and max values to 0
 Const  fpxJackpotmax = 250000             ' Maximum Jackpot a
player can score (In points)
  fpxSuperJackpotMultiplier = 2              ' multiples the
existing Jackpot score by this multipler

' * Bonus
 ' This handles the bonus count. You set the Interval (in
milliseconds) between each count. If you have a lot of bonus,
or you want the count to speed up
 ' between each count, set the SetTimerCountSpeedUp = 1 and
the amount of time you want to decrease the time between each
count, and watch her fly...
 '
  BonusAmount = 1000                           ' score for each
bonus count (usually 1000 pts for modern tables)
  BonusCountMax = 39                           ' maximum amount
of bonus (Up to 159 bonus)(EM & Early Solid State had 29,
1980's Bally/Williams had 39)
  MaximumBonusMultiplier = 5                   ' Forced max
amount of multiplier(up to 15x)
  SetTimerInterval = 400                       ' The time you
want between each count, or the initial time ...
  SetTimerCountSpeedUp = 1                     ' if you want the
bonus to increase speed with each bonus count ...( 1 = On, 0 =
Off )
  SetTimerIntervalDecreaseby = 20              ' and how much in
milliseconds you want to decrease the amount of time for each
bonus count
```

```
' * Delay at Drain
 ' this sets a delay (in ms) at the drain before the bonus
count starts. Just leave this like it is for the moment
 Const  fpxBonusDelayTime = 20
' * Memory System
 ' REMARKED OUT AT THE MOMENT. IT DOES WORK, it just needs
more help written.
 ' This stores the light state from the previous player ball
to his next ball, even with Multiplayer. The script takes a
"snap shot" of the lights
 ' defined here, and remembers it's state for all players.
Just make sure MemorySlot(x) is in numeric order (1,2,3 etc)
and you have 99 slots(or 99 lights)
 '
 ' This example just remembers the top guide triggers, any
lights "on" will be remembered at the start of that players
next ball.
 ' You can set any light to be "remembered" and if needed, add
custom code in the AddEvent "New_Ball". The engine itself
handles the rest.
  'Set MemorySlot(1)=LightTriggerGuide1                      ' 4
Top trigger lanes.
 'Set MemorySlot(2)=LightTriggerGuide2                      '
the code automatically handles Light States in the triggers
Hit_Event
 'Set MemorySlot(3)=LightTriggerGuide3                      ' So
we just need to add them to the memory slots
 'Set  MemorySlot(4)=LightTriggerGuide4
 'Set MemorySlot(5)=Light20k
'  NOTE: The memory system does require coding, its very
advanced it will require very detailed instructions, so for
beginners just ignore this for the time being
'
```

Hit Code Section

Hit Code Section

Beginner's Guide »

Hit Code Section

Contents

And now, we get to actual computer coding, starting with subroutines

Just a brief explanation before we start. In the previous page the user input section, we just put in numbers and changed the text, but if you noticed, in direct code. This type of code, and all the other code in the engine that is "by itself", is run automatically by FP as soon as you press start in the editor. Subroutines are different, this is code that is only run when you tell it to, and nearly all of it is completely ignored by FP at the start of a table. (other than special routines, which is part of the Future Pinball core)

In other words, this is code you have to tell FP to run, and when you want.

SubRoutines are your friends

Subroutines are actually quite easy to setup. If you look at the first batch of code in the Hit Section, you will notice they are all quite similar, and all have a common way of doing it. When I built the script for fpx, one of the things I decided was instead of having multiple ways of doing coding, and then confusing you by trying to teach you every single method, instead, I do everything as much as possible just one way, that way it's a lot easier for you to learn, and once you understand the basic concept, you will be able to understand everything else because it is just the one way.

Lets take a sample code and show you how it's done...

```
code: Sub fpxMyTrigger_Hit
        'some  code....
      End  Sub
```

First off, you have to tell FP that this is a subroutine, so you need to type in "**`Sub`**" and then a space.

Right after is "`fpxMyTrigger`".This is the name of the actual object that you place in the editor. When you create a object in the editor, you also can name the object in the top righthand side under the options menu. The name of the object in the hit code must match the name of the object in the editor. (You also can not use that name for any other object in the editor, all object names must be unique)

right after that is "`_Hit()`" this tells FP what to do if the ball "hits" that object, and will only run if that object does. Notice that a underscore has to be included, it's a code thing, and though the 2 brackets are empty, you should add them as habit. I will get to that one in time....

The second line is the actual code within the subroutine for fpx to run. (This is just a example, so it's remarked out, it doesn't do anything)

The third line tells the script there's no more instructions within that subroutine. You have to tell FP the subroutine is finished so that is what the "**`End Sub`**" is for. If you don't have a End Sub to close the subroutine, Future Pinball will throw out a error and refuse to play until you fix it.

AddScore ( )

Now you know what a subroutine is, and of course you want to add points to a players score don't you? That's what AddScore does. You just type AddScore and the two brackets, and put a number between the brackets that is the points you want to add to the players score. I included the AddScore right in each subroutine, so you can change it. It's exactly like the User Input Section, except you have to change the number between 2 brackets (which you do not delete by the way.)

Lets use AddScore to give some points, this time 500 points is added to the players total.

```
code: AddScore (500)
```

Now lets use a preset already in fpx, `"fpxAdd1000".` This is the name of the actual object that is in the stock fpx template. You can find it at the top left loop lane, it's a star trigger and it's very simple code, as all it does is give out 1000 points every time the ball rolls over it. If you click on the trigger, you will notice the name of the trigger in the very top box in the options menu on the right side. The name of the object in the hit code must match the name of the object in the editor. (You also can not use that name for any other object in the editor, all object names must be unique).

Lets show you the actual code for the trigger, called fpxAdd1000

```
code: Sub fpxAdd1000_Hit
        AddScore (1000)
      End Sub
```

Pretty simple isn't it? Lets show you how to add a bonus

AddBonus

I'm pretty sure if you have ever played a pinball game, you would know what a Bonus does. The player collects bonus throughout the ball in play, and then when he loses the ball, the bonus is counted down and then added to the players score before the next ball starts. Same thing here, using the AddBonus feature. It's done the same way as AddScore, the only thing is the number is the amount of bonus you give, and for nearly all occasions, it should be just 1.

```
code: AddBonus(1)
```

Some tables sometimes gave out 3 for the outlanes like the 1970's Ballys

```
code: AddBonus(3)
```

Lets show you the built in example to add a bonus. There's another start shaped trigger, this time at the entrance to the top right loop. Every time a ball rolls over this trigger, it increases the bonus count by one. Here's the code for that trigger.

```
code:  Sub fpxAddBonus_Hit
         AddBonus (1)
       End Sub
```

That's it. Now again have a look at the first group of subroutines, makes a lot more sense now doesn't it?

In fact, the first 2 subroutines are all pretty much the same way, just the name of the object they point to and the middle code that subroutine points to is different. They are all subroutines, they all run when "hit", and they all have End sub at the last line..

Still, this is the most basic example possible, there's no sound or music even. You will need to add AddScore(), AddBonus (and AddJackpot, which we will get to a little later) if you want the player to score points or increase his bonus, but fpx has a lot more to offer than just that... It's time to get to the magic of fpx, the ability to have complex scoring routines all by just by copying and pasting a couple lines of code into a subroutine.

AddScoringEvent is your Best Friend

AddScoringEvent is a special subroutine, and one you will use a lot. You will see it all over the place, and was designed to have all the scoring features available in fpx in one place, that you can access with one or two lines of code. All you need to do is write AddScoringEvent and the feature you want to use typed in between the brackets. In fact, the next group of subroutines do exactly that, in fact everything in the hit code does exactly that. So not only is that very easy to use, you can reuse those anytime you want anywhere you want, and as many times as you want. All you do is quite literally "point and shoot", or as most of you are likely to do, just copy and paste in the middle between the opening subroutine line and the closing end subroutine line.

fpx has multiple AddScoringEvents already built in, a lot more will come, and all will work exactly like that, just one line. Advanced coders will point out that these are presets and can be limiting, but then, some of the code with AddScoringEvent can be over 50 lines of code. Even intermediate coders have a lot of problems with code, there's a lot more to scripting than adding some points and flashing lights, there's things like checking variables (numbers assigned to script objects in code) checking if the game is in progress, special instructions if other objects have been hit, tilt state, and many other things. As a beginner, if you see all that code just for one of the AddScoringEvent features, you just won't want to touch it or even code that. And I don't blame you, I didn't really want to write it either.

If you are a coder though, well, pretty simple way to add your custom code and/or use the built in preset routines within fpx. But that's in another section.

Okay? Lets go to the code for the slingshots, it's the simplest ones to use, and we need to add points to a player's score using AddScore.

You will notice the AddScoringEvent code with text between 2 quotes through the hit section. This is also a select case subroutine, the text between the quotes is the name of the CASE, and what we are telling the script is to go to the AddScoringEvent subroutine, find the case that is typed between the quotes, and run the code just for that case, and ignore the other cases that is there. It's basically shorthand, instead of typing multiple subroutines for each piece of code, we can put it all in one place, sort of a huge filing cabinet drawer, with each case a folder with the name of that case in that drawer, and the code as pages in that folder.

```
code: Sub LeftSlingshotRubber_Hit()          ' The Left
      Slingshot has been Hit
       AddScore(10)              ' Add some points. We add
      the value in the brackets, in this case 10 points,
      to our score
        AddScoringEvent "LeftSlingshot"          ' Call the
      preset routine for bulbs flashing/music etc
      End Sub
```

Lets do the right slingshot now. You will notice that the object name has changed, because we want to run FP when the right slingshot is hit with the ball instead of the left side. But also, you will notice the middle line has also changed. Instead of "LeftSlingshot" it's now "RightSlingshot", as in code, we want to flash the bulbs directly underneath that slingshot, and since there are 2 pf bulbs under each slingshot, we just want to flash those lights and not the opposite ones. So we need a routine for each slingshot just for those bulbs, and that's why we have 2 separate pieces of code, one each for the left side and right side slingshots.

```
code: Sub RightSlingshotRubber_Hit()          ' The Right
      Slingshot has been Hit
       AddScore(10)              ' Add some points. We add
      the value in the brackets, in this case 10 points,
      to our score
        AddScoringEvent "RightSlingshot"
      End Sub
```

Most slingshots in the actual arcade tables score just 10 points, so that's what we do here.

Working with a light or bulb

If You Have Never Coded Before

We are going to be dealing with working with a light to your table, and also now what the script needs to do if that light is turned off, or turned on.

I would first recommend that you quickly review the light section in the fp manual first, as it gives you a lot of very helpful information on some of the settings, and explains them quite well. Just press the F1 key, or click on "Help" at the top menu in the editor, and "Open Manual". Go all the way down to "Table Components", click on that to expand the section, and then select "Lights" and finally "Round Lights" Whew, it's actually harder to use the help file than it is to use the fpx engine so far isn't it...

For the most part, you don't have to worry about the settings, you will be using a light I already created for you, but the one part of the help file you should pay the most attention to for this example is "State". Here, let me copy and paste the part that you will be using...

'BulbOff - Turns Off the Bulb.
'BulbOn - Turns On the Bulb.
'BulbBlink - The Bulb will follow 'pattern' defined by the Blink Pattern field.

Most actual arcade games has a light at each of the inlane/outlane lanes, when that light is "OFF" then it will score one value, but when that light is "ON" it will score a completely different value. No matter what you can do, there is still some code you may have to do, so this is just the easiest way I can think of to show you and teach you how.

Lets start by you going into the editor. On the left hand side, just above the slingshot/flippers, are 2 lanes, with a red colored light and a white colored light. Put your mouse cursor over the white light and then click the left mouse button to highlight it. You will notice the left side of editor changes, and now details that light. The first thing you will notice is the name of that light (LightLeftInLaneTrigger) Most of the objects you use in FP have names assigned to them, and those names must be unique (you can't call a second object the same name as another object, each name must be different from each other.

Scroll down that left hand side menu (called Options in the editor by the way) and you will see the State is set to "BulbOff". (don't worry about the 2 additional settings below, we will get to that down the road) This tells the script the bulb is turned OFF at the very start. We can do this in the code as well, so if we want that light to be OFF, or switched OFF, we type in this:

```
code: LightLeftInLaneTrigger.State = BulbOn
```

Lets break that single line of code into 3 parts to make it easier to understand.

LightLeftInLaneTrigger
- is the name of the light, and should be exactly the same as the name of the light in the editor. 9 times out of ten, if a light doesn't work, it's because the name is not right in your code and doesn't match exactly the name in the actual editor. (it's okay, I do that one multiple times a day myself)

State =
- This tells FP You want to change or set the light bulb. you need the period and the equal sign in this part, it's a coding thing....

BulbOn
- I guess when the person who coded FP, he decided to be REAL accurate, and called it by bulbs. Whatever, as you can tell by now, BulbOn means Turn on the light

Okay, pretty easy so far, lets give you the code for turning off that same light

```
code: LightLeftInLaneTrigger.State = BulbOff
```

So instead of "BulbOn", it's now "BulbOff" to turn off the light. Lets test it out...

At the left of the table, there's a second group of Triggers, more at the middle. The first one is called TriggerInlaneOn. Drag the trigger to the table to turn the Inlane Lights ON and look at the code just below.

```
code: Sub TriggerInlaneOn_Hit()
           LightLeftInLaneTrigger.State = BulbOn
           LightRightInLaneTrigger.State  = BulbOn
         AddScore(500)     ' We add the value in the
      brackets, 10 points, to our score
         AddScoringEvent "trigger"
      End Sub
```

You see we were a bit more generous with the score and gave the player 500 points to add to his score.

TriggerOutlaneOn is the second trigger in the middle group, so drag it to the table, and when the ball roll overs it, the outlane lights will turn on. You can see the difference in the name of the lights, and in the editor and in the game, they are colored red, which is traditionally what the lights are colored to score a special or free Game.
But here is also a very good place to add another line of code that you will use quite a bit, the AddBonus()
A lot of tables would give out bonuses for targets lanes etc as well, so you get the idea.
Lets add one bonus to the trigger that turns on the Outlanes

```
code: Sub TriggerOutlaneOn_Hit()
             LightLeftOutLaneTrigger.State  = BulbOn
             LightRightOutLaneTrigger.State  = BulbOn
          AddScore(500)                 ' Add some points.
      a bit more than before
          AddBonus (1)                 ' Add to the Bonus
      Count when the ball is lost
          AddScoringEvent "trigger"             ' did you
      know you can mix and match AddScoringEvents?
         ' This adds a "prompt" message to let player know
      a special is lit.
          Message(1)= "SPECIAL":Message(2)= " IS LIT
      ":Message(3)= "SPECIAL":Message(4)= " IS LIT " ' Add
      Messages for the display
          AddScoringEvent "SpecialIsLit"
      ' Tell Player a special can be collected
      End Sub
```

You will notice another AddScoringEvent (`AddScoringEvent  "SpecialIsLit"`).
This is a simple "prompt" message you can add to tell the player that a special has been lit and can be collected. This uses Messages, we will discuss this a bit further down, but what this does it allow you to add custom messages in your display when ever the player makes a certain feature or goal.

Doing 2 scores at once

So far it's pretty simple. Not too much, and I have been adding one thing at a time with each example. But it's also very common to have a trigger or other feature with a Light assigned to it, and that usually means there's two separate scores, one for when the light is "off", and another score for when the light is "On". Which we do have with the inlanes, and the outlanes, you can see those lights "Turn On" when the ball rolls over the 2 triggers we just discussed (TriggerInlaneOn and TriggerOutlaneOn)

Since we have 2 sets of scoring to do (one for the light being off, and a second for the light being on), we can do this as simply as possible by just having the script look for the state of the bulb (on or off) and then handle what to do. So, when we need to add the code, we use what is referred to as a IF THEN statement.

IF/THEN

IF/THEN are very useful, you just tell the script to look at something, and **IF** it's true, **THEN** you tell it what to do. Once you add the code, you have to add **END IF** so the script knows that is all you want for it to do if that bulb is not lit. Then, you do the same thing thing, but if the bulb is lit. With most earlier games, it's usually a different higher score and a bonus added. Lets do that with the left inlane...

```
code: ' Return lane to left flipper
      Sub LeftInLaneTrigger_Hit()
       IF LightLeftInLaneTrigger.State=BulbOff THEN    '
      Look to see if light assigned is off first
         AddScore(500)
         AddScoringEvent "InLaneUnlit_bally81"     ' Go to
      routine for music etc
       END IF         ' I'm finished telling you what to
      do if the bulb is off
      ' If the bulb is lit, lets reward the player with a
      better score
       IF LightLeftInLaneTrigger.State=BulbOn THEN     '
      Look to see if light assigned is on first
         AddScoringEvent "InLaneIslit_bally81"     ' Go to
      routine for music etc
       END IF         ' I'm finished telling you what to
      do if the bulb is on
      End Sub
```

Okay, lets do the right inlane now...

```
code: ' Return lane to left flipper
      Sub RightInLaneTrigger_Hit()
       IF LightRightInLaneTrigger.State=BulbOff THEN    '
      Look to see if light assigned is off first

         AddScoringEvent "InLaneUnlit_bally81"     ' Go to
      routine for music etc
       END IF         ' I'm finished telling you what to
      do if the bulb is off
      ' If the bulb is lit, lets reward the player with a
      better score
```

```
 IF LightRightInLaneTrigger.State=BulbOn THEN   '
Look to see if light assigned is on first
   AddScoringEvent "InLaneIslit_bally81"    ' Go to
routine for music etc
 END IF          ' I'm finished telling you what to
do if the bulb is on
End Sub
```

As you can see, we just need to change the name of the trigger (as we have 2 inlanes), and also the name of the light above it (as we have a light for each inlane). The code itself is the same.

The IF/THEN statement within your code is a very powerful tool for your coding, if you look at my main code, you will see I use it a lot. There are other statements you can use with IF/THEN (Like the ELSE statement) but that's another time and it's really not needed at the moment.

Now, lets do the outlanes. The code is the same way as the inlanes, we just want it to score a bit differently is all. Bally games usually give you a higher score, sometimes gives you more bonus to add to the bonus count (3 bonus sometimes) and also gives you a free game if the outlane lights are lit.

When you look at the code, you will notice a few things
- We now score a special if the outlane light is lit. It's the exact same code as the fpxSpecial_Hit() trigger (`AddScoringEvent  "Special"`) but we have to remember to switch that light off and the other outlane light or it will just keep scoring specials every time (which I do for you anyway) This is a very common feature, so I made it a standard feature within AddScoringEvent

TriggerOutlaneOn is the second trigger in the middle group, so drag it to the table, and when the ball roll overs it, the outlane lights will turn on

```
code: Sub TriggerOutlaneOn_Hit()
           LightLeftOutLaneTrigger.State = BulbOn
           LightRightOutLaneTrigger.State = BulbOn
        AddScore(500)
  ' Add some points. a bit more than before
        AddBonus (1)
' Add to the Bonus Count when the ball is lost
        AddScoringEvent "trigger"
             ' did you know you can mix and match
AddScoringEvents?
     ' This adds a "prompt" message to let player know
a special is lit.
        Message(1)= "SPECIAL":Message(2)= " IS LIT
":Message(3)= "SPECIAL":Message(4)= " IS LIT " ' Add
Messages for the display
        AddScoringEvent "SpecialIsLit"
                   ' Tell Player a special can be
collected
End Sub
```

It is pretty much the same way as the Inlanes, the actual code is similar.

There is one more bit of code added though to this subroutine, the AddJackpot()

AddJackpot

It's a pretty common feature now days, so really no need to explain it. What AddJackpot does is just add to the Jackpot, and it works exactly like AddScore does. Just type in the amount of points you want to add to the Jackpot that the player has to collect with the number between the 2 brackets. In fact, in the example triggers fpxJackpot_Hit collects the jackpot and fpxSuperJackpot collects the super Jackpot (which is the Jackpot times the multiplier amount you already set in the user input section) (fpxSuperJackpotMultiplier if you had forgotten)

Lets do that code for the Outlanes. You will notice AddJackpot is there now. If you want to do your first piece of modification to code, you can add the AddJackpot code to the inlanes yourself.

code:
```
Sub LeftOutLaneTrigger_Hit()

' The Left OutLane trigger has been Hit
 IF LightLeftOutLaneTrigger.State=BulbOff THEN  '
Look to see if light assigned is off first
   AddScore(3000)         ' add some points
   AddJackpot(3000)         ' Adds to Jackpot total
   AddBonus(1)          ' adds to Bonus score
   AddScoringEvent "OutLaneUnlit_bally81"   ' run
routine for music etc
 END IF
 IF LightLeftOutLaneTrigger.State=BulbOn THEN  '
Look to see if light assigned is off first
   AddScore(5000)         ' add some points
   AddJackpot(5000)         ' Adds to Jackpot total
   AddBonus(1)          ' adds to Bonus score
   AddScoringEvent "OutLaneSpecial"    '
Automatically Awards Special.
 End If
End Sub
Sub RightOutLaneTrigger_Hit()       ' The Right
OutLane trigger has been Hit
 IF LightRightOutLaneTrigger.State=BulbOff THEN  '
Look to see if light assigned is off first
   AddScore(3000)         ' add some points
   AddJackpot(3000)         ' Adds to Jackpot total
   AddBonus(1)          ' adds to Bonus score
   AddScoringEvent "OutLaneUnlit_bally81"   ' run
routine for music etc
 END IF
 IF LightRightOutLaneTrigger.State=BulbOn THEN  '
Look to see if light assigned is off first
```

```
    AddScore(5000)          ' add some points
    AddJackpot(5000)          ' Adds to Jackpot total
    AddBonus(1)           ' adds to Bonus score
    AddScoringEvent "OutLaneSpecial"     '
Automatically Awards Special.
 End If
End Sub
```

One thing, If you turn these Inlane and Outlane lights on, they will just stay on until you exit the game. Why? Because we didn't tell the script to turn off the lights when a player loses a ball, so, there is a bit more code where we turn these lights back off at the start of each new ball in play. We will show you how to do in in the Event Section and discuss how we do it there. For you coders, it's in the AddEvent subroutine, and the "NEW_BALL" case. "NEW_BALL" is where you put in all your code for the start of each ball, turn off lights, reset some things, pop up targets etc.

The built in fpx feature examples

To the left of the table there's groups of triggers. These are the examples I have put in to show off each of the main AddScoringEvent features. At the top is a group of triggers for each of the examples I give here. I would suggest reading up on triggers in the FP Help manual (under Table Components) first. There are several types of triggers available in FP, but I use the Trigger-Star ones here (the funny looking star shaped objects)

Just click on each trigger, note the name of the trigger, and then look at the code for that trigger to see what feature AddScoringEvent uses. If you want to see the trigger in actual game play, just click on that trigger, and while holding down the left mouse button, drag it right on to the table. Place it so the ball will roll over it (you should see a pretty good place to put it so the ball will only rollover once) and then press play and watch...

Because AddScoringEvent will be so huge down the road, it's best it has it's own section, as some will require certain objects to be placed and named correctly, and also coders will want not just the info on AddScoringEvent, but the presets for the display and lightSeq routines information. If you are a beginner, that one line of code is pretty easy to figure out, because I was as descriptive as possible with the naming. It's pretty obvious if you test out each trigger in gameplay as well...

So now we come to the final bit of code for the hit section, a small selection of the presets like extra ball and specials. If you had played with the triggers, or just ran the table, you will have noticed that the 4 displays have blinking/flashing text to highlight certain features. fpx has the ability to add custom "messages", those are text messages that the display will show especially if you score something big like a extra ball. You will see these type of messages all the time, and I am pretty sure you may want to learn how to do that and even change some of them.

Displaying Messages

You may have noticed in `fpxAddMultiplier_Hit` this line...

```
code:  Message(1)= "BONUS   ":Message(2)= "MULTI   "
```

This allows you to display messages instead of the players score in each of the 4 displays. You will notice it's used for a lot of things, like when the player makes a extra Ball, wins a game, you know, the exciting stuff. It's also used a lot by the engine, you will notice the `Message(x)` code lines in a lot of areas, from table power up to the match routines. AddEngineEvent gives you access so you can place your code directly into the engine, but it also gives you some things you can change to suit you.

Message(x) are the text messages that are displayed in the display when you are playing a game. There are 4 displays, so there are 4 messages you have to use. (and the 4 HUD displays as well that match the displays in the translight) So, Message(1)=Player1 display, Message(2)=player 2 display and so on.

The things you need to remember

- Only change the text message that is between the quotes ("). Do not delete or change anything else. Because there are 9 digits in each display, your words should be no more than 9 characters long. Any lowercase letters you add will show when you are playing a game, but as Upper Case letters, This is normal for non-dmd games.
- Even though you have 4 displays, each display can only display 9 characters at any time.
- Some areas you only have 1 message (like the tilt) to do, others 2 messages, but most are 4 messages.
- Sometimes, Message(3) and Message(4) will have quotes with nothing between them.This means that Display 3 and Display 4 will show completely blank while in game. You can add messages if you want, I just like how it looks.

  You can leave it alone if you like, the engine is quite happy to run the defaults I typed into it, but if you do, it's very simple... as a example, using Message(2), if you want to change the Engine text to the word Template then, remove the word Engine from between the quotes
  and then type Template instead between the quotes. So your code for the Message(2) line should look from this:

```
code:  Message(2)= "Engine"
```

To this:

```
code:  Message(2)= "Template"
```

Now, lets show you another example using `fpxAddMultiplier` . You will see this code there:

```
code:  Sub fpxAddMultiplier_Hit()
        Message(1)= "BONUS    ":Message(2)= "MULTI    "
         AddScoringEvent "AddMultiplier"
       End Sub
```

You can change both messages to what you want. Suppose you want to call "Bonus Multi" to something like "Double Bonus" . You can do so, just replace the words with in the quotes with what you want. Here's a example that shows the difference for that

```
code:  Sub fpxAddMultiplier_Hit()
        Message(1)= "DOUBLE  ":Message(2)= "BONUS    "
         AddScoringEvent "AddMultiplier"
       End Sub
```

If you notice, there are some spaces within the quotes. These allow the messages to be "centered" in each of the display, which looks a lot nicer than text being "glued" from the right of the display. Unfortunately, there is no way to change the alignment of the displays in the code. Once you set the text alignment in the editor for the translight and HUD displays, it stays set that way.

Here's the code for the triggers. It's pretty much self explanatory if you read the remarks included.

```
code:  Sub fpxBallSaver_Hit()                         ' Relights
       BallSaver
        ' NOTE: There is no Message code or music for Ball
       Saver, as it's used in different ways by the engine.
         AddScoringEvent "BallSaver"
       End Sub
       Sub fpxAddMultiplier_Hit()                        ' Adds a
       Bonus Multiplier (2x, 3x etc)
        ' NOTE: There are only 2 messages needed. The
       engine will add the Super Jackpot value in the other
       two messages (in Display/HUD 3 and 4)
        Message(1)= "BONUS   ":Message(2)= "MULTI    "
         AddScoringEvent "AddMultiplier"
       End Sub
       Sub fpxJackpot_Hit()                          ' Scores a
       jackpot
        ' NOTE: There are only 2 messages needed. The
       engine will add the Jackpot value in the other two
       messages (in Display/HUD 3 and 4)
        Message(1)= "JACKPOT"
        Message(3)= "JACKPOT"
         AddScoringEvent "Jackpot"
       End Sub
       Sub fpxSuperJackpot_Hit()                       ' Scores
       Super Jackpot (Jackpot x fpxSuperJackpotMultiplier)
        ' NOTE: There are only 2 messages needed. The
       engine will add the Super Jackpot value in the other
       two messages (in Display/HUD 3 and 4)
        Message(1)= "S U P E R"
        Message(2)= "JACKPOT  "
         AddScoringEvent "SuperJackpot"
       End Sub
       Sub fpxExtraBall_Hit()                         ' Scores
       Extra Ball
        Message(1)= "EXTRA"
        Message(2)= "BALL  "
```

```
 Message(3)= "EXTRA"
 Message(4)= "BALL  "
  AddScoringEvent "ExtraBall"
End Sub
Sub fpxSpecial_Hit()                    ' Scores a
free game
 Message(1)= "SPECIAL"
 Message(2)= "SPECIAL"
 Message(3)= "SPECIAL"
 Message(4)= "SPECIAL"
  AddScoringEvent "Special"
End Sub
Sub fpxMystery_Hit()    ' Mystery Award (Black
Knight)
 ' NOTE: There are only 2 messages needed. The
engine will add the Mystery value in the other two
messages (in Display/HUD 3 and 4)
 Message(1)= "Mystery  "
 Message(2)= "Mystery  "
  AddScoringEvent "Mystery"
End Sub
```

And now you know how to change the message display for not only this part, but all the other parts as well. You don't have to change any messages if you don't want to .

Well, you are done this section now. I wrote this manual with build 1.3, I can assure you there will be a lot more scoring features, and all will be done like this. This page may have been very long to go through, but there is only one more page left, the Events Section. (Don't worry, this one is quite a bit shorter)

Event Section

Event Section

Beginner's Guide »

The Event Section

Events are "hooks" in the engine scripts, to allow easy access to key areas without having to go through thousands of lines of script. Even elite coders place things in the wrong spot, so these are pretty fool proof way to add your own code without then spending hours debugging it if it throws up a error.

AddKeyEvent

Future Pinball has two subroutines for controlling keypress while in game. The problem with what most people have is finding the right place to put in custom keys (like adding a third flipper) without getting errors. To compound the problem, some things (like the code for the start key) has multiple routines, so it can be a long frustrating experience.

AddKeyEvent solves this problem, by using a concept called "Hooks". These are a special code that is inserted in the main code, that allows you to add additional code directly to the keys section in Future Pinball. All you have to do is add your code here for flippers/special keys hud etc.

For beginners it's best to just leave this alone for the time being, I have already done this for you.

It should be noted that in game, you use the default keys as set in the top Preferences menu in the editor, but some keys may not work if you use BAM, as some default FP keys are being used by BAM. You may have to change the following keys...

- HUD On/Off (usually the ` tilthe key(directly below the esc key)
- Pause (usually the "P" key)

  FP has a manual (Help right at the top when clicked opens up the manual) that explains the keys. You can find the portion that describes how to change the key by clicking the Preferences chapter on the left of the manual (almost at the bottom) and then Game Keys and Controls.

  Here's a copy (build 1.1) of the AddKeyEvent. Notice the new code called Select Case? I use this as much as possible, because it's very flexible, you can add on to it easily, and very straightforward. There are so many ways to do coding, but all you need to do is learn one way, and this is the way. I will discuss the Case code just below this...

```
code:  Sub AddKeyEvent(KeyEventVariable)
        Select Case KeyEventVariable
         case "LEFTFLIPPER-PRESSED"
            LeftFlipper.SolenoidOn
           PlaySound "FlipperUp",(fpxSoundVolume)
         case "LEFTFLIPPER-RELEASED"
            LeftFlipper.SolenoidOff
           PlaySound "FlipperDown",(fpxSoundVolume)
         case "RIGHTFLIPPER-PRESSED"
            RightFlipper.SolenoidOn
           PlaySound "FlipperUp",(fpxSoundVolume)
         case "RIGHTFLIPPER-RELEASED"
            RightFlipper.SolenoidOff
           PlaySound "FlipperDown",(fpxSoundVolume)
         case "SPECIAL1-PRESSED"
         case "SPECIAL1-RELEASED"
         case "SPECIAL2-PRESSED"
         case "SPECIAL2-RELEASED"
         case "HUDDISPLAY-SHOWING"
         case "HUDDISPLAY-HIDDEN"
         case "COININ"
        End Select
       End Sub
```

The One Way - Select Case

Okay a brief primer.

You will notice the AddScoringEvent code with text between 2 quotes through the hit section. This is also a select case subroutine, the text between the quotes is the name of the CASE, and what we are telling the script is to go to the AddScoringEvent subroutine, find the case that is typed between the quotes, and run the code just for that case, and ignore the other cases that is there. It's basically shorthand, instead of typing multiple subroutines for each piece of code, we can put it all in one place, sort of a huge filing cabinet drawer, with each case a folder with the name of that case in that drawer, and the code as pages in that folder.

Lets do a example using a downsized version of AddKeyEvent

```
code:  Sub AddKeyEvent(KeyEventVariable)
        Select Case KeyEventVariable
         case "LEFTFLIPPER-PRESSED"
         case "LEFTFLIPPER-RELEASED"
         ' and so on....
        End Select
       End Sub
```

```
code:  Sub AddKeyEvent(KeyEventVariable)
        Select Case KeyEventVariable
         case "LEFTFLIPPER-PRESSED"
        End Select
       End Sub
```

We tell the script to go to AddKeyEvent, look for the "LEFTFLIPPER-PRESSED" case, and run the code only that's in the "LEFTFLIPPER-PRESSED" case, and ignore the other case settings and it's code. So, the script will look at case "LEFTFLIPPER-PRESSED" and will run this

```
code:  LeftFlipper.SolenoidOn
       PlaySound  "FlipperUp",(fpxSoundVolume)
```

so it should look like this

```
code:  Sub AddKeyEvent(KeyEventVariable)
        Select Case KeyEventVariable
         case "LEFTFLIPPER-PRESSED"
```

```
            LeftFlipper.SolenoidOn
            PlaySound "FlipperUp",(fpxSoundVolume)
        End Select
      End Sub
```

It doesn't run anything else, just the code placed in "LEFTFLIPPER-PRESSED". You will notice other case settings have code with them, what happens when the left flipper is released, and also what happens if the Right flipper key is press, and released as well.

Hope you understand this, if you do, good, because everything is based around using the case code, and now you have the concept, you can now do some very cool things just with one line of code.

AddEngineEvent

Next (and the last one) is AddEngineEvent. If you read above, I talk about "hooks" which is code within the main engine that points to a subroutine with case settings. All those hooks point to AddEngineEvent.

AddEngineEvent handles all of the main engine features. This is where you can customize the engine (display/music/Lightseq etc) without having to search through thousands of lines of code by dipping into the main engine code.and also not worry about breaking the engine if you do something wrong. Only the most advanced scripters should modify the actual engine code.

About 95% of everything anyone would want to change (sound/lights/display etc) has been placed here and is heavily commented, so it should be very easy to follow and change.

Here's the list of the main places you can insert code using just the AddEngineEvents, though you can change anything you want

- "INITIALIZE" - The opening variables and code you want to execute as soon as the table loads from the editor.
- "LIGHTATTRACTSCORE" - the very start of the Light attract.
- "LIGHTATTRACT1" - One of the routines for the LightAttract displays text in the 4 displays that give the game name and the author. You can change this to your table name
- "START_GAME" - The first thing does when a player starts a game. Like LightAttract1, you may want to change the messages in there
- "NEW_BALL" - This is all the code you want to run at the start of a game and every ball in that game. You will see the code for the template example here, that resets lights/targets etc.
- "TILT" - The code for the objects you want to switch off in case the table is tilted. Just the flippers are here as a example
- "MATCHSTART" - This is the place to put "closing" code if you need it, and is run just before the match routine starts.

New Ball

Just a little section on this, this is where coders would put their custom code for the start of a new ball. Remember those Inlane and Outlane lights we did in The Hit Code Section? I mentioned we needed to turn off those lights or they will stay on forever, so this is where that code was placed.

```
code: case "NEW_BALL"
      ' THIS IS WHERE YOU ADD YOUR CODE FOR THE START OF
      EACH BALL !!!!
      '  Use this to turn on any lights, reset any
      variables you use, popup targets etc.
      ' this switches off the lights used in the Inlanes
      and Outlanes
      LightLeftInLaneTrigger.State=BulbOff
      LightRightInLaneTrigger.State=BulbOff
      LightLeftOutLaneTrigger.State=BulbOff
      LightRightOutLaneTrigger.State=BulbOff
```

⚙ Tilt

Here's another area for coders, what to do if the game is "tilted". If you scroll down a bit more, you will see Case "TILT". The engine will switch off pretty much everything (including the scoring code built into the engine) but we need to disable the flippers as well, so here's the code already there:

```
code: case "TILT"
      ' THIS IS WHERE YOU ADD YOUR CODE FOR WHEN THE GAME
      TILTS !!!!
      ' Use this to turn off things, like the flippers
      below.
      LeftFlipper.SolenoidOff
      RightFlipper.SolenoidOff  ' Can't forget to turn
      these off,
```

You can change what you want for messages, as you scroll down, you will see the same message code for those triggers we used for examples of AddScoringEvent.

Yes you are done. There's presets and optional coding, but if you have no coding experience, you wouldn't want to change that anyway. If you are looking for something to do still, just go to the next page

fpxEngine System Features

Using the fpxEngine system features

Using the fpx System Features

(This page still in progress)

List of keys used

fpxEngine uses the keys defined in your editor. (In the top Editor, under Preferences/Game Key and Controls)
There are several keys used by fpxEngine though, so here's a list of the keys

- Special2 Key - (usually the ' key set as default) Opens menu system
- H key - HUD display visable/hidden

As well, there are Debug keys using the number pad. you need to start the game by pressing the F9 key from the editor to enter debug mode.

- Numberpad 0 - Debug Main Menu
- Numberpad 1 - User settings. This writes a copy of the User Input Section. Handy to keep track of variables, or to make a copy of any fpxEngine table settings to use in your table. Just delete your user input section, and paste in the new code. This is written in code in fpDebugTextLog
- Numberpad 5 - Bam settings. All BAM variables used by fpxEngine are written here.

This section details all of the features of the fpxEngine. This engine was written to give you as the table creator a very powerful and flexible system while being as easy to use as possible.

The Table Menu

The Table Menu

The Table Menu

All fpxEngine tables now have a menu system included that allows for easy adjustment of settings. This menu (new in version beta 1.2) will allow the player to set the number of balls per game,reset the high score, and adjust other settings. As the fpxEngine matures, more settings will be included within the menu, such as BAM (Better Arcade Mode) physics settings, and dynamic lighting presets.

The menu will automatically display at the start of a table being loaded in from the editor, and will fade out after a few seconds.

The player can press the "Special1" key (usually the ' key) to display the menu anytime, and use the flipper keys to adjust the settings.

The "H" key will turn on or off the HUD display, which allows the player to see the display in desktop mode.

- You need to save the table and exit if you want any changes to the high score or the replay values (prevents cheating)
- The left flipper will go to the next option (or page) while the right flipper changes the options for that page.
- To exit out of the menu, press the "Special1" key a second time.
- The settings in the menu will always overwrite any settings in the user input section. Settings in that section are the default values for when the table is started for the first time.
- The menu system will save any changes you want when you exit the table to return to the editor (esc. key)

The Menu Options

The menu system will be expanded as time goes on, so also check this page whenever a new beta or version is released. These are the options available to the player in table mode

Balls per Game

The player selects either a 3 ball or 5 ball game. Note that the engine will adjust the replay goals and the high score automatically depending on the amount of balls, and also the difficulty level selected. fpxReplay1,fpxReplay2 and fpxHighScore, which you set in the user input section, are the base numbers for the fpxEngine default and are for 3 ball play. If a player selects 5 ball play. the engine will add a additional score to these values to make the game fairer, and also for a future feature planned for the fpxEngine. The difficulty setting will either decrease the scores needed to win a free game (easy) or increase that score (hard). This is very similar to a common feature in pinball games, and is meant to simulate the "pin-settings" that are adjusted by the operater.

Balls per game can not be changed while in game, the player must exit the table and then reload the table from the editor to make these settings work.

The Engine will automatically show the replay goals in the display when the table is in Light Attract mode.

Scrolling.

Some players use a view that scrolls the playfield with the ball, but don't like the effect, just the general views. This creates a alternative view, that is more locked down towards the bottom of the table and prevents FP from scrolling the playfield with the ball.

Difficulty

Select able as either easy, normal, and hard. At the moment, this only affects the replay goals as set in the user input section, (as written in "balls per Game" directly above) but future releases will add more features and functionality to these settings.

Plunger arrows

Most FP games use the "enter" key to fire the ball from the plunger, but it also makes exact shots with perfect weight pretty tough to do, especially for skill shots. setting this to "use arrow keys" allows you to use the UP and DOWN arrow keys instead, ad then press the ENTER key to release the plunger.

High Score to Date

This resets and clears the high score list (and players names) back to it's initial default values of when the table is loaded in for the very first time.

Changing The High Score Defaults

This uses the BAM high score list feature. When a player gets a high score, BAM bypasses the stock FP high score list.
- To change the high score list search for this line InitHighScoreEntry().
- nvS1 = "fpxenginefpx " is the names for the high score for 5 ball play. The first 3 characters ("fpx") is the initials for the first high score, the next 3 initials ("eng") is the initials for the second high score and so on.
- nvS2 are the default high score values for 5 ball play. (4 in total)
- nvS5 are the initials for 3 ball play (just like nvS1)
- nvS6 are the default high score values for 3 ball play

Note: The entire high Score routine may be replaced down the road, as it's quite confusing to most people, and a bit unwieldy. It is my recommendation that you just leave it alone.

Custom Balls

Custom Balls

Custom Ball Example

A Note To Coders

To use this feature search for Const fpxBallBlinkingOn =
This is turned off by default. (Const fpxBallBlinkingOn = 0)
To turn this feature on, change Const fpxBallBlinkingOn = 0 to Const fpxBallBlinkingOn = 1

Built in fpxEngine are several examples for additional capabilities of BAM, though they may not be actually folded into the engine. As BAM is being developed, Ravacade has done

several examples to demonstrate various features, so they have been folded in fpx. All the code has been placed in, you just need to "turn it on" to see how it works. Also included in this page are the complete explanations as written by rav that I have rewritten in places to make it a bit easier to understand.
Also included is all the scrip table code for the custom ball feature within BAM below this example.

To see this example in action, search for and change: `Const fpxBallBlinkingOn = 1`

This example will start each ball as red and blinking (for the time set in `fpxBLinkDuration`) and also when the ball hits the slingshots will "flash" a different color.
The demo table done by ravacade can be found here:
http://www.ravarcade.pl/beta/ballDemo9.fpt

This code is found at the top of the "presets" section, this are the settings you can use to adjust the effect. `fpxBlinkInterval` and `fpxBlinkDuration` are for adjusting the blinking ball effect when a ball is released into the trough area, as well as the total amount of time you want the ball to be red (in seconds, 15 is default) while `fpxFlashDuration` controls the amount of time the ball will change color when it strikes the slings.

code:
```
' (page 192 bam main thread gopinball.com)
' please see code in RightSlingshot_hit and
LeftSlingshot_hit  for  example
 Const  fpxBallBlinkingOn = 0     ' Turns on (1) or
Off (0) ball blinking and color changing at sling
hit.
 Const  fpxBlinkInterval = 0.5      ' ball blinks for
time  specified
 Const  fpxBLinkDuration = 15.0     ' ball is colored
red at new ball for time specified
 Const  fpxFlashDuration = 0.2     ' Ball flash color
for time  specified
```

The main code is found in the BAM section of the engine, also in the presets section.

code:
```
'  --------------------  Ball  Blinking
-------------------------------
' Options to control this effect are at top of the
main  optional  section
'  controls  ball  blinking  effect
Sub BlinkBalls(ball)
 Dim a
 IF  fpxBallBlinkingOn = 1 THEN            ' Check if
ball  blinking  feature  is  on
  If ball.ExtTimer1 > 0.0 Then            ' ExtTimer1
is not 0.0 when it is started after ball creation
   If ball.ExtTimer1 >  fpxBLinkDuration Then
```

```
          ball.UpdateBall  192, 192, 192
          ball.Opacity = 1.0
          ball.StopExtTimer  1
        Else
         a = ball.ExtTimer1 / fpxBlinkInterval
         a = (a - CLng(a)) * 2.0
         If a < 0 Then  a = -a
          ball.UpdateBall 60 + CInt(192*a), 60, 60
        End If
      End If
    ' This changes the color of the ball during a
    slingshot  hit
      If ball.ExtTimer2 > 0.0 Then
        If ball.ExtTimer2 >  fpxFlashDuration Then
          ball.StopExtTimer  2
          ball.UpdateBall  192, 192, 192
        Else
         Select case ball.ExtInt2
          case 1 'green
            ball.UpdateBall  60, 255, 60
          case 2 'blue
            ball.UpdateBall  60, 60, 255
          case else
            ball.UpdateBall  255, 255, 255
         End Select
        End If
      End If
     END IF
    End Sub
    ' calls ball blinking effect for ALL balls
    Sub DrawFrameTick()
        xBAM.EnumBalls  100, 0, "BlinkBalls"
    End Sub
    ' creates customball as opposed to stock fp ball
    Sub  CreateCustomBall (source)
     Dim bi
       Set  bi = xBAM_BallManager.CreatCustomBall(0)
         Source.CreateBall
    bi.Red, bi.Green, bi.Blue, bi.BallNumber
    End Sub
```

Sub BlinkBalls(ball)
The first part of this subroutine makes the ball blink for 15 second after ball creation:

So, in first "If" we check if ExtTimer1 is still running. If so, we check if that 15 seconds passed.... If, so set ball color to normal values and we disable timer with ball.StopExtTimer 1. If ExtTimer1 value is between 0 and 15 second we change ball color with ball.UpdateBall subroutine.

Second part of BlinkBalls is used to "flash" ball after slingshot rubber is hit.

Only difference is we use ExtTimer2 to count time (0.2 sec) after _hit() event and we set ball color base on ExtInt2 value.

Sub DrawFrameTick()
This is same type subroutine as NewtonPhysicsTick. If you have that subroutine in script it will be called right before new frame will be rendered.
The Line - xBAM.EnumBalls orders BAM to call BlinkBalls subroutine once for every ball on table with "BallInfo" object as argument. So, if we have 3 balls on table, BlinkBalls will be called 3 times.

Sub CreateCustomBall
All balls using this example are Custom Balls and are created automatically at the trough. You don't need to define any "custom ball" with xBAM.BallManager.DefineCustomBall anymore now. You can just use 0 (zero) as a argument to xBAM_BallManager.CreatCustomBall. With this method, you can change ball textures any time you want as well.

Both the RightSlingshot and LeftSlingshot subroutines in the hit code has special code added to do the ball changing color briefly effect.
Setting `xBAM.Ball.ExtInt2 =1` will give a green flash on the ball, while setting `xBAM.Ball.ExtInt2=2` will give a blue flash. Any other number will be set to no effect or coloring of the ball.
This is a copy of that code that you can place within your own hit code

```
code:          'Search for Const fpxBallBlinkingOn. 0 is
     off, 1 is on. pg192 of BAM main thread
      IF fpxBallBlinkingOn = 1 THEN              ' Ball
     Flash blue
        xBAM.Ball.ResetExtTimer  2
        xBAM.Ball.ExtInt2 = 2 ' sets ball flash color
      End If
```

CUSTOM BALLS SCRIPT
These are the scrip table commands as written by ravacade.
Note:
Every time you call xBAM.BallCloseTo you get in return this object. Also in every **something_hit()** subroutine you have access with xBAM.Ball to info what ball hit that "somthing" (like Sub LeftSlingshotRubber_Hit())

- **xBAM.Ball.Id**
  unique id for every custom ball. It is =-1 if it is "normal" ball, not custom
- **xBAM.Ball.Name**
  You can GET or SET ball "type". In previous BAM version if you want to change ball texture you need to define ball "type" with xBAM.BallManager.DefineCustomBall and you need to pass this "type" as param to xBAM.BallManager.CreatCustomBall. In _hit() subroutine to change that one ball you need only

```
code: xBAM.Ball.Name = Ball_A
```

- **xBAM.Ball.UpdateBall red, green, blue, dirtTextrureName, reflectionTextureName,**

reflectionInPlayfieldTextureName
Change custom ball look. Now, you don't need to define custom ball with **xBAM.BallManager.DefineCustomBall**, you can change look of one single custom ball any time in script. You can change texture or color of ball. (ONLY CUSTOM BALL).

- **xBAM.Ball.Opacity**
  Set ball opacity... like with xBAM.BallManager.SetBallOpacity function, but easy way
- **xBAM.Ball.SetBallOpacityWithEasing Opacity, Time, Easing**
  Set ball opacity... like with xBAM.BallManager.SetBallOpacityWithEasing function, but easy way
- **xBAM.Ball.ExtTimer1[b], xBAM.Ball.[b]ExtTimer2[b], xBAM.Ball.[b]ExtTimer3[b]**
  Every ball on table have 3 own timers. Unit is 1 second. You can use it in script. Also, [b]ExtTimer1 is started when ball is created. So, when you read it you will get how long that one ball is on table. Every disable timer will return 0.0. Every enabled timer will return value bigger than 0.0.
- **xBAM.Ball.ResetExtTimer timerIdx**
  (timerIdx = 1 or 2 or 3), you can use this function to reset timer
- **xBAM.Ball.StopExtTimer timerIdx**
  (timerIdx = 1 or 2 or 3), stop timer.

Additional

These are copies of forum posts made by rav, placed here for reference. Over time, this section will be rewritten.

Pleses use xBAM.Ball or fpBallId ONLY in something_hit() subroutines, not in FuturePinball_KeyPressed() or in CreateNewBall()
In CreateNewBall you have this:

Code:
```
   xBAM.Ball.Name = xBAM.Ball.Name + 1
   xBAM.Ball.UpdateBall 192, 192, 192, "Beach_452", "Beach_334",
"Beach_269"'xBAM.Ball.Opacity = 0
   CreateCustomBall PlungerKicker
```

That 2 lines xBAM.Ball may change look of any existing ball on table, but they will not change look of new ball created on PlungerKicker.

If you want to change look of ball in FuturePinball_KeyPress(), please do it this way:

Code:
```
Sub FuturePinball_KeyPressed(ByVal KeyCode)
   Dim ball
   Set ball = xBAM.BallCloseTo(0,0)
   If ball.exist Then
      if keycode = 46 then ball.UpdateBall 192, 192, 192, "Beach_452", "Beach_334", "Beach_269"
      If keycode = 47 then ball.UpdateBall 255, 255, 255, "Volley_452", "Volley_334", "Volley_269"
      If keycode = 48 then ball.UpdateBall 222, 253, 130, "Tennis_452", "Tennis_334", "Tennis_269"

   End If
```

So:
1. Find ball on table with xBAM.BallCloseTo

2. Check if ball exist
3. Change ball look (ball variable is THAT ball)

On "CustonBall(1).fpt" you have this:

Code:
```
Dim Beachball, TennisBall, VolleyBall
BeachBall = xBAM.BallManager.DefineCustomBall(192, 192, 192, "Beach_452", "Beach_334",
"Beach_269")
TennisBall = xBAM.BallManager.DefineCustomBall( 222, 253,  130, "Tennis_452", "Tennis_334",
"Tennis_269")
VolleyBall = xBAM.BallManager.DefineCustomBall( 255, 255, 255, "Volley_452", "Volley_334",
"Volley_269")
```

You don't need to remove this code. Lets say you keeped it as it is.

Now, in LeftSlingshotRubber_Hit(), you can use BallTransformations.UpdateBall like you do on CustonBall(1).fpt
... or you can write same rules like this:

Code:
```
Sub LeftSlingshotRubber_Hit()
  If xBAM.Ball.Name = BeachBall Then
    xBAM.Ball.Name = TennisBall
  ElseIf xBAM.Ball.Name = TennisBall Then
    xBAM.Ball.Name = VolleyBall
  ElseIf xBAM.Ball.Name = VolleyBall Then
    xBAM.Ball.Name = BeachBall
  End If
```

Flipper Shadows

Flipper Shadows

Flipper Shadows

- Flipper shadows don't require xBAM.CreateAllExt any more. So, all old tables will get ball & flipper shadows without any modifications
- xBAM.DisableFlipperShadows / xBAM.EnableFlipperShadows can be used to disable/enable shadows for selected or all flippers
If you pass as arg name of flipper like this:
xBAM.DisableFlipperShadow "LeftFlipper"

it will disable shadow only for LeftFlipper,
If you skip argument it will be applied to all flippers.
So, for table with many extra-invisible-hidden-flipper, you can hide unwanted shadows.

Ball Shadows

Ball Shadows

Ball Shadows

About ball shadows:
They are not "real" shadows. It is just added dark spot under ball. This method have no impact of performance and works allways. It may create some "errors" but in most cases it looks realy good.

There is no point in adding real shadows calculated for every light position. First, it may work only in new-renderer, second it will have huge impact of performance, third it may not look any better.

New ball shadows are enable by default. You can find options to change it in Addons menu.
- dark level - (default: 0.7) - if you want to disable shadows, set value to 0.0. It determines how dark is shadow.
- radius soft - (default: 1.8) - how far from ball shadow will be drawed [in ball radius units]. This is "soft" edge of shadow.
- radius hard - (default: 0.0) - "hard" edge of shadow. You "hard" = "soft" when whole spot under ball is uniform. If "hard" < "soft", between that 2 edges shadow will be blured.

Please play with that params. Maybe you will finde better default values.

Script:
xBAM.SetBallShadows dark_level, radius_soft, radius_hard, maxLevelAbovePlayfield, removeShadowsFromInvisibleBall

Note, you have 2 more params than in BAM menu.
By default, when ball is above playfield, shadow under ball is smaller. If ball is ~13.5mm above (1 ball radius) shadow will disappear.
- maxLevelAbovePlayfield - (default value = 1.0) - you can increase how far from playfield ball shadow will diappear.
- removeShadowsFromInvisibleBall - (true/false, default true), If you have invisble balls on table (with opacity = 0.0), when ball will don't have shadow. So, in script you can set this to false and even invisible ball will have shadow. This is added in case if someone have wird ball on table with attached miniplayfield and wan't have shadow.

If you skip last 2 params in call to xBAM.SetBallShadows, when default value will be used.

BAM

BAM

BAM

Bam (Better Arcade Mode) is a plug-in for Future Pinball, designed to enhance Future Pinball with additional features and capabilities. BAM has had a huge impact on FP, not only just better physics, but also in other areas like lighting effects and soon, the ability to run COM objects (programs outside of Future Pinball like music engines, .net applications, and other plug-ins that previously were restricted by the FP dev)

The fpxEngine requires the use of BAM, and it is our recommendation that you always use the latest version.

FPX has only the most basic support of BAM, the improved physics, and only the main improvements. The fpxEngine has presets for physics packages (3 so far, based on era,Solid State (1980), Data east (1985) and wpc (1990s), as well as adjustable "flipper bounce" settings. All the settings for the BAM support in the fpxEngine can be found in the Beginners Guide.

At the moment (beta 1.2) there will be no additional BAM support for the foreseeable future. BAM, as good as it is, requires a very steep learning curve, and is not very organized in a manner that makes using it more trial and error than instantaneous. There are a lot of settings and adjustments, some people have been playing with it for months, and you may have to go through ten's of pages just to find the information needed to get even one feature working, or even find out what it does.

It was the decision of the fpxEngine developer (that would be me) that the weeks, if not months of studying, testing, coding, debugging and then playing with BAM would be put to better use in developing fpx. As much as some things in BAM are pretty damn awesome, just don't have that time to learn how to use it.

Additional BAM features will be added if there is enough demand from the people who use fpx for the creation of their own tables, or if another developer with that knowledge adds the additional BAM code.

QUICK NOTES

- Start game in debug mode by hitting the F9 key instead of the play icon, press the Numberpad 5 key to generate a report of the BAM settings used in fpxEngine that is written in thefpDebugTextLog.txt
- fpxSetBounce (in User Input Section) sets the amount of bounce the ball will have off the flippers. There is slower ball setting for the bounce, and a second fast ball setting.

The slow ball settings have less "bounce" than faster balls.

- xPhysics can use 3 xml physics packages. Once more xml files are released, this section will change to have more distinctive physics. 0= No XML(default xml in BAM (NOT TESTED YET)) 1=wpc(1990) 2=Bally(1982) 3=DataEast(1985)
- fpxBallShadows adds a shadow underneath the ball as it is rolling around the playfield. 0=Ball shadows Off:1=BAM Default:2 = custom setting
- fpxBAMpfLighting adjusts the overall lighting of the entire table. (1=bright/2=Medium/3=Dark)
- fpxUseShadowMaps allows user to turn on or off dynamic shadow maps (in case of very slow or under powered computer)
- MaxOmega and MinOmega are the force of the flippers, and how hard they shot the ball.
- xMaxBallSpeed is the maximum setting for Ball Speed. You would set this to lower for EM type games (roughly 2000) while the latest games are a lot faster (around 3500) fpx default as set is 3000

How the table is built

fpx Table Build

How the Table is Built

The fpxEngine includes several templates, with all of the elements built in a simple manner that is amazingly enough, quite simple concepts. The templates were designed, built and placed within the engine with a eye towards making the elements as simple to modify and change in the easiest manner possible. Many authors have their own way of doing things, this way is as easiest as possible as I could think of. This page details how each object is set up with textures, the layer orders etc.

The general "build" of the Base template as well as other things like the Vault is quite detailed, and everything will be already built for you complete with textures and any additional objects such as ornaments, so you really don't have to do anything to have a professional looking, polished table as it's already done for you.

Set Surface Heights

One of the nice things about Future Pinball is the ability to set objects to the top of a surface instead of adding a Height or a Offset to that object. The big advantage to this is when you adjust the height of a surface, any objects that are attached to it will also change their height to reflect the change. So if you adjust the height of a surface such as the top playfield like Black Knight or Flash Gordon, any objects attached, like posts, rubbers, targets, and ornaments will also change their height as well.

fpx has three main surfaces that nearly all objects are attached to.

- **a_PFSurface -** which is the floor of the table. All posts/rubbers/lights/holes etc are attached to a_PFSurface, so if you adjust the height of a_PFSurface, all those objects will be at that height set. This is handy if you wish to have a sunken playfield (like in Black Hole) or a subway system within your game.
- **a_PlasticSurface -** Some FP models require you to set their height above the playfield, such as Gate Wires and Brackets. Usually, this is set to 32mm, or the standard plastic height, but instead, we have a dedicated surface (a_PlasticSurface) for this. a_PlasticSurface is set to 32 height, so most ornaments (such as single screws on top of the header plastics) will rest right on top of the plastics. Some objects may have a "offset" added, that makes that object slightly higher. The Offset are not affected by any changes to the surface height, they just add a higher value to the surface or if the object is attached directly to the playfield.
- **a_Light_Surface -** FP has a bug when you attach a light object to a surface that is set to be a playfield surface. The lights themselves become very transparent, and any graphics attached are not very visible. On the other hand, as long as the "**Surface is a PlayingField**" is unchecked, lights will render fine. a_Light_Surface is raised 1 millimeter so the lights will always be on top of the playfield, and these include bulbs as well as lights. Make sure that all lights and bulbs are included in the LightList Manager (top menu under "Table") so every light will work with LightSeq routines (AddLightFX). All lights and bulbs have custom textures, you can change the textures by selecting the texture in the Texture Manager, reimport and then replace the textures with another texture.

A couple objects require that they are set to a different height. LightCredit is attached to the Apron surface (p_Apron_46h) and the plunger gate bracket is set to the right far wood wall (WoodWall_Right) as it needs to be set at 30 height.

Texture and Model Manager

All the stock textures included with the future pinball install have been included, as well as some custom graphics and models. As time goes on, more custom textures and models will be added as fpx matures, but at the moment, there are just a few new additions. Included with the fpxEngine package is a Resource Folder, these will contain the default fp textures, as well as a complete copy of custom textures that you can use.

Models

Future Pinball uses 3d models for it's objects, in a special format called fpm files. These are actually models done in a program called Milkshape, which is a retail product. Alas, it's not very popular, and FP only allows this one way of doing things, and doesn't support the most popular formats like .obj

Still,over the years, other members have made their own models, including more accurate models to replace some of the stock models,and have put in far better "collision" meshes so FP plays far better than it use to. BAM has added far more capabilities as well, Future Pinball now is a very different editor than what it use to be.

fpxEngine uses some of these models as it's default, such as better flippers and posts, but the entire base FP models are also included for your use. Eventually, fpxEngine will use nothing but custom models,as it allows you to add or replace custom models with other models without having to touch the base fp models.

Textures

All textures used by the fpxEngine templates are custom textures done by me as opposed to the standard textures included within the FP install.There were a couple reasons for this,and not just because they look nicer. A lot of authors don't use the textures for their graphics, instead they use colors instead, especially with plastic and metal posts. If you wish to change the color of a T5 plastic post (as a example) you have to select each post, change the color of that post,and then do it for every post.

With fpx, all objects use textures for the images and colors, and all objects are set to the same neutral color (a white-grey) so all you have to do is change a texture by replacing the texture with another texture in the texture manager. This mean that if you want to replace the color of the T5 post used in the template from blue to red, you can replace the post texture (called a_post in the Texture Manager) with a red texture,and every single post will automatically be updated and change to that red texture.

Layers

On the left hand side of the Future Pinball editor, is a group of numbers from 1 to 0. These are layers, which you can click on and off to hide or make visible any object assigned to that layer. Unfortunately, there are only 10 layers, so it makes some tables (especially the more complex tables) very hard to find certain objects. On top of that, sometimes it's pretty hard to just remember which object is on a certain layer, so it can be a while before you find it. All authors have a certain way to use layers, some group some sections into it's own layer, while most group the most common items like lights into it's own layer, but personally, even I forget where I set my objects to their layers.

fpxEngine uses layers as well. Most authors have their objects set to a certain layer, with ornaments set to one layer, targets set to another layer etc. fpxEngine though does it a different way. The vault system has prebuilt components that you can mix and match (like a jigsaw puzzle), and sometimes you will need to modify or move the various vault items around within your table, so instead, fpxEngine uses the layers BY VAULT TYPE, in which all items within that vault are on one layer. This makes it a lot easier to change the design of the vault item or move that item, all you need to do is to turn off all the other layers. The other main advantage is if you have a group of items all on different layers, it makes it a lot harder to modify, while with everything on that one layer, it becomes far simpler.

The following is a list of the layers used for the main objects used in each vault item.

- **Layer 1** - Drop Targets
- **Layer 2** - Stand-Up Targets
- **Layer 3** - Kickers
- **Layer 4** - Triggers
- **Layer 5**
- **Layer 6**
- **Layer 7**
- **Layer 8**
- **Layer 9** - Flipper and Header areas
- **Layer 0** - fpxEngine objects

About the Vault

The Vault allows the developer to use complete pre-built sections based from actual arcade pinball games within his own table design, with advanced scoring and light routines, within literally minutes. The objects in the Vault sections will match the objects, textures, and models already included within the fpxEngine Template, and can be changed just like the others. In fact, if you change a texture in one of the objects, and then you add in objects from the Vault, those Vault objects if they have the same texture name will also be changed to the new texture as well. Same goes for Models as well.

fpxEngine Presets

fpxEngine Presets

fpxEngine Presets

Because the main engine contains code that only advanced scripters should change, the FPX Template is designed to give you the ability to customize your table in as easy a manner as possible without having to dig through very confusing and easy breakable advanced code

To make it easy, this template is split into 2 main sections, the user coder section, and the main engine code.

The user coder section was written to give access to all the main things you would want to change, like sounds/lights/ display messages, without having to search through thousands of lines of code or accidentally modifying engine code that could break the script.

The main engine code you can leave alone, all of the code you may even want to modify is pointed at the top of this script, and done as simple and as easy to understand way as possible. These are "Hooks" within the main code, and give you access without causing errors (even placing a line of code the wrong order can cause errors, you don't have to

worry about it now, just keep your code here within this subroutine) With everything in one place, and set and called properly from within the engine, all you have to do is add your unique table code (Or just leave it the way it is)
All Display/LightSync/ music is tied into here, and will run off of just one timer, that you can set the interval time for, and then you don't worry about anything else.

From the very start to the end, all the grunt work and engine stuff is done for you, just add anything special you want to it.

Instead of having separate subroutines for each portion, this uses the Case statements. It is a bit tricky sometimes for new coders to understand, but it's actually very easy to use. Each Case corresponds to a section of the script. You just need to add whatever code you want, or just leave it entirely alone.
Most of the code is the same, and any changes here you can make are usually text messages, music, and timer intervals to set the amount of total time you want each case to play before it automatically switches off. It really is the easy way to do things.

YOU DON'T HAVE TO CHANGE ANY OF THIS IF YOU DON'T WANT TO!

These are just presets, if you like it the way it is, leave it alone. If you do want to experiment then this is a great place to do so, as since this is outside the main engine code, you have a safe place to learn without causing errors to the engine (And if you do, let me know)

AddScoringEvent

- AddScoringEvent is the main preset section that handles all in game scoring, from adding multipliers, to Extra Balls, to preset routines for inlanes and targets. This very powerful system allows you to just write (or copy and paste) a couple lines of code in your hit section for each hit object to have fully functional scoring code. This preset also will include all the code for the objects mechanics, kickers will kick the ball out, drop targets will reset etc.

Link

AddMusicSet

- AddMusicSet controls all the sounds, music, lightseq, and display code, all in on place. You will even be able to select from multiple era styles, even by company style, complete with their own custom displays and light show, just by changing one setting. The AddMusicSet even handles the background sound for you automatically, everything is already done for you

Link

AddDisplay

- AddDisplay is the subroutine that controls all the display code, and the effects (such as blinking text or score). Usually called by AddMusicSet, you can also reuse these routines directly in your custom code with the Message(x) line code and then the call to the case setting within AddDisplay (See the Beginner's Guide for the tutorial on adding messages)

Link

AddLightFX

- AddLightFX are the preset lighting routines using the Light Sequencer feature built into Future Pinball.

Some Quick Things You Need to Know

- The TimerSetEvent1 is the main timer used by all the AddEngineEvent case settings. Once this timer stops playing, it usually executes additional code, so I moved it out of view, but more advanced coders may want to run additional code. I used the SetEventNum variable for this, to prevent errors, but I also marked each case so it's pretty easy to follow. This timer points to vital engine code, so if you don't know what you are doing, leave this alone
- Tilt: Just pointing this one out, Case "TILT" in the AddEngineEvent subroutine contains all the code you want to happen when the game is tilted. The engine will automatically handle it. If you want to add your own code (like flippers) this is where you put all the code. I left all the "turn off" code in this case setting, if you remove it, then it's still active even in tilt state. Just add your own code, everything else is automatically handled in the other main subroutines and presets. If You need to add your own tilt check, here's the code:

`code:`
```
If (fpTilted = False) Then             '  IF the table IS NOT tilted THEN only
execute this code *
    ' bunch of code
End If                                ' Okay we are done, so go to the next line of
code *
```

- This template uses the 'Light Sequencer' and preset codes. You need to put all the lights in the "Light List Manager" in the tables menu at the top of the page. Open the manager, select "AllLights" and press edit. You can add and remove lights and bulbs, this is used for the Light Attract mode, tilt etc and is run automatically. You can find more in the FP Manual pages : Table Components/Lights/Light Sequencer and Managers/Light List Managers
- Display Message: Everything is automated, it can be tricky to use because of the math, so it is out of sight to beginners. The display will flash/show/scroll messages instead of the score (i.e. Making a Extra Ball) but you can change the actual message you want. Just keep the message inside the quotes.
- Music Channels: FP has a very powerful music system, and has 8 music channels to use. It is always best to have groups of sounds assigned to a channel, as this helps prevent multiple sounds playing at once, which can be jarring, and makes it far easier to keep things organized. Furthermore, FP has a EffectMusic command, which allows you to fade in or out a channel or pause a channel (which we use for the background sounds) The template uses 3 channels: Channel 2 - Is used for the Background music. Channel 5 - Is used for the "hit event" music, music that plays when a object is made during game play. Channel 6 - Is used and reserved for the actual engine, like the game start up sounds, Match etc. PlaySound - anything that uses the PlaySound command instead of the PlayMusic command are the "mechanical" sounds.
- AddEngineEvent These are noted as these are sections you will need for your code. Case "INITIALIZE" - table start up, here you put in your opening table variables. Case "NEW_BALL" - This is where you add your code for the start of each new ball. No need to dip in the engine, just put your routines like resetting target banks, clearing variables etc right here. Case "TILT" - If you don't use the above hit event method, or

need to turn off flippers etc, this is the place to add your code. Does Match, then restarts LightAttract mode at GAME OVER display

- The AddEngineEvent timer (TimerSetEvent1) has been moved to be part of the main engine, and should not be modified unless you really really know what you are doing
- The background music timer has also been moved to the engine code from the user modification system
- A new timer has been created (TimerCloseScoringEvent) to give users a simple single timer if they need it for the hit_code (e.g. pop targets delay time before they pop back up) More code will be added in the future, this was used in my Jungle Girl table, and proved to be quite useful. When a music file has finished playing, the engine then points to the background music timer (to play the background sound). The background sound system then points to TimerCloseScoringEvent to run any closing statements. The very nice thing about this is you don't even have to code a timer interval, it's already set by MusicIntervalTime in AddMusicSet.
- BG Sounds: The template can play Background sounds, but we have to do extra coding to pause the background music so we can play other music before we unpause the BG Music to continue playing. This prevents the Background music from restarting at the very begining every time we want a new sound. We use the "EffectMusic" command for this. We point to the EffectMusic Command to fade out and pause the BG sound, we add a interval delay to match the replacement music length and then call the timer to unpause the Background music to resume playing

```
code: If you want to change the BG sound, search
      for "TimerBackgroundMusic_Expired()"
      StopMusic 2:StopMusic 5:StopMusic 6     ' Stop any
      sounds from playing. This is a error catcher more
      than anything
       EffectMusic 2, FadeOutAndPause,0, 100    ' We need
      to pause the BG if it is playing for player added
      sound. Channel 2 is used the background music
      EffectMusic 6, PlayAndFadeIn, 1, 100     ' This fades
      in the player added sound. Note the music channel is
      6, which is the channel used for the engine music
      If PlayBackgroundMusic = 1 then     ' only if you
      want background music to play as set. If this is set
      to 0, then no background music will play at all
      ' Unpause BG at the interval time set for the timer.
      This is usually the total music time of that file in
      milliseconds you want to play
      TimerBackgroundMusic.Interval=2500:TimerBackgroundMu
      sic.Enabled=TRUE
      End If
      PlayMusic 6, "Bally81_new_ball",, fpxMaxMusicVolume
      ' Play that music file.
```

AddEngineEvent

AddEngineEvent
fpxEngine Presets »

AddEngineEvent

Writing a engine script for pinball (either for FP or VP) is one of the hardest things to do. It requires a lot of experience with code to be able to pull it off, and can take months of writing and testing before even a basic template engine is ready to be released. The biggest problem though is for the vast majority of people, it is actually using that template, as the concepts and knowledge needed to understand and use that template is way beyond the capabilities of about 99% of the worlds population. Coding something is really hard. Understanding what the code does when you don't know any coding at all is even harder.

This has always been the biggest thing with templates, or even making your own table. You have to learn how to code, and most people just don't have that time to learn. Even intermediate coders have problems dealing with a engine code, especially on where they can add or modify code. A decent engine can be in the thousands of lines of code, and it is almost impossible to figure out where to place a small piece of code within the engine without causing errors and hours and hours of debugging because of that.

The AddEngineEvent subroutine allows you to add your own code to the main engine in a way that is totally different than any other template before it. Instead of making you go through thousands of lines of code, instead, fpxEngine uses the concepts of "hooks". Within one subroutine, you will have complete access to all parts of the main engine core, without having to even look at that core if you don't want to.

Within the main engine code are the hooks that point to AddEngineEvent, safe places for you to add your code like what you want a light to do at the start of a new ball, or if you wish to turn off a bumper if the game is tilted. There are many sections within a engine, from the very start when a table is loaded in, the light attract mode, and all the various parts of a game being played, from when a coin is inserted, to the game is finished. AddEngineEvent contains hooks for all these parts, so you can add your own code or modify the default code to your tastes.

As the engine matures, this section will contain any updates, and also have examples of additional code needed for future more advanced features. In the meanwhile, the Beginners guide has a section just on AddEngineEvent, that explains everything in detail if you wish to make any changes (though it's optional)

AddKeyEvent

AddKeyEvent
fpxEngine Presets »

AddKeyEvent

AddKeyEvent is a subroutine that allows you to add custom code or additional features (such as a third flipper) that requires a keypress to use or activate. Like AddEngineEvent, theses are "hooks" that point to the main key press routines, safe places for you to add your code. Though the main engine routines look relatively simple, they can be very hard for beginners as proper placement of any custom code can be quite tricky.

AddKeyEvent uses the keys that are defined in the Preferences menu in the editor by the user, as opposed to calling the keys directly. By default FP sets the *SHIFT* keys for the flippers, the *ENTER* key for the plunger. the *A* and the ' keys for the special keys (basically a second set of flipper buttons, like the magne save on Black Knight) and the *SPACEBAR*, / and *Z* keys for nudging. By using the default key code used by FP, a user can change the keys and it will still be recognized and used by fpx.

BAM has it's own set of keys, unfortunately though, BAM has set as it's defaults, some of the reserved keys used by FP, so you will lose some functionality. The P key which was the pause key in FP is now the key to change the texture of the ball, and the tithe key which is the default key for displaying or hiding the HUD display in desktop instead calls the BAM menu (as well as the Q key, which is rather puzzling)

As having the HUD being able to display or fade out (if you use a cabinet) is very important, the HUD key is coded in fpx to be the H key.

FP has a manual (help right at the top when clicked will open up the manual) that explains the keys. You can find the portion that describes
how to change the key by clicking the Preferences chapter on the left of the manual (almost at the bottom) and then Game Keys and Controls

AddKeyEvent has code hooks for the flippers and special keys (pressed and released as you have to tell the script to move the flipper up or down), additional code for the HUD (which is unused at the moment) and also for a coin in (in case you wish to add something when a player "inserts" a coin). Everything else would be handled by AddEngineEvent (like game start or match routines)

At the moment, AddKeyEvent doesn't have a lot within it, it is very basic, though as the engine matures and with more releases to come, you can expect more key press code to be added.

The routine

```
code:  Sub AddKeyEvent(KeyEventVariable)
        Select Case KeyEventVariable
         case "LEFTFLIPPER-PRESSED"
```

```
            LeftFlipper.SolenoidOn
           PlaySound "FlipperUp",(fpxSoundVolume)
         case "LEFTFLIPPER-RELEASED"
            LeftFlipper.SolenoidOff
           PlaySound "FlipperDown",(fpxSoundVolume)
         case "RIGHTFLIPPER-PRESSED"
            RightFlipper.SolenoidOn
           PlaySound "FlipperUp",(fpxSoundVolume)
         case "RIGHTFLIPPER-RELEASED"
            RightFlipper.SolenoidOff
           PlaySound "FlipperDown",(fpxSoundVolume)
         case "SPECIAL1-PRESSED"
         case "SPECIAL1-RELEASED"
         case "SPECIAL2-PRESSED"
         case "SPECIAL2-RELEASED"
         case "HUDDISPLAY-SHOWING"
         case "HUDDISPLAY-HIDDEN"
         case "COININ"
        End Select
       End Sub
```

As you can see, only the Left and Right Flippers are used. The flippers are part of fpxEngine, but were placed here for easy modification. The fp manual explains the flippers in far greater detail, but briefly, SolenoidOn means the flipper is moving up or is fully up when you press and hold a flipper key, and SolenoidOff means the flipper is moving down or is fully down when you release that flipper key.

The PlaySound line is for the mechanical sound of the solenoid that the player hears when he presses or releases a flipper key.

The other settings are blank. These are very handy though for testing while you are developing your own game. You can add debug code to test features or other things as you wish (just make sure you remove that code if you decide to share your table with others)

AddMusicSet

AddMusicSet

fpxEngine Presets »

AddMusicSet

AddMusicSet is the main routine that controls the display, Sound/Music, and lighting routines. These are a series of presets that are run automatically, and there is a separate preset for each part of the engine. If you remove the main code for this, there will be no effects or sounds, and the engine may "break". You should leave the main AddMusicSet subroutine alone. If you want to use your own custom code, you can by just typing your own music, display and lighting routines directly in your hit code, and not use the AddMusicSet routines at all.

AddMusicSet though is a very powerful advanced system that gives amazing flexibility to the engine just by changing one line to point to a different music set. The user can select from one of the main music sets (such as from Bally or Williams games) and have fully coded music and display routines running within his game automatically. Not only are there different music sets, but also some routines have completely different display routines and lighting effects from than from other music sets. It is also possible to even have completely different rules for a feature (such as a target bank for example) depending on which music set is being used. In other words, even if the playfield design is the same, you can have completely different ways to play that design, different music, even a completely different set of rules. If you ever wondered for example how the Williams game "firepower" would have been like if it was a Bally game instead of a Williams game, now you can find out, just by changing the MusicSet line in the User Input Section from "w79" to 'Bally81". Down the road, you may even be able to change the physics so you can play a EM style game, and if you want, change it to a Solid State game, complete with different physics, scoring routines and flashing effects (Just like Bally did with Fireball, updated the classic EM game to a 1980's game)

Note: there is no custom coding of different rules per rule set in the default fpx code, this would have to be coded in by the developer of the table. A future update of this manual will deal in detail with how to code in different rules/displays/lights etc.

If you want a different set of rules for each music set, all you would need to do is have your Hit code look at which music set is selected, and run a set of rules accordingly.

```
code: If MusicSet="Bally81_" then
        ' code for rules for bally games
        AddScoringEvent "AddMultiplier"
      Else
        ' code for the other musicsets
          AddScoringEvent "Mystery"
      End If
```

Changing Music Sets

Music sets can be changed just by modifying one line in the User Input Section by changing one prefix to match the beginning of the file name to point to the proper set. The engine can support a unlimited amount of music sets, fpx has included 3 music sets based on era and company.

Each music set also contains preset pointers to the display for special effects like scrolling text, and also special routines for the lights using the LightSeq feature within Future pinball.

A Note To Coders

You can also use the code within your hit event as well, so you can change music sets right in game. In fact, if a player gets bored with one music set, he can just change it to a completely different one any time he wishes. All supported music sets within fpx can be used by any table at anytime, so you can make a more modern table sound like a em table.

If you wish to use a different music set, then comment out the active line and uncomment the line you wish to use. In this example, wsys7 is active, just put a apostrophe (') in the front to turn the line to a remark, and remove the apostrophe from one of the other lines. Setting the MusicSet to "off" means no music/display text/or lightseg. This is for coders who wish to put in their own custom code or use their own combination of presets.

```
code:          ' MusicSet="Off_"       ' This switches OFF ALL
      MUSIC/DISPLAY/LIGHTSEQ routines so you can code all
      that yourself
               ' MusicSet="Bally81_"     ' Set the
      music/lightseq/display for Bally 1982 era tables
               ' MusicSet="w79_"     ' Williams system 6 era

               MusicSet="wsys7_" ' Williams System 7 era
```

AddDisplay

AddDisplay

fpxEngine Presets »

AddDisplay

AddDisplay is the subroutine that handles all display text messages and adds special routines for scrolling, blinking or flashing text. Generally used by AddMusicSet and the main engine, AddDisplay was put in it's own subroutine (as a Select Case) so you can "mix and match" display code with the other main routines. You can also directly call AddDisplay in your custom code for one of the preset display routines.

This allows you to display messages instead of the players score in each of the 4 displays. It's also used a lot by the engine, you will notice the `Message(x)` code lines in a lot of areas, from table power up to the match routines. AddEngineEvent gives you access so you can place your code directly into the engine, but it also gives you some things you can change to suit you.

Message(x) are the text messages that are displayed in the display when you are playing a game. There are 4 displays, so there are 4 messages you have to use. (and the 4 HUD displays as well that match the displays in the translight) So, Message(1)=Player1 display, Message(2)=player 2 display and so on.

Some of the AddScoringEvent presets uses just 2 messages, and add a scoring value to the 2 bottom displays (like the Mystery Feature or displaying a bonus Multiplier), while others will use all 4 displays

The things you need to remember

- Only change the text message that is between the quotes ("). Do not delete or change anything else. Because there are 9 digits in each display, your words should be no more than 9 characters long. Any lowercase letters you add will show when you are playing a game, but as upper case letters, This is normal for non-dmd games.
- Even though you have 4 displays, each display can only display 9 characters at any time.
- Some areas you only have 1 message (like the tilt) to do, others 2 messages, but most are 4 messages.
- Sometimes, Message(3) and Message(4) will have quotes with nothing between them.This means that Display 3 and Display 4 will show completely blank while in game. You can add messages if you want, I just like how it looks.

List of AddDisplay preset Routines

code:
```
' --------------------- Display settings
--------------------------------
Dim AddDisplayCase ' Case settings for display
system
Dim  FlashForDisplayInterval ' interval time for
flashing display
Dim  DisplayQueueTextInterval ' time for each
QueueText (makes it easy to change one number
instead of a hundred)
FlashForDisplayInterval=50  ' Default-Interval time
for Display blinking,UpdateInterval property is set
to half this value automatically in AddDisplay Sub.
Sub AddDisplay (AddDisplayCase)
  ' Handles display effects
   IF constAddDebug = 2 THEN AddDebugText
"AddDisplay - " & (AddDisplayCase)
 Select Case AddDisplayCase
 ' Blanket code to display players score
  Case "DisplayReloadScore"
   FOR x = 1 to PlayersPlayingGame:Seg(x).Text
= nvScore(x):HSeg(x).Text = nvScore(x):NEXT
 ' Blinks Display at medium speed player up display
  Case "BlinkScoreCurrentPlayer"
   FOR x = 1 TO SegCount
    Seg(x).SlowBlinkSpeed =50:HSeg(x).SlowBlinkSpeed
=50
    Seg(CurrentPlayer).QueueText
nvScore(CurrentPlayer), seBlink,
(MusicIntervalTime), 0, TRUE, ""
```

```
     HSeg(CurrentPlayer).QueueText
nvScore(CurrentPlayer), seBlink,
(MusicIntervalTime), 0, TRUE, ""
   NEXT
  Case "BlinkPlayerUpScore"
   FOR x = 1 to SegCount
     Seg(x).UpdateInterval  = 10:Seg(x).SlowBlinkSpeed
=100:HSeg(x).UpdateInterval
= 10:HSeg(x).SlowBlinkSpeed  =100
   NEXT
   FOR x = 1 to PlayersPlayingGame:Seg(x).Text
= nvScore(x):HSeg(x).Text = nvScore(x):NEXT
    Seg(CurrentPlayer).QueueText
nvScore
(CurrentPlayer), seBlink, 999999, 0, FALSE, ""
    HSeg(CurrentPlayer).QueueText
nvScore
(CurrentPlayer), seBlink, 999999, 0, FALSE, ""
    AddLightFX "Stop"
 ' These are variable speeds for blinking, as set in
the AddScoringEvent to match the blinking display to
the bulbs/lens interval blinking speed
  Case "BlinkScoreCurrentPlayerVary"
   FOR x = 1 TO SegCount
     Seg(x).UpdateInterval  = 10:Seg(x).SlowBlinkSpeed
=(DisplayBlinkInterval):HSeg(x).UpdateInterval
= 10:HSeg(x).SlowBlinkSpeed  =(DisplayBlinkInterval)
     Seg(CurrentPlayer).QueueText
nvScore(CurrentPlayer), seBlink,
(MusicIntervalTime), 0, TRUE, ""
     HSeg(CurrentPlayer).QueueText
nvScore(CurrentPlayer), seBlink,
(MusicIntervalTime), 0, TRUE, ""
   NEXT
  Case "BlinkPlayerUpScoreVary"
   FOR x = 1 to SegCount
     Seg(x).UpdateInterval  = 10:Seg(x).SlowBlinkSpeed
=(DisplayBlinkInterval):HSeg(x).UpdateInterval
= 10:HSeg(x).SlowBlinkSpeed  =(DisplayBlinkInterval)
   NEXT
    Seg(CurrentPlayer).QueueText
nvScore
(CurrentPlayer), seBlink, 999999, 0, FALSE, ""
    HSeg(CurrentPlayer).QueueText
nvScore
(CurrentPlayer), seBlink, 999999, 0, FALSE, ""
    AddLightFX "Stop"
 ' matching score display effects after a message
event
  Case "DisplayScoreScrollLeft"
 'Scrolls the score
```

```
     FlushDisplay()
    FOR x = 1 to PlayersPlayingGame
      Seg(x).QueueText
nvScore
(
x), seScrollLeft, 2000,0, TRUE, "":HSeg(x).QueueText
nvScore(x), seScrollLeft, 2000,0, TRUE, ""
    NEXT
    IF  constAddDebug >1 THEN AddDebugText
"AddDisplay  DisplayScoreScrollLeft"
 ' Preset blinking speeds
  Case "BlinkMessageSlow"
   FOR x = 1 TO SegCount
     Seg(x).SlowBlinkSpeed
=(FlashForMSBlinkInterval*3):HSeg(x).SlowBlinkSpeed
=(FlashForMSBlinkInterval*3)
     Seg(x).QueueText
Message
(x), seBlink, (MusicIntervalTime), 0, TRUE, ""
     HSeg(x).QueueText
Message
(x), seBlink, (MusicIntervalTime), 0, TRUE, ""
   NEXT
  Case "BlinkMessageMedium"
   FOR x = 1 TO SegCount
     Seg(x).SlowBlinkSpeed
=(FlashForMSBlinkInterval*2):HSeg(x).SlowBlinkSpeed
=(FlashForMSBlinkInterval*2)
     Seg(x).QueueText
Message
(x) & "", seBlink, (MusicIntervalTime), 0, TRUE, ""
     HSeg(x).QueueText
Message
(x) & "", seBlink, (MusicIntervalTime), 0, TRUE, ""
   NEXT
  Case "COININ"
    FOR x = 1 to SegCount:Seg(x).UpdateInterval
= 50:HSeg(x).UpdateInterval
= 50:Seg(x).SlowBlinkSpeed
= 100:HSeg(x).SlowBlinkSpeed
= 100:Seg(x).Text="":HSeg(x).Text="":NEXT
     Seg(1).QueueText
(Message
(
1))

&
 ""
, seBlink
, (MusicIntervalTime), 0, TRUE, "":Seg(2).QueueText
(Message
(2)) & "", seBlink, (MusicIntervalTime), 0, TRUE, ""
    HSeg(1).QueueText
(Message
(
1))

&
 ""
, seBlink
, (MusicIntervalTime), 0, TRUE, "":HSeg(2).QueueText
(Message
(2)) & "", seBlink, (MusicIntervalTime), 0, TRUE, ""
    Seg(3).QueueText
(Message
(
3))

&
 ""


, seNone
, (MusicIntervalTime),0, TRUE, "":Seg(4).QueueText
(Message
(4)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
    HSeg(3).QueueText
(Message
(
3))

&
 ""


, seNone
, (MusicIntervalTime),0, TRUE, "":HSeg(4).QueueText
(Message
(4)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
  Case "BlinkMessageFast"
     FlushDisplay()
   FOR x = 1 TO SegCount
    Seg(x).SlowBlinkSpeed
=(FlashForMSBlinkInterval):HSeg(x).SlowBlinkSpeed
=(FlashForMSBlinkInterval)
    Seg(x).QueueText
Message
(x) & "", seBlink, (MusicIntervalTime), 0, TRUE, ""
    HSeg(x).QueueText
Message
(x) & "", seBlink, (MusicIntervalTime), 0, TRUE, ""
     AddLightFX "BlinkMedium"
   NEXT
  Case "Display2MessageAlternateBlink"
  FOR x = 1 TO SegCount
```

```
    Seg(x).UpdateInterval
= (FlashForDisplayInterval/4)
    HSeg(x).UpdateInterval
= (FlashForDisplayInterval/4)
  NEXT
  For x=1 to 5                                   '
flashs the display 10 times
    Seg(1).QueueText
(Message
(
1))

& "" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    Seg(2).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    Seg(3).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    Seg(4).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    HSeg(1).QueueText
(Message
(
1))

& "" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    HSeg(2).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    HSeg(3).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    HSeg(4).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    Seg(1).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    Seg(2).QueueText
(Message
(
2))

& "" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    Seg(3).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    Seg(4).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    HSeg(1).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    HSeg(2).QueueText
(Message
(
2))

& "" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    HSeg(3).QueueText
```

```
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
    HSeg(4).QueueText
"" , seNone, (FlashForDisplayInterval),0, TRUE, ""
  NEXT
   ' then displays solid display till music is
finished before it resets to player scores
   Seg(1).QueueText
(Message
(1)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
   Seg(2).QueueText
(Message
(2)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
   Seg(3).QueueText
(Message
(3)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
   Seg(4).QueueText
(Message
(4)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
    HSeg(1).QueueText
(Message
(1)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
    HSeg(2).QueueText
(Message
(2)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
    HSeg(3).QueueText
(Message
(3)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
    HSeg(4).QueueText
(Message
(4)) & "" , seNone, (MusicIntervalTime),0, TRUE, ""
  Case "Display2MessageBlink"
   FOR x = 1 to SegCount
    Seg(x).SlowBlinkSpeed  =50:HSeg(x).SlowBlinkSpeed
=50
    Seg(x).QueueText
(Message(1)) & "":HSeg(x).QueueText
(Message(1)) & ""
    Seg(1).QueueText
(Message(1)) & "", seBlink, 2000, 0, FALSE, ""
    Seg(2).QueueText
(Message(2)) & "", seBlink, 2000,0, FALSE, ""
    Seg(3).QueueText
(Message(1)) & "", seBlink, 2000, 0, FALSE, ""
    Seg(4).QueueText
(Message(2)) & "", seBlink, 2000, 0, FALSE, ""
    HSeg(1).QueueText
(Message(1)) & "", seBlink, 2000, 0, FALSE, ""
    HSeg(2).QueueText
(Message(2)) & "", seBlink, 2000,0, FALSE, ""
    HSeg(3).QueueText
(Message(1)) & "", seBlink, 2000, 0, FALSE, ""
```

```
     HSeg(4).QueueText
(Message(2)) & "", seBlink, 2000,0, FALSE, ""
   NEXT
 ' Scrolling
  Case "DisplayMessageScrollLeft"
 'Adds 4 Message, then scrolls it out
   FOR x = 1 To SegCount: Seg(x).Text = "
":HSeg(x).Text = " ": NEXT
    Seg(1).QueueText
(Message
(
1))

&
 ""
 , seScrollLeft, (MusicIntervalTime/2),0, TRUE, ""
    Seg(2).QueueText
(Message
(
2))

&
 ""
 , seScrollLeft, (MusicIntervalTime/2),0, TRUE, ""
    Seg(3).QueueText
(Message
(
3))

&
 ""
 , seScrollLeft, (MusicIntervalTime/2),0, TRUE, ""
    Seg(4).QueueText
(Message
(
4))

&
 ""
 , seScrollLeft, (MusicIntervalTime/2),0, TRUE, ""
     HSeg(1).QueueText
(Message
(
1))

&
 ""
 , seScrollLeft, (MusicIntervalTime/2),0, TRUE, ""
     HSeg(2).QueueText
(Message
(
2))
```

```
&
 ""
 , seScrollLeft, (MusicIntervalTime/2),0, TRUE, ""
    HSeg(3).QueueText
(Message
(
3))

&
 ""
 , seScrollLeft, (MusicIntervalTime/2),0, TRUE, ""
    HSeg(4).QueueText
(Message
(
4))

&
 ""
 , seScrollLeft, (MusicIntervalTime/2),0, TRUE, ""
   For x=1 TO 4:Seg(x).QueueText
""

, seScrollLeft
,
 (MusicIntervalTime/2),0, TRUE, "":HSeg(x).QueueText
""

, seScrollLeft
, (MusicIntervalTime/2),0, TRUE, "":NEXT
 ' Radar
  Case "Radar"
   FOR x = 1 To SegCount: Seg(x).Text = "
":HSeg(x).Text = " ": NEXT
   Seg(1).QueueText
(Message
(
1))

&
 ""


, seWipeRadarLeft, (MusicIntervalTime/2),0, TRUE, ""
   Seg(2).QueueText
(Message
(
2))

&
 ""


, seWipeRadarLeft, (MusicIntervalTime/2),0, TRUE, ""
```

```
    Seg(3).QueueText
(Message
(
3))

&
 ""


, seWipeRadarRight
, (MusicIntervalTime/2),0, TRUE, ""
    Seg(4).QueueText
(Message
(
4))

&
 ""


, seWipeRadarRight
, (MusicIntervalTime/2),0, TRUE, ""
    HSeg(1).QueueText
(Message
(
1))

&
 ""


, seWipeRadarRight
, (MusicIntervalTime/2),0, TRUE, ""
    HSeg(2).QueueText
(Message
(
2))

&
 ""


, seWipeRadarRight
, (MusicIntervalTime/2),0, TRUE, ""
    HSeg(3).QueueText
(Message
(
3))

&
 ""


, seWipeRadarLeft, (MusicIntervalTime/2),0, TRUE, ""
    HSeg(4).QueueText
(Message
(
4))

&
 ""


, seWipeRadarLeft, (MusicIntervalTime/2),0, TRUE, ""
   Seg(1).QueueText
(Message
(
1))

&
 ""


, seWipeRadarRight
, (MusicIntervalTime/2),0, TRUE, ""
   Seg(2).QueueText
(Message
(
2))

&
 ""


, seWipeRadarRight
, (MusicIntervalTime/2),0, TRUE, ""
   Seg(3).QueueText
(Message
(
3))

&
 ""


, seWipeRadarLeft, (MusicIntervalTime/2),0, TRUE, ""
   Seg(4).QueueText
(Message
(
4))

&
 ""


, seWipeRadarLeft, (MusicIntervalTime/2),0, TRUE, ""
   HSeg(1).QueueText
(Message
(
1))

&
 ""
```

```
, seWipeRadarLeft, (MusicIntervalTime/2),0, TRUE, ""
    HSeg(2).QueueText
(Message
(
2))

&
 ""

, seWipeRadarLeft, (MusicIntervalTime/2),0, TRUE, ""
    HSeg(3).QueueText
(Message
(
3))

&
 ""

, seWipeRadarRight
, (MusicIntervalTime/2),0, TRUE, ""
    HSeg(4).QueueText
(Message
(
4))

&
 ""

, seWipeRadarRight
, (MusicIntervalTime/2),0, TRUE, ""
  Case "DisplayMaxExtraBallAlternateScore"
   FOR x = 1 to SegCount
     Seg(x).UpdateInterval  = 50:Seg(x).SlowBlinkSpeed
=100
     HSeg(x).UpdateInterval
= 50:HSeg(x).SlowBlinkSpeed  =100
     Seg(x).QueueText
(fpxMaxExtraBallAlternateScore), seBlink
, (MusicIntervalTime), 0, FALSE, ""
     HSeg(x).QueueText
(fpxMaxExtraBallAlternateScore), seBlink
, (MusicIntervalTime), 0, FALSE, ""
   NEXT
  Case "DisplayMatch"
   FOR x = 1 to SegCount:Seg(x).Text="
":HSeg(x).Text=" ":NEXT
    FOR x = 1 to PlayersPlayingGame
      Seg(x).Text = nvScore(x):HSeg(x).Text
= nvScore(x)                 ' Reload scoring values
for all 4 player scores
    NEXT
  Case "DisplaynvScore"

 Seg
(
1).SetValue
(nvScore1): Seg
(
2).SetValue
(nvScore2): Seg
(3).SetValue(nvScore3): Seg(4).SetValue(nvScore4)

 HSeg
(
1).SetValue
(nvScore1): HSeg
(
2).SetValue
(nvScore2): HSeg
(3).SetValue(nvScore3): HSeg(4).SetValue(nvScore4)
  Case "DisplaynvHighScore"

 Seg
(
1).SetValue
(nvHighScore1):Seg
(
2).SetValue
(nvHighScore2):Seg
(
3).SetValue
(nvHighScore3):Seg(4).SetValue(nvHighScore4)

 HSeg
(
1).SetValue
(nvHighScore1):HSeg
(
2).SetValue
(nvHighScore2):HSeg
(
3).SetValue
(nvHighScore3):HSeg(4).SetValue(nvHighScore4)
  Case "DisplayGameOver"
   FOR x = 1 To SegCount: Seg(x).Text = "
":HSeg(x).Text = " ": NEXT
   Seg(1).Text = "   GAME   ": Seg(2).Text = "   OVER
 ":HSeg(1).Text = "   GAME   ":HSeg(2).Text = "   OVER
 "

 End Select
End Sub
```

AddLightFX

AddLightFX

fpxEngine Presets »

AddLightFX

AddLightFX are the code using the LightSeq feature within Future Pinball. These are preset lighting routines used that are called by AddMusicSet for a light show while music is playing complete with display effects. In future, other code like from BAM will also be added in, but at the moment, this is pure LightSeq code. Since AddMusicSet also contains the timer interval to play the music and display routine, this time is also used to run AddLightFX for that time as well. fpx will automatically when the timer interval is finished, stop the lightseq and display effects, reset the displays back to show the scoring, and then fade back in the background music.

AddLightFX can be used in custom code, if the developer wishes to bypass the AddMusicEvent system. At the present moment though, there are only a few routines for lightseq built within the engine. Most of the more complex light effects (including multiple routines) will be included with the Vault routines, and then included in AddLightFX for use with other tables.

List of preset light routines

- Stop
- AllLightsOff
- BlinkSlow
- BlinkMedium
- BlinkFast
- DownAndCircle
- CircleInOn
- Random
- RandomFast
- Random2second

AddScoringEvent

AddScoringEvent

AddScoringEvent

AddScoringEvent are the preset feature scoring routines that you can point to within your hit code, like extra ball, or the Mystery Award. The engine will react differently depending on which music set you select, as the music/display/light effects are different from music set to music set.

A Note To Coders

AddScoringEvent will point to AddMusicSet to determine which company music set will play. From there, AddMusicSet will select a display (blinking, scrolling, other fx) and also a lighting effect.

Using the AddScoringEvent is pretty simple, you can just copy and paste the code provided with in a subroutine directly into your hit code. Some Events require messages to be input, (which you learned how to do in the Hit Code Section in the Beginners Guide) followed by the main code line that points to the preset routines. All the main "features" are here, and it is all set up the exact same way, so it's a pretty simple way to add complex scoring within your game.

If You Have Never Coded Before

Just a reminder, to use the code for the features, that code should be encased within a subroutine

```
code: Sub fpxMyTrigger_Hit
        'This is your code....
      End Sub
```

This section details each feature using AddScoringEvent, explains what they do, and (for brave coders) gives a list of the main variables, engine subroutines and the code within fpxEngine. Everything related to these features will be placed in their page, including (in future versions) alternate code examples, additional commands you can use, and sections on modifying the code to do more advanced routines. As well, with future releases of fpxEngine, new scoring features, such as multiball or new features written for the vault will be added.

Contents

- [Extra Ball]
- [Mystery]
- [Outlanes]
- [Special]
- [SlingShots]
- [AddMultiplier]
- [Inlanes]

AddScore

AddScore
AddScoringEvent »

AddScore

Adds points to the player's score
The points you wish to give to the player should be enclosed within 2 brackets. This example gives 100 points.

code:
```
Sub fpxMyTrigger_Hit
   AddScore (100) ' Adds 100 points to the players
score
End Sub
```

You can use a variable to add points as well, such as a randomly generated number or a increasing advance value. This way, you can define different points depending on the game conditions

code:
```
Dim MyAddedScore
MyAddedScore=1000
Sub fpxMyTrigger_Hit
   AddScore (MyAddedScore) ' Adds points value from
MyAddedScore to the players score
End Sub
```

You can also use math expressions within your code. This example adds 100 points and MyAddScore together

code:
```
Dim MyAddedScore
MyAddedScore=1000
Sub fpxMyTrigger_Hit
   AddScore (100 + MyAddedScore) ' Adds points value
plus MyAddedScore to the players score
End Sub
```

You can add, subtract, times, divide etc. You can also use 2 variables as well.

code:
```
Dim MyAddedScore
MyAddedScoreMultiplier=2
MyAddedScore=1000
Sub fpxMyTrigger_Hit
  AddScore ( MyAddedScore
* MyAddedScoreMultiplier) ' Adds points value times
MyAddedScoreMultiplier to the players score
End Sub
```

⚙ Engine Code

List of main variables:

- `Dim Points` - Awards the amount written in the AddScore(x) statement to the players total score, where (x) is a numeric value..

List of main Engine Subroutines:

- `AddScore(points)` - Main scoring routine. This adds the points to the current player's score, updates the player's display, and then checks for any replays if the score is high enough for a free game. Note this routine uses FP's stock variables, nvScore and CurrentPlayer
- `ReplaySpecial()` - Called after AddScore(), this checks if the player's score goes over the replay scores as set by the user in fpxReplay1 and fpxReplay2 (settings are at the top of the script in the user input section) and awards a free game.

code:
```
Sub AddScore(points)
 If (fpTilted = False) and (GameIsStarted=1) Then

nvScore
(CurrentPlayer) = nvScore(CurrentPlayer) + points
                ' add the points to the current
players score variable
  If LockDisplay=0 then
 ' This prevents any scoring on the display or sound
during a Timed Event (AddEngineEvent sub)
   For x = 1 to PlayersPlayingGame
      ' Goto main display score routine
(DisplayScore())
     Seg(x).Text = nvScore(x):HSeg(x).Text
= nvScore(x):
   Next
     ReplaySpecial()                          '
Check if you made a replay or High Score
  End If
 End If
End Sub

' This handles if player makes a replay goal
Sub ReplaySpecial()
 If (fpTilted = False) Then
  If nvScore(CurrentPlayer) >= fpxReplay2 and
ReplayToReach(CurrentPlayer) =1 Then         ' any
replay wins? Note the order of which replay we check
first

 ReplayToReach
(CurrentPlayer) = ReplayToReach(CurrentPlayer) +1

   If nvCredits < fpxMaxCredits Then
     fpxSpecial_Hit()
    IF constAddDebug = 1 THEN AddDebugText
"ReplaySpecial() - ReplayToReach(CurrentPlayer) =2"
   End If
  End If
  If nvScore(CurrentPlayer) >= fpxReplay1 and
ReplayToReach(CurrentPlayer) =0 Then

 ReplayToReach
(CurrentPlayer) = ReplayToReach(CurrentPlayer) +1

   If nvCredits < fpxMaxCredits Then
     fpxSpecial_Hit()
    IF constAddDebug = 1 THEN AddDebugText
"ReplaySpecial() - ReplayToReach(CurrentPlayer) =1"
   End If
  End If
  If ReplayToReach(CurrentPlayer) => 3 Then
     ReplayToReach(CurrentPlayer) = 3
   IF constAddDebug = 1 THEN AddDebugText
"ReplaySpecial() - ReplayToReach(CurrentPlayer) =3"
  End If
 End If
End Sub
```

BallSaver

BallSaver

AddScoringEvent »

BallSaver

The Ball Saver feature is a timed event that starts when the start of the player's ball is in play from the plunger lane. When the player first shoots the ball with the plunger, it will roll over the PlungerLaneTrigger, which automatically will start ballsaver. The BallSaver light will turn on, and if the ball is drained and the light is on, the ball is "saved" and a new ball will appear at the plunger with no loss to the players ball. The BallSaver will then switch off.

PlungerLaneTrigger will only award BallSaver once per ball, any other time the ball rolls over that trigger will be ignored, so if the ball is saved, you can not light the ballSaver again from the plunger area.

If you wish to turn on the Ballsaver as a special scoring feature, you can relight the Ballsaver this code.

If You Have Never Coded Before

Just a reminder, to use this feature, the code should be encased within a subroutine

```
code: Sub fpxMyTrigger_Hit
         'This is where your code goes....
      End Sub
```

```
code:  AddScoringEvent  "BallSaver"
```

Settings

```
code:  fpxBallSaverTime  = 10000
```

The settings for this feature can be found in the User Input Section at the top of the script. This sets the time for the ball Saver feature to be active when a new ball is fired from the plunger and enters the playfield to start play. By default, this is set to 10000 milliseconds or 10 seconds (1000 milliseconds = 1 second, 2500 milliseconds = 2.5 seconds) If you put in "0", the BallSaver feature is switched off. The setting at the top of the script in the User Input Section

Changing the Display Message

Because this feature is used through out the main engine, the message code and any other code is in the AddEngineEvent subroutine. Just search for that subroutine (Sub AddEngineEvent) and scroll down till you see the case setting and make your changes within the quotes.

```
code:  ' * ball saver feature is still on.
          Case "BALLSAVER_MADE"
        Message(1)= "   BALL  "
        Message(2)= "  SAVED  "
        Message(3)= ""
        Message(4)= ""
         AddMusicSet "BallSaverMade"
```

A Note To Coders

AddScoringEvent will point to AddMusicSet to determine which company music set will play. From there, AddMusicSet will select a display (blinking, scrolling, other fx) and also a lighting effect.

Engine Code

This feature Event is part of the main engine core, and has several objects that is part of the table. These objects must be present in the table, deleting any of these objects will cause a error message when you play the table.
Objects needed

- LightBallSaver (light object)
- BallSaverTimer (timer object)
- PlungerLaneTrigger (trigger object)(in plunger lane)

List of main variables:

- `Dim bBallSaverActive` - Check if the Ball Saver is Active and in progress, or if it's "off" or disabled.
- `Dim BallSaverStart` - Flag used to control if the ball saver feature can be used, or if it has already been finished and is not to be used again for that ball.
- `Dim fpxBallSaverTime` - used to set the time for ballsaver (in seconds). This is a user defined option in the user input section at the top of the script.

List of Main Engine Subroutines:

- `Drain_Hit()` - The ball saver feature is checked at a Drain_hit, to see if the BallSaver feature is active. If so, the engine will create a replacement ball, and kick it back out to the plunger so the player can continue to play his ball.
- `ReplaySpecial()` - Called after AddScore(), this checks if the player's score goes over the replay scores as set by the user in fpxReplay1 and fpxReplay2 (settings are at the top of the script in the user input section) and awards a free game.

The Main AddScoringEvent Code

```
code: case "BallSaver"
         bBallSaverActive = TRUE:BallSaverStart=1  ' set our
      game  flag
        BallSaverTimer.Interval =  fpxBallSaverTime  '
      Interval time for ballsaver. Set the time at the top
      of the script
        LightBallSaver.State=BulbOn:BallSaverTimer.Enabled
```

```
= TRUE
 IF constAddDebug = 1 THEN AddDebugText
"BallSaverTimer Interval =" & fpxBallSaverTime
```

The **PlungerLaneTrigger** is part of the engine, and uses a timer called `BallSaverTimer,` here's a copy of the code used. If the ballsaver feature is to be active, PlungerLaneTrigger will activate the ball saver once the ball is rolled over. `BallSaverStart` is the variable that only allows ball saver to be used once from the plunger (only at the start of the players ball)

code:

```
' A ball is pressing down the trigger in the
shooters lane
Sub PlungerLaneTrigger_Hit()
 IF constAddDebug = 1 THEN AddDebugText
"PlungerLaneTrigger_Hit()"
  bBallInPlungerLane = TRUE
 GameInProgress=1    ' LEAVE THIS LINE ALONE, DO NOT
DELETE this tells the script a game is started
 If (fpxBallSaverTime <> 0) And (bBallSaverActive
<> TRUE) Then                ' if there is a need
for a ball saver, then start off a timer, only start
if it is currently not running
  If (BallSaverStart=1) Then  ' and only if the ball
hit the trigger in the plunger wire. (ball in the
shooters lane)
    AddScoringEvent "BallSaver" ' Start the Ball
Saver timer
   IF constAddDebug = 1 THEN AddDebugText
"PlungerLaneTrigger_Hit() - StartBallSaver()"
  End If
 End If
 IF constAddDebug = 1 THEN AddDebugText
"PlungerLaneTrigger_Hit() -
GameInProgress" & "" & (GameInProgress)
 set LastSwitchHit = PlungerLaneTrigger  ' remember
last trigger hit by the ball
End Sub
' The ball saver timer has expired.  Turn it off and
reset the game flag
'
Sub BallSaverTimer_Expired()
 IF constAddDebug = 1 THEN AddDebugText
"BallSaverTimer_Expired()"
 BallSaverTimer.Enabled = FALSE     ' stop the timer
from repeating
  bBallSaverActive = FALSE
: LightBallSaver.State=BulbOff:BallSaverStart=0 '
clear the flag, turn off the light
End Sub
```

Code for the ballsaver feature in Drain_Hit()
The code first checks to see if the BallSaver is still active. If it is, then it will run the Ballsaver made routine,and then pass it on to the Drain2 _hit routine to kick out a replacement ball. The second part of the code is for if the ballsaver is NOT active, it will then run the end of ball routine and start the bonus countdown.

code:
```
        If (bBallSaverActive  = TRUE) AND (fpTilted
        = False) Then                     ' This checks if
        the BallSaver is active
          IF  constAddDebug = 1 THEN AddDebugText
        "BALLSAVER_MADE  "
           ' *** ENGINE CODE. DO NOT DELETE THIS!!!! ***
           Drain2()                                    ' Kicks
        ball out and bypasses ball end routines
           AddEngineEvent "BALLSAVER_MADE"
                  ' *HOOK* to AddEngineEvent subroutine where
        you can add your unique code for Ballsaver
           SetEventNum=15                                   '
        sets case for TimerSetEvent1. This points to end of
        the routine
            TimerSetEvent1.Enabled = True
                ' run AddEngineEvent timer at interval to
        clear message, restore scoring
            BallSaverStart=0                                 '
        ENGINE CODE VERY IMPORTANT!!! DO NOT REMOVE!!!
          Exit Sub                                     ' GET
        OUT of this subroutine, do not pass go, do not
        collect 200 dollars
         End If
         If (bBallSaverActive  = FALSE) Then
                   ' Ballsaver Not active
              MemorySave()                                 '
        Save Player memory
             AddEngineEvent "STOPMUSICPLAYING"
                    ' Stops music channels used by engine.
             PlayMusic 6, "drain_delay",,  fpxMaxMusicVolume
               TimerSetEvent1.Interval = 500
               ' set the delay time before the bonus countdown
        begins
              SetEventNum=7:TimerSetEvent1.Enabled = TRUE
                        ' And use the AddEngineEvent Timer to
        start the bonus countdown
         End If
```

Code for Drain2_Hit()
When Drain_Hit() passes on control while it is plaing the "ball is saved" routine, Drain2() then handles the replacement ball so that player can continue his ball in play. Drain2 also turns off any active ballsaver code, sets bBallSaverActive to off so the ball saver doesn't relight again coming from the plunger and then exits the subroutine to prevent any additional code from being used.

code:
```
 If (fpGameInPlay  = TRUE) And (fpTilted
= FALSE) Then                        ' if there is a
game in progress and
  If (bBallSaverActive  = TRUE) Then
        ' is the ball saver active,
   IF  constAddDebug = 1 THEN AddDebugText
"bBallSaverActive=TRUE"
     CreateNewBall()                               '
yep, create a new ball in the shooters lane
     BallSaverTimer.Enabled  = FALSE
       ' stop the ball saver timer from repeating
    bBallSaverActive = FALSE
: LightBallSaver.State=BulbOff                    '
clear the ball saver flag
   Exit Sub
  Else
   ' NOTE: THIS IS A ERROR CATCHER, it SHOULD be
caught at drain_Hit(), but because of other
features, the code may point here instead
   If (BallsOnPlayfield  = 1) Then
      ' cancel any multiball if on last ball (ie.
lost all other balls)
    If (bMultiBallMode  = True) then
       ' and in a multi-ball??
     IF  constAddDebug = 1 THEN AddDebugText
"Drain2()  -  bMultiBallMode=FALSE"
      bMultiBallMode =  False
  ' not in multiball mode any more
      AddEngineEvent "SetUpMultiball"
           ' Resets to allow multiball. This is
called multiple times, so all the code is placed in
the  AddEngineEvent
    End If
   End If
   If (BallsOnPlayfield  = 0) Then
      ' was that the last ball on the playfield
     EndOfBall()                                  '
handle the end of ball (change player, high score
entry etc..)
   End If
  End If
 End If
```

BallSaverTimer
A basic timer used to keep track of the amount of time the ball saver is active. Once that set time is finished, the timer clears all the lights and variables

code:
```
' The ball saver timer has expired.  Turn it off and
reset the game flag
```

```
Sub BallSaverTimer_Expired()
 IF  constAddDebug = 1 THEN AddDebugText
"BallSaverTimer_Expired()"
 BallSaverTimer.Enabled  = FALSE
      ' stop the timer from repeating
  bBallSaverActive = FALSE
: LightBallSaver.State=BulbOff:BallSaverStart=0
        ' clear the flag, turn off the light
End Sub
```

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature uses "LockDisplay" to override any other music, lights, and display routines until the time specified is finished. (mechanical sounds will still play) While the music is playing, no other music/display/lights will be used, as this is considered a main feature award and should have priority over anything else.

Jackpot

Jackpot

The Jackpot feature is a special feature that is built up during a ball in play and then awarded to the player when that player makes a special feature or hits a certain object. This is a very common feature with modern tables, and is used mainly for multiball play, though you can use the jackpot event anytime you want to. You can add in your code that adds to the Jackpot value directly in your hit code, and have it score a normal jackpot or a superjackpot, which is the jackpot value times a multiplier.

To increase the value of a jackpot, place this within your hit code of the object you wish to use. (Example is here)

The example code will add 5000 points to the jackpot total.

```
code:  AddJackpot(5000)          ' Adds to Jackpot total
```

Hit Code

In the beginners template is a example that will award the Jackpot value to the players score total using a trigger. You can replace the 2 messages as needed, the engine will add the Jackpot value in the other two messages (in Display/HUD 3 and 4)

```
code:  Sub fpxJackpot_Hit()                          ' Scores a
       jackpot
        ' NOTE: There are only 2 messages needed. The
       engine will add the Jackpot value in the other two
       messages (in Display/HUD 3 and 4)
        Message(1)= "JACKPOT"
        Message(3)= "JACKPOT"
         AddScoringEvent "Jackpot"
       End Sub
```

Using this code will award the SuperJackpot value to the players score total. There are only 2 messages needed. The engine will add the Super Jackpot value in the other two messages (in Display/HUD 3 and 4)

```
code:  Sub fpxSuperJackpot_Hit()                        ' Scores
       Super Jackpot (Jackpot x fpxSuperJackpotMultiplier)
        ' NOTE: There are only 2 messages needed. The
       engine will add the Super Jackpot value in the other
       two messages (in Display/HUD 3 and 4)
        Message(1)= "S U P E R"
        Message(2)= "JACKPOT  "
         AddScoringEvent "SuperJackpot"
       End Sub
```

Settings

```
code:  ' * Jackpot
         fpxJackpotmin = 10000         ' minimum Jackpot a
       player can score (In points) To turn off Jackpot,
       set both min and max values to 0
         fpxJackpotmax = 250000        ' Maximum Jackpot a
       player can score (In points)
          fpxSuperJackpotMultiplier = 2  ' multiples the
       existing Jackpot score by this multiplier
```

The settings for this feature can be found in the User Input Section at the top of the script.

`fpxJackpotmin` - is the minimum jackpot a player can score

`fpxJackpotmax` - is the maximum jackpot a player can score
`fpxSuperJackpotMultiplier` - is the jackpot multiplier used for superjackpot. This will score the jackpot value times the fpxSuperJackpotMultiplier, so if your jackpot is 10000, and your multiplier is 3, then the superjackpot will score 30000 points

A Note To Coders

The Jackpot feature is independent of the built in FP version.

Engine Code

List of main variables:

- `Dim JackpotAward` -The jackpot award given (in points).

  These are set in the user input setting at the top of the script

- `Dim fpxJackpotmin` - The minimum amount a Jackpot will score.
- `Dim fpxJackpotmax` - The maximum amount a jackpot will score.
- `Dim fpxSuperJackpotMultiplier` - the multiplier of a jackpot used for "Super Jackpot".

  List of Main Engine Subroutines:

- `Drain_Hit()` - The ball saver feature is checked at a Drain_hit, to see if the BallSaver feature is active. If so, the engine will create a replacement ball, and kick it back out to the plunger so the player can continue to play his ball.
- `ReplaySpecial()` - Called after AddScore(), this checks if the player's score goes over the replay scores as set by the user in fpxReplay1 and fpxReplay2 (settings are at the top of the script in the user input section) and awards a free game.

  The Main AddScoringEvent Code

```
code: case "Jackpot"
       IF  constAddDebug = 1 THEN AddDebugText
      "AddScoringEvent  Jackpot"
       If (JackpotAward >= fpxJackpotmax) Then   ' Sets
      jackpot to a upper limit or lower limit, set at very
      top of the script
         JackpotAward = fpxJackpotmax
       Else
        If (JackpotAward  < fpxJackpotmin) Then
          JackpotAward = fpxJackpotmin
        End If
       End if
         AddScore(JackpotAward)
       IF LockDisplay=1 THEN LockDisplay=0   ' We need to
      check IF there is another Event going on first so we
      override it to run this next code
```

```
 FlushDisplay():TimerBackgroundMusic.Enabled
= FALSE:AddLightFX "Stop"  ' Kills any
light/sound/music display routines running
 SetEventNum=11:AddEngineEvent "JACKPOT":AddMusicSet
"JACKPOT"
  JackpotAward = fpxJackpotmin      ' resets to min
jackpot for the next time
 IF constAddDebug = 1 THEN AddDebugText "Jackpot()
" & (JackpotAward) & " pts"
 set LastSwitchHit = fpxJackpot

case "SuperJackpot"
 IF constAddDebug = 1 THEN AddDebugText
"AddScoringEvent Jackpot"
 If (JackpotAward >= fpxJackpotmax) Then   ' Sets
jackpot to a upper limit or lower limit, set at very
top of the script
   JackpotAward = fpxJackpotmax
 Else
  If (JackpotAward < fpxJackpotmin) Then
    JackpotAward = fpxJackpotmin
  End If
 End if
 AddScore(JackpotAward
* (fpxSuperJackpotMultiplier))
 IF LockDisplay=1 THEN LockDisplay=0   ' We need to
check IF there is another Event going on first so we
override it to run this next code
 FlushDisplay():TimerBackgroundMusic.Enabled
= FALSE:AddLightFX "Stop"  ' Kills any
light/sound/music display routines running
 SetEventNum=11:AddEngineEvent
"SUPERJACKPOT":AddMusicSet "SUPERJACKPOT"
  JackpotAward = fpxJackpotmin      ' resets to min
jackpot for the next time
 IF constAddDebug = 1 THEN AddDebugText
"SuperJackpot() " & (JackpotAward) & " pts"
 set LastSwitchHit = fpxSuperJackpot
```

AddJackpot
This "builds" up the jackpot value and storing that value in the `JackpotAward` variable. Note the checks for minimum and maximum routines. Awarding a jackpot orSuper Jackpot is done within your Hit code.

```
code: ' Adds points to the jackpot. Works exactly like
      AddScore, but you still have to collect it. You
      would write AddJackpot(100) in your code to add 100
      points to the jackpot
      Sub AddJackpot(points)
       If (fpTilted = False) Then
```

```
   ' Jackpots only generally increment in multiball
mode and not tilted but this dosn't have to be the
case
   IF  constAddDebug = 1 THEN AddDebugText
"AddJackpot()  " & (points) & " pts"
    JackpotAward =  JackpotAward + points
   If (JackpotAward  >= fpxJackpotmax) Then
            ' you may wish to limit the jackpot to a
upper limit or lower limit, ie..
      JackpotAward= fpxJackpotmax
   Else
    If (JackpotAward  < fpxJackpotmin) Then
      JackpotAward= fpxJackpotmin
    End If
   End if
 End if
End Sub
```

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature uses "LockDisplay" to override any other music, lights, and display routines until the time specified is finished. (mechanical sounds will still play) While the music is playing, no other music/display/lights will be used, as this is considered a main feature award and should have priority over anything else.

Extra Ball

Extra Ball
AddScoringEvent ››

Extra Ball

A Extra Ball is an additional bonus ball that can be earned by achieving a specific task. If the player has made a extra ball, and then loses his ball (drained) then the engine will score the bonus count and then give the same player a additional ball to play.

```
code:  Message(1)= "EXTRA"
```

```
          Message(2)= "BALL  "
          Message(3)= "EXTRA"
          Message(4)= "BALL  "
           AddScoringEvent "ExtraBall"
```

Settings

The settings for this feature can be found in the User Input Section at the top of the script.

```
code:  fpxMaxExtraBalls=1
```

This sets the limit to amount of extra balls a player can earn per ball.

```
code:  fpxMaxExtraBallAlternateScore=25000
```

In case a player already has made a Extra Ball, and is over the extra ball limit set by `fpxMaxExtraBalls` on the same ball in play, this instead will give a alternate score in points. Default is 25 thousand points added to the score.

Changing the Display Message

The messages are changed in your hit code, so you can have multiple extra ball features in your table design each with their own message. All 4 displays are used for the extra ball, the display effects are handled automatically by the engine depending on which music set you are using.

```
code:  Sub fpxExtraBall_Hit()  ' Scores Extra Ball
          Message(1)= "EXTRA"
          Message(2)= "BALL  "
          Message(3)= "EXTRA"
          Message(4)= "BALL  "
           AddScoringEvent "ExtraBall"
       End Sub
```

A Note To Coders

AddScoringEvent will point to AddMusicSet to determine which company music set will play. From there, AddMusicSet will select a display (blinking, scrolling, other fx) and also a lighting effect.
This feature uses the AddEventTimer with a case setting that is part of the main engine core. We use the `SetEventNum` variable to control that, so do not delete that line of code.
We also point to AddEngineEvent in the code. These are hooks from the main engine core, and though the case setting is blank, this still allows you to add code if you wish to add custom code to this feature without having to directly modify the main engine code.

⚙ Engine Code

This feature Event is part of the main engine core, and has a round light that is part of the table. This object must be present in the table, deleting this object will cause a error message when you play the table.

Objects needed

- LightShootAgain (light object)

List of main variables:

- `Dim ExtraBallsAwards(4)` - Number of EB's out-standing (for each player)

  These are set in the user input setting at the top of the script

- `Dim fpxMaxExtraBalls` -The jackpot award given (in points).
- `Dim fpxMaxExtraBallAlternateScore` - The minimum amount a Jackpot will score.

  List of Main Engine Subroutines:

- `Drain_Hit()` - The ball saver feature is checked at a Drain_hit, to see if the BallSaver feature is active. If so, the engine will create a replacement ball, and kick it back out to the plunger so the player can continue to play his ball.
- `TimerSetEvent1_Expired()` - (Case 8) Timer used by AddEngineEvent, this closes the code when a extra ball is made and resets the displays back to show the players score.

  The Main AddScoringEvent Code

```
code: case "ExtraBall"
       SetEventNum=8
       IF LockDisplay=1 THEN LockDisplay=0
          ' We need to check IF there is another Event
      going on first so we override it to run this next
      code
        AddEngineEvent "EXTRABALL":AddMusicSet  "EXTRABALL"
       IF  constAddDebug = 1 THEN AddDebugText
      "AddScoringEvent
      ExtraBall"

      &
       "" & "(+" & (ExtraBallsAwards(CurrentPlayer)) & ")"
       LightShootAgain.FlashForMs
      (MusicIntervalTime),
       (FlashForMSBlinkInterval),BulbOn
       If ExtraBallsAwards(CurrentPlayer)>fpxMaxExtraBalls
      Then
       ' check amount of extra balls already made, if we
      are already at the maximum amount of extra balls...
```

```
 ExtraBallsAwards(CurrentPlayer)=fpxMaxExtraBalls   '
check and force set amount of Extra Balls
 End If
 ExtraBallsAwards
(CurrentPlayer
) = ExtraBallsAwards(CurrentPlayer) + 1
 IF ExtraBallsAwards(CurrentPlayer)>fpxMaxExtraBalls
THEN  AddScore (fpxMaxExtraBallAlternateScore)
```

ResetForNewGame()

Note: only the code related to extra ball is shown here, all other code has been stripped out.
This clears out and resets all variables at the start of each new game.

```
code:   FOR i = 1 To constMaxPlayers         'initialize all
       the variables (do all players in case any new ones
       start a game)
          BallsRemaining(i) = nvBallsPerGame ' Balls Per
       Game
          ExtraBallsAwards(i) = 0            ' Number of
       EB's out-standing
        NEXT
```

EndOfBallTimer_Expired()

When the player is awarded a extra ball, and at the loss of his ball, this routine handles giving the extra ball to the player, display the message and music prompt, and kicking the ball out to the plunger.
This routine calls the AddEngineEvent subroutine (case "SHOOTAGAIN") so you can insert any custom code you wish to use, and is also the place where you can change the display message that is shown. The rest of the code then calls TimerSetEvent1 to "clean up" and reset any Extra Ball code before it pops out a ball for that player to play. You will notice the display routine code as well which displays the message you can set from AddEngineEvent "SHOOTAGAIN".

```
code:  If (ExtraBallsAwards(CurrentPlayer) <> 0) Then
                           ' has the player won an extra-
       ball ? (might be multiple outstanding)
         IF constAddDebug = 1 THEN AddDebugText
       "EndOfBallTimer_Expired() - Award XBall"

        ExtraBallsAwards
       (CurrentPlayer
       ) = ExtraBallsAwards(CurrentPlayer) - 1
       ' yep got to give it to them
          If (ExtraBallsAwards(CurrentPlayer) = 0) Then
                           ' if no more EB's then turn off any
       shoot again light
            LightShootAgain.State = BulbOff
```

```
   End If
     FlushDisplay()                                   '
Stop Present display routine
    AddEngineEvent "SHOOTAGAIN"
      ' To AddEngineEvent subroutine where you can
add your unique code, points to CreateNewBall()
   For x = 1 to SegCount:Seg(x).UpdateInterval
= 50:Seg(x).SlowBlinkSpeed
=100:HSeg(x).UpdateInterval
= 50:HSeg(x).SlowBlinkSpeed =100:Next
    Seg(1).QueueText (Message(1)) & ""
          ' Add the static message first due to
better effect
    Seg(2).QueueText (Message(2)) & ""
    Seg(3).QueueText (Message(1)) & ""
    Seg(4).QueueText (Message(2)) & ""
     HSeg(1).QueueText (Message(1)) & ""
     HSeg(2).QueueText (Message(2)) & ""
     HSeg(3).QueueText (Message(1)) & ""
     HSeg(4).QueueText (Message(2)) & ""
    Seg(1).QueueText
(Message(1)) & "", seBlink, 2000, 0, FALSE, ""
          ' Now we blink it.
    Seg(2).QueueText
(Message(2)) & "", seBlink, 2000,0, FALSE, ""
    Seg(3).QueueText
(Message(1)) & "", seBlink, 2000, 0, FALSE, ""
    Seg(4).QueueText
(Message(2)) & "", seBlink, 2000, 0, FALSE, ""
     HSeg(1).QueueText
(Message(1)) & "", seBlink, 2000, 0, FALSE, ""
     HSeg(2).QueueText
(Message(2)) & "", seBlink, 2000,0, FALSE, ""
     HSeg(3).QueueText
(Message(1)) & "", seBlink, 2000, 0, FALSE, ""
     HSeg(4).QueueText
(Message(2)) & "", seBlink, 2000,0, FALSE, ""
     SetEventNum=9:TimerSetEvent1.Enabled = True
                ' clean up and then runs
AddEngineEvent timer to point to CreateNewBall()
 Else                                          ' no extra
balls

 BallsRemaining
(CurrentPlayer) = BallsRemaining(CurrentPlayer) - 1
   If (BallsRemaining(CurrentPlayer) <= 0) Then
                ' it was the last ball
      AddEngineEvent "STOPMUSICPLAYING"
          ' Stops music channels used by engine.
      bEnteringAHighScore = True
    ' Enter high score
```

```
            EnterHighScore(CurrentPlayer)
          Else
            EndOfBallComplete()
          End If
        End If
```

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature uses "LockDisplay" to override any other music, lights, and display routines until the time specified is finished. (mechanical sounds will still play) While the music is playing, no other music/display/lights will be used, as this is considered a main feature award and should have priority over anything else.

Special

Special
AddScoringEvent »

Special

A special award is a free game, that a player earns by either making a certain task, or by exceeding a certain scoring goal, such as a replay. In the fpxEngine, there is a example that lights the special award in the outlanes with it's own playfield insert on the playfield.

If You Have Never Coded Before

Just a reminder, to use this feature, the code should be encased within a subroutine *code:*

```
Sub fpxMyTrigger_Hit
   'This is where your code goes....
End Sub
```

There are 3 separate types of specials built into fpx, the first is the basic example using a trigger, the second is for the outlane "special", and the third handles the Replay goals and high score specials.

To award a special in game, just make sure this code is included in your hit code. Both the Outlane specials, fpxReplay1 and fpxReplay2 will point to this subroutine first for the messages to be displayed before it turns off the lights for the outlanes. **MAKE SURE THIS SUBROUTINE IS PRESENT as this is the subroutine that is pointed to for making replay goals.** If you delete fpxSpecial, you will no longer be able to award specials, and may cause the table to display a error message. Doing the code this way means you can have display messages for all 3 special routines. you can modify the text to be displayed (between the quotes) just make sure each line is no more than 9 characters

```
code:  ' NOTE: KEEP THIS CODE FOR SPECIALS, DO NOT DELETE!
       Sub fpxSpecial_Hit()                    ' Scores a
       free game
        Message(1)= "SPECIAL"
        Message(2)= "SPECIAL"
        Message(3)= "SPECIAL"
        Message(4)= "SPECIAL"
         AddScoringEvent "Special"
       End Sub
```

Settings

```
code:   fpxReplay1 = 500000     ' First replay award
        fpxReplay2 = 750000     ' Second Replay award
        fpxHighScore = 1250000  ' High Score award default
       value.
        fpxMaxCredits =15       ' Maximum amount of credits
       a game will set.
```

The settings for this feature can be found in the User Input Section at the top of the script.

`fpxReplay1` sets the first replay goal for the player
`fpxReplay2` sets the second replay goal for the player
`fpxHighScore` sets the default high score.
`fpxMaxCredits` sets the maximum amount of credits a player can have.

Changing the Display Message

Because this feature is used existentially by the main engine, the message code and any other code should be placed within the Hit() section as part of a subroutine. The fpxBeginnersTutorial already has this following code as a example, so just search for that subroutine and make your changes within the quotes. All 4 displays are used for the special, the display effects are handled automatically by the engine depending on which music set you are using. Because this code is used to handle messages for all specials, make sure this entire code is present in your hit code. The Outlanes page in this section has the example for giving out a free game, but the basic code below is all you need.

```
code:   Message(1)= "SPECIAL"
        Message(2)= "SPECIAL"
        Message(3)= "SPECIAL"
        Message(4)= "SPECIAL"
        AddScoringEvent "Special"
```

A Note To Coders

AddScoringEvent will point to AddMusicSet to determine which company music set will play. From there, AddMusicSet will select a display (blinking, scrolling, other fx) and also a lighting effect.

Engine Code

This feature Event is part of the main engine core, and has several objects that is part of the table. These objects must be present in the table, deleting any of these objects will cause a error message when you play the table.

List of main variables for Special

- `Dim ReplayToReach(4)` - Stock FP code, this is the range of replay needed by player

These are set in the user input setting at the top of the script

- `Dim fpxReplay1` - sets the first replay goal for the player
- `Dim fpxReplay2` - sets the second replay goal for the player
- `Dim fpxHighScore` - sets the default high score
- `Dim fpxMaxCredits` - sets the maximum amount of credits a player can have.

Objects needed

- LightLeftOutLaneTrigger, LightRightOutLaneTrigger, CreditLight (light objects)
- Dispcredit (display object, in translight)
- LeftOutLaneTrigger, RightOutLaneTrigger (trigger object)(Outlanes)

The Main AddScoringEvent Code

```
code: case "Special"
       SetEventNum=10   ' sets case for TimerSetEvent1.
      end of the routine, which calls CreateNewBall()
       IF LockDisplay=1 THEN LockDisplay=0 ' check IF
      there is another Event going on first to override it
        AddEngineEvent "SPECIAL":AddMusicSet "SPECIAL"
       IF constAddDebug = 1 THEN AddDebugText
      "AddScoringEvent Special"
       IF nvCredits < fpxMaxCredits THEN
         nvCredits = nvCredits +1
          DispCredit.AddValue(1): CheckCredit
          CreditLight.State = BulbOn: PlaySound "Knocker",
      (fpxSoundVolume)
         PlaySound "Knocker",(fpxSoundVolume)
       END IF
```

The AddScoringEvent code for the Outlanes Specials. Note that this code actually loops back to the "Special" code above.

code:
```
case "OutLaneSpecial"
 ' special routine for Outlanes. This repoints to
stock special code, and switches off special lights
at outlane
 SetEventNum=10 ' sets case for TimerSetEvent1. end
of the routine, which calls CreateNewBall()
 IF LockDisplay=1 THEN LockDisplay=0 ' We need to
check IF there is another Event going on first so we
override it
  AddEngineEvent "SPECIAL":AddMusicSet  "SPECIAL"
 IF  constAddDebug = 1 THEN AddDebugText
"AddScoringEvent  Special"
 IF  nvCredits <  fpxMaxCredits THEN
   fpxSpecial_Hit()
 END IF
 ' Have to remember to swich those special lights
off!
 LightLeftOutLaneTrigger.State=BulbOff
 LightRightOutLaneTrigger.State=BulbOff
```

ReplaySpecial()

This handles if player makes a replay goal

code:
```
Sub ReplaySpecial()
 If (fpTilted  = False) Then
  If nvScore(CurrentPlayer) >=  fpxReplay2 and
ReplayToReach(CurrentPlayer) =1 Then          ' any
replay wins? Note the order of which replay we check
first

 ReplayToReach
(CurrentPlayer) = ReplayToReach(CurrentPlayer) +1

   If  nvCredits <  fpxMaxCredits Then
     fpxSpecial_Hit()
    IF  constAddDebug = 1 THEN AddDebugText
"ReplaySpecial()  -  ReplayToReach(CurrentPlayer)  =2"
   End If
  End If
  If nvScore(CurrentPlayer) >=  fpxReplay1 and
ReplayToReach(CurrentPlayer) =0 Then

 ReplayToReach
(CurrentPlayer) = ReplayToReach(CurrentPlayer) +1

   If  nvCredits <  fpxMaxCredits Then
```

```
          fpxSpecial_Hit()
         IF  constAddDebug = 1 THEN AddDebugText
"ReplaySpecial()  -  ReplayToReach(CurrentPlayer)  =1"
        End If
       End If
       If ReplayToReach(CurrentPlayer) => 3 Then
          ReplayToReach(CurrentPlayer) = 3
        IF  constAddDebug = 1 THEN AddDebugText
"ReplaySpecial()  -  ReplayToReach(CurrentPlayer)  =3"
       End If
      End If
     End Sub
```

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature uses "LockDisplay" to override any other music, lights, and display routines until the time specified is finished. (mechanical sounds will still play) While the music is playing, no other music/display/lights will be used, as this is considered a main feature award and should have priority over anything else.

AddMultiplier

AddMultiplier

AddMultiplier

A feature found on many games that allows you to multiply the end of ball bonus or a mode bonus by some factor, such as x2, x3, etc., if certain features are hit enough times. The fpxEngine supports up to a multiplier of x15.

The engine has 5 main multiplier lights, from 1x to 5x, and has a built in routine to handle the correct multiplier as the score counts down so there is nothing needed for additional coding.

If You Have Never Coded Before

Just a reminder, to use this feature, the code should be encased within a subroutine

```
code: Sub fpxMyTrigger_Hit
         'This is where your code goes....
      End Sub
```

```
code: Message(1)= "BONUS   ":Message(2)= "MULTI   "
      AddScoringEvent  "AddMultiplier"
```

In the code, you need to add 2 message lines for the display text, with the message you wish to display in the quotes. If you leave the quotes blank, then the display will show blank the the duration of the event routine. Display 3 and 4 will show the multiplier value made automatically, so you just need to add messages for 1 and 2.

Settings

```
code:   MaximumBonusMultiplier = 5 ' Forced max amount
```

The MaximumBonusMultiplier is the maximum amount of multipliers you want in your game. (up to 15x) Once the Engine reaches that amount, no more multipliers will be added. The multiplier resets back to 1x at the start of each new ball (including the start of a extra ball).

The settings for this feature can be found in the User Input Section at the top of the script.

Engine Code

This feature Event is part of the main engine core, and has several objects that is part of the table. These objects must be present in the table, deleting any of these objects will cause a error message when you play the table.

List of main variables for Bonus Multiplier

- `Dim MaximumBonusMultiplier` - Sets the maximum amount of Bonus Multipliers a player can reach in game. This value is reset to 1x at the start of each new ball.
- `Dim Mult` - Variable that stores the Bonus Multiplier value while the ball is in play. As a example, `mult=2` means that the Bonus Multiplier is at 2x, `mult=4` means the Bonus Multiplier is 4x etc.

List of main Engine Subroutines

- `MultiplierLight()` - Handles the routine to assign a proper light to match the multiplier value when the ball is still in play, and not during the bonus count routine.
- `CheckMulti()` - 'This routine checks for the multiplier (from the bonus countdown routine), and calls `MultiplierLight()` to handle and display the proper BonusX light while the bonus is counting down. Once all the bonus, and the Bonus Multiplier are finished, this routine then calls the end of ball routine to start the next player ball.
- `TimerBonus_Expired()` - The main bonus routine, this handles the actual bonus and calls `CheckMulti()` to see if there are any multipliers left to do.
- `ResetBonus()` - This resets the bonus and the Bonus Multipliers. This is called

once the countdown routine is finished and part of the end of the players ball routine.

Objects needed

- Light1x (light object)
- Light2x (light object)
- Light3x (light object)
- Light4x (light object)
- Light5x (light object)
- TimerBonus (timer object)(Part of Bonus routine)
- TimerSetEvent1(timer object)(main Engine Timer)
- PlungerLaneTrigger (trigger object)(in plunger lane)

The Main AddScoringEvent Code

```
code: case  "AddMultiplier"
          AddDebugText "AddMultiplier()"
       ' Press F9 instead of play icon in editor to see
      debug info. (Read the manual)
         mult = mult+1
         If  MaximumBonusMultiplier > 15
      then  MaximumBonusMultiplier = 15          ' error
      checker to prevent Multiplier going over 15 if set
      by user
         For z = 1 To 5 : eval("Light" & z & "x").State
      = 0: Next             ' Switch off mult lights first,
      so we just need to turn on the lights we want in the
      case setting
         If mult >  MaximumBonusMultiplier Then mult
      =  MaximumBonusMultiplier : End If          ' Forced
      max mult set by user
         If mult > 15 Then mult = 15 : End If
        ' Forced mult max to prevent errors
          AddMusicSet "AddMultiplier"
            MultiplierLight()
```

MultiplierLight()

This turns on the proper multiplier light when ever the player increases his Bonus Multiplier in-game. There is support for a maximum of 15 times with the multiplier. The engine is completely automated, and has set routines for each multiplier as it counts down, so there is no need to do anything. This can use all multiplier lights (including the "1x" light)
If there is a multiplier light you don't actually need, you should just drag that light so it's hidden underneath the apron. Do not delete any lights, you will get a error.

```
code:   Sub MultiplierLight()
       AddDebugText "AddMultiplierLight()"
                ' Press F9 instead of play icon in editor
      to see debug info. (Read the manual)
       ' Turn off all lights first before we turn on the
```

```
correct light for the multiplier value
 Light1x.State = BulbOff : Light2x.State = BulbOff
: Light3x.State = BulbOff : Light4x.State = BulbOff
: Light5x.State = BulbOff
  Select Case mult
   Case 15:For z = 1 To 5 : eval("Light" & z
& "x").State = BulbOn: Next              ' 15x
   Case 14:Light5x.State = BulbOn : Light4x.State
= BulbOn : Light3x.State = BulbOn : Light2x.State
= BulbOn  ' 14x
   Case 13:Light5x.State = BulbOn : Light4x.State
= BulbOn : Light3x.State = BulbOn : Light1x.State
= BulbOn  ' 13x
   Case 12:Light5x.State = BulbOn : Light4x.State
= BulbOn : Light3x.State = BulbOn          ' 12x
   Case 11:Light5x.State = BulbOn : Light4x.State
= BulbOn : Light2x.State = BulbOn          ' 11x
   Case 10:Light5x.State = BulbOn : Light4x.State
= BulbOn : Light1x.State = BulbOn          ' 10x
   Case 9 :Light5x.State = BulbOn : Light4x.State
= BulbOn                  ' 9x
   Case 8 :Light5x.State = BulbOn : Light3x.State
= BulbOn                   ' 8x
   Case 7 :Light5x.State = BulbOn : Light2x.State
= BulbOn                   ' 7x
   Case 6 :Light5x.State = BulbOn : Light1x.State
= BulbOn                   ' 6x
   Case 5 :Light5x.State = BulbOn
       ' 5x
   Case 4 :Light4x.State = BulbOn
       ' 4x
   Case 3 :Light3x.State = BulbOn
       ' 3x
   Case 2 :Light2x.State = BulbOn
       ' 2x
   case 1 :Light1x.State = BulbOn
        ' 1x
  End Select
 If  constAddDebug = 1 Then HudBonus1.Text
= (Bonus) & " " & (mult) & "x":End If         '
FOR DEMO PURPOSES
end sub
```

CheckMulti()

Though some of the code is similar to MultiplierLight(), this handles the multiplier routines when the player loses the ball and the bonus count starts. This actually controls the bonus count, everytime the bonus is counted down to 0, it calls this subroutine to check if there is still a bonus multiplier to do. Once the multiplier is finished (mult=0) then this routine will send the engine to reset the table for a start of a new ball, and also to start the next ball in play.

code:
```
Sub CheckMulti()
 AddDebugText "CheckMulti()  " & (mult) & "x"
                ' Press F9 instead of play icon in
editor to see debug info. (Read the manual)
  If mult >0 then                                  '
If there is still a Multiplier
  Bonus = HoldBonus                                 '
Restore total bonus collected during ball in play at
drain hit
  If mult >  MaximumBonusMultiplier Then mult
= MaximumBonusMultiplier:End If               ' Forced
Bonus Value set by user
  If mult > 15 then mult = 15
    ' Prevent multiplier going over 15
  ' Handles multiplier lights during a collect bonus
count
     Light1x.State = BulbOff: Light2x.State
= BulbOff: Light3x.State = BulbOff: Light4x.State
= BulbOff: Light5x.State = BulbOff
  ' Check for mult, add lights needed
  Select Case mult
   Case 15 : For z = 1 To 5 : eval("Light" & z
& "x").State = BulbOn: Next                        '
15x
   Case 14 : Light5x.State = BulbOn : Light4x.State
=  BulbOn : Light3x.State =  BulbOn : Light2x.State
=  BulbOn : Light1x.State =  BulbOff ' 14x
   Case 13 : Light5x.State = BulbOn : Light4x.State
=  BulbOn : Light3x.State =  BulbOn : Light1x.State
= BulbOn           ' 13x
   Case 12 : Light5x.State = BulbOn : Light4x.State
=  BulbOn : Light3x.State = BulbOn
' 12x
   Case 11 : Light5x.State = BulbOn : Light4x.State
=  BulbOn : Light2x.State = BulbOn
' 11x
   Case 10 : Light5x.State = BulbOn : Light4x.State
=  BulbOn : Light1x.State = BulbOn                  '
10x
   Case 9  : Light5x.State = BulbOn : Light4x.State
= BulbOn                              ' 9x
   Case 8  : Light5x.State = BulbOn : Light3x.State
= BulbOn                               ' 8x
   Case 7  : Light5x.State = BulbOn : Light2x.State
= BulbOn                               ' 7x
   Case 6  : Light5x.State = BulbOn : Light1x.State
= BulbOn                               ' 6x
   Case 5  : Light5x.State = BulbOn
```

```
                      ' 5x
        Case 4  : Light4x.State = BulbOn
                      ' 4x
        Case 3  : Light3x.State = BulbOn
                      ' 3x
        Case 2  : Light2x.State = BulbOn
                      ' 2x
        case 1  : Light1x.State = BulbOn : Light2x.State
= BulbOff :AddDebugText "last bonus count"
       End Select
      End if
      If mult = 0 then                              '
No more multipliers
         TimerBonusEnabled = False
          ' Turn off the timer
        ResetForNewPlayerBall()
   ' this calls reset bonus

       AddDebugText "Exit CheckMulti() " & (mult) & "x"
       Drain2()                                   ' On to
End Of Ball routines
      Else                                       ' Hey, you
tilted you dummy
        AddBonusLights()                              '
Loser, never tilt during a bonus count
      End If
End Sub ' end sub CheckMulti()
```

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature uses "LockDisplay" to override any other music, lights, and display routines until the time specified is finished. (mechanical sounds will still play) While the music is playing, no other music/display/lights will be used, as this is considered a main feature award and should have priority over anything else.

Mystery

Mystery
AddScoringEvent »

Mystery

The mystery feature (common in Williams games) is a random award that will light up using a timer for a specific period of time. The most notable game to have a mystery feature is Black Knight (1980) but this feature is used as well under another name by other companies. The score generated is a random score based on 2 values, a minimum and a maximum value.

If You Have Never Coded Before

Just a reminder, to use this feature, the code should be encased within a subroutine

```
code: Sub fpxMyTrigger_Hit
         'This is where your code goes....
      End Sub
```

Using this code will award the Mystery value to the players score total. The engine will add the mystery value in the other two messages (in Display/HUD 3 and 4)

```
code:   ' NOTE: You need to add messages here (or just
       leave it as a default) for the displays.
        ' The engine will add the award in the other two
       messages (in Display/HUD 3 and 4)
        Message(1)= "Mystery ":Message(2)= "Mystery "
         AddScoringEvent "Mystery"
```

Settings

```
code:  ' * Jackpot
         fpxJackpotmin = 10000        ' minimum Jackpot a
       player can score (In points) To turn off Jackpot,
       set both min and max values to 0
         fpxJackpotmax = 250000       ' Maximum Jackpot a
       player can score (In points)
         fpxSuperJackpotMultiplier = 2  ' multiples the
       existing Jackpot score by this multiplier
```

The settings for this feature can be found in the User Input Section at the top of the script.

```
MinMysteryAward=5000

         ' Min Mystery award
MysteryAwardMaxium=25000
```

```
    ' Max Mystery award
```

Changing the Display Message

You can change the message in the quotes if you wish to display a different message. Just remember that you are limited to 9 characters per display.

Engine Code

This feature Event is part of the main engine core, and can be used for most object hit events, such as a kicker. In the beginners tutorial, we use a star trigger, but you can delete that if you wish. This is always "turned on" and no light or timer is assigned to it. Please check the Vault section for any example that adds lights or timers (when available)

Not that the Mystery feature requires Randomize to be present in the script. Do not delete the Randomize code, as you will get errors and the table will not start.

The Main AddScoringEvent Code

```
code: MysteryAward  = (MinMysteryAward)+((int(rnd(1)*25))
      *1000)
      If MysteryAward>MysteryAwardMaxium
      then MysteryAward=MysteryAwardMaxium
      AddScore(MysteryAward)
      AddMusicSet  "Mystery"
      IF  constAddDebug = 1 THEN AddDebugText
      "AddScoringEvent  Mystery" & (MysteryAward)
```

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature uses "LockDisplay" to override any other music, lights, and display routines until the time specified is finished. (mechanical sounds will still play) While the music is playing, no other music/display/lights will be used, as this is considered a main feature award and should have priority over anything else.

In the Williams music set, Mystery uses the "radar" setting for the display.

Slingshots

Slingshots

Built into fpxEngine are the slingshot code. There is also additional code that has been added (such as the ball color changing example) as demonstrations, but this page will just deal with the main engine code. The slingshots above the flippers are pretty generic, and have been used by the actual companies for nearly 60 years. Since they are all pretty much the same, there is no need to go into detail.

Hit Code

You will find the main code for the Slingshots in the Hit Code section.
The code for the ball changing example has been removed to make it easier to understand.

code:
```
Sub LeftSlingshotRubber_Hit()                  ' The Left Slingshot has been Hit
 AddScore(10)                            ' Add some points. We add the value in the brackets, in this case 10 points, to our score
  AddScoringEvent "LeftSlingshot"                ' Call the preset routine for bulbs flashing/music etc
 'Search for Const fpxBallBlinkingOn. 0 is off, 1 is on. pg192 of BAM main thread
End Sub
Sub RightSlingshotRubber_Hit()                 ' The Right Slingshot has been Hit
 AddScore(10)                            ' Add some points. We add the value in the brackets, in this case 10 points, to our score
  AddScoringEvent "RightSlingshot"
End Sub
```

Engine Code

This feature Event is part of the main engine core, and has several objects that is part of the table. These objects must be present in the table, deleting any of these objects will cause a error message when you play the table. All other objects, such as ornaments, can be safely deleted.
Objects needed

- LeftSlingshotBulb1,LeftSlingshotBulb2,RightSlingshotBulb1,RightSlingshotBulb2 (pf bulb object)

- LeftSlingshotRubber,RightSlingshotRubber (rubber objects, set to slingshot)

  Nearly all slingshots traditionally give 10 points on a ball hit, but you can add a different value between the brackets in the AddScore line.
  The slingshots are set to automatically "flash" the 2 playfield bulbs on a hit, and has it's own mechanical sound that will play only when a game is being played and is not in a tilted state.

  The Main AddScoringEvent Code

```
code:  case "LeftSlingshot"
        LeftSlingshotBulb1.FlashForMs  100, 50,  BulbOff
                  ' Flash the pf bulb lights in side the
       slingshot, not really needed, just looks pretty
        LeftSlingshotBulb2.FlashForMs  100, 50, BulbOff
         PlaySound "slingshot",(fpxSoundVolume)     ' Solnoid
       sound
         AddMusicSet "sling"
       Case "RightSlingshot"
        RightSlingshotBulb2.FlashForMs  100, 50, BulbOff
         PlaySound "slingshot",(fpxSoundVolume)
         AddMusicSet "sling"
```

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature uses "LockDisplay" to override any other music, lights, and display routines until the time specified is finished. (mechanical sounds will still play) While the music is playing, no other music/display/lights will be used, as this is considered a main feature award and should have priority over anything else.

InLanes

InLanes
AddScoringEvent ››

In Lanes

If You Have Never Coded Before

We are going to be dealing with working with a light to your table, and also now what the script needs to do if that light is turned off, or turned on.

I would first recommend that you quickly review the light section in the fp manual first, as it gives you a lot of very helpful information on some of the settings, and explains them quite well. Just press the F1 key, or click on "Help" at the top menu in the editor, and "Open Manual". Go all the way down to "Table Components", click on that to expand the section, and then select "Lights" and finally "Round Lights" Whew, it's actually harder to use the help file than it is to use the fpx engine so far isn't it...

Built into fpxEngine are the Inlane Trigger code. The Inlane is the path feeding a falling ball from the playfield to the flippers, usually behind a slingshot. Some people refer to the Inlanes as the "flipper Return Lanes". The code used by the fpxEngine for inlanes is a bit more complex, as we also have a light object and additional coding based on the light state.

Hit Code

You will find the main code for the Inlanes in the Hit Code section. The code for both the right and the left inlanes are basically the same, we look at the light object assigned to each side (`LightLeftInLaneTrigger  and  LightRightInLaneTrigger`) and determine what to do depending on whether that light is turned on or turned off. In this case, the score is different, if the light is off, it gives 1000 points to the player, but if that light is on, then the player will earn 5000 points. The player also collects a bonus point that is collected at the loss of a ball. Note there is a seperate AddScoringEvent routine based on light state. This is to have different music/ light effect for each bulb state.

```
code: ' Return lane to left flipper
      Sub LeftInLaneTrigger_Hit()
         IF LightLeftInLaneTrigger.State=BulbOff THEN
       ' Look to see if light assigned is off first
           AddScore(1000)                 ' Add some points.
      a bit more than before
           AddBonus (1)                  ' Add to the Bonus
      Count when the ball is lost
           AddScoringEvent "InLaneUnlit_bally81"        '
      Go to routine for music etc
         END IF                       ' I'm finished telling
      you what to do if the bulb is off
         ' If the bulb is lit, lets reward the player with
      a better score
         IF LightLeftInLaneTrigger.State=BulbOn THEN
       ' Look to see if light assigned is on first
           AddScore(5000)                 ' Add some points.
      quite a lot to make the light on worthwhile
           AddBonus (1)                  ' Add to the Bonus
```

```
Count when the ball is lost
     AddScoringEvent "InLaneIslit_bally81"        '
Go to routine for music etc
   END IF                      ' I'm finished telling
you what to do if the bulb is on
End Sub

' Return lane to Right flipper
Sub RightInLaneTrigger_Hit()
   IF LightRightInLaneTrigger.State=BulbOff  THEN
  ' Look to see if light assigned is off first
     AddScore(1000)                  ' Add some points.
a bit more than before
     AddBonus (1)                  ' Add to the Bonus
Count when the ball is lost
     AddScoringEvent "InLaneUnlit_bally81"        '
Go to routine for music etc
   END IF                      ' I'm finished telling
you what to do if the bulb is off
    ' If the bulb is lit, lets reward the player with
a better score
   IF LightRightInLaneTrigger.State=BulbOn  THEN
 ' Look to see if light assigned is on first
     AddScore(5000)                  ' Add some points.
quite a lot to make the light on worthwhile
     AddBonus (1)                  ' Add to the Bonus
Count when the ball is lost
     AddScoringEvent "InLaneIslit_bally81"        '
Go to routine for music etc
   END IF                      ' I'm finished telling
you what to do if the bulb is on
End Sub
```

Turning on the InLane Lights

As part of the Beginners tutorial, we also have a example trigger that contains the code to turn on the bulbs when a ball rolls over that trigger.

```
code: Sub TriggerInlaneOn_Hit()
       LightLeftInLaneTrigger.State = BulbOn
       LightRightInLaneTrigger.State = BulbOn
       AddScore(100)' Add some points. We add the value in
      the brackets, in this case 10 points, to our score
        AddScoringEvent "trigger"
      End Sub
```

This is just a sample code, but what it does is turn on the Inlane lights, give a score for rolling over that trigger, and play some music for that particular trigger being made.

Turning off the InLane Lights

The engine will automatically turn off the inlane lights at a loss of a ball, and have to be re lit by the player with his next ball. The code is found in the AddEngineEvent subroutine, and in the Case "NEW_BALL". AddEngineEvent are the hooks to the main portions of the engine, and are there so you can add your own engine code or what you wish the engine to do in one place, as opposed to going through thousands of lines of code. The "NEW_BALL" case within AddEngineEvent handles all the code for the start of each new ball, just before the ball is kicked out to the plunger. In this case, the lights are "switched off" at the start of the next ball.

Note: We only include the code related to the Inlane/OutLane bulbs here, and have removed the other code related to a new player(s) start of a ball.

```
code:                   ' * Code for start of each ball, and
      what the system does
         Case "NEW_BALL"                          ' Called
      From  ResetForNewPlayerBall()
          ' THIS IS WHERE YOU ADD YOUR CODE FOR THE START
      OF EACH BALL !!!!
          '  Use this to turn on any lights, reset any
      variables you use, popup targets etc.
           ' this switches off the lights used in the
      Inlanes and Outlanes
              LightLeftInLaneTrigger.State=BulbOff
              LightRightInLaneTrigger.State=BulbOff
              LightLeftOutLaneTrigger.State=BulbOff
              LightRightOutLaneTrigger.State=BulbOff
```

A Note

Built into fpxEngine are the ability to Alternate the Inlane or OutLane Lights. This feature will alternate a lit Inlane or Outlane light as opposed to having the 2 lights for the Inlanes and Outlanes always on. The lanes "switch" sides when ever the ball strikes a powered slingshot, or if the ball strikes a Bumper.
If you want this feature, you will find the AddScoringEvent "AddAlternatingLanes" here

Engine Code

This feature Event is part of the main engine core, and has several objects that is part of the table. These objects must be present in the table, deleting any of these objects will cause a error message when you play the table. All other objects, such as ornaments, can be safely deleted.
Objects needed

- LightLeftInLaneTrigger,LightRightInLaneTrigger (round light object)
- LeftInLaneTrigger,RightInLaneTrigger (trigger objects, set to "wire")

The Main AddScoringEvent Code

```
code:    case  "InLaneUnlit_bally81"                  ' The
      Left InLane trigger has been Hit
           PlaySound "trigger",(fpxSoundVolume)       '
      Mechanical Sound, not affected by tilt state
           AddMusicSet "trigger"
          set  LastSwitchHit = LeftInLaneTrigger      '
      remember last trigger hit by the ball, it's good
      coding, but actually pretty useless at the moment
         case  "InLaneIslit_bally81"                  ' The
      Left InLane trigger has been Hit
           PlaySound "trigger",(fpxSoundVolume)       '
      Mechanical Sound, not affected by tilt state
           AddMusicSet "inlaneislit"
          set  LastSwitchHit = LeftInLaneTrigger      '
      remember last trigger hit by the ball, it's good
      coding, but actually pretty useless at the moment
```

Note that the engine has 2 routines for AddScoringEvent, one for each light state. We also play a mechanical sound when the ball rolls over the trigger.

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature will be overridden by any high priority feature, such as Extra Ball or a special, but the player will still receive credit if this feature is made during that time.

OutLanes

OutLanes
AddScoringEvent ››

OutLanes

If You Have Never Coded Before

We are going to be dealing with working with a light to your table, and also now what the script needs to do if that light is turned off, or turned on.

I would first recommend that you quickly review the light section in the fp manual first, as it gives you a lot of very helpful information on some of the settings, and explains them quite well. Just press the F1 key, or click on "Help" at the top menu in the editor, and "Open Manual". Go all the way down to "Table Components", click on that to expand the section, and then select "Lights" and finally "Round Lights" Whew, it's actually harder to use the help file than it is to use the fpx engine so far isn't it...

Built into fpxEngine are the OutLane Trigger code. These lanes are commonly refered to as "Drain Lanes" as you usually lose the ball in play. The code is very much the same as the Inlane Code, though Outlanes award higher scores and traditionally, award a special or free game when "lit". The code used by the fpxEngine for OutLanes is a bit more complex, as we also have a light object and additional coding based on the light state.

Hit Code

You will find the main code for the OutLanes in the Hit Code section. The code for both the right and the left OutLanes are basically the same, we look at the light object assigned to each side (`LightLeftOutLaneTrigger` and `LightRightOutLaneTrigger`) and determine what to do depending on whether that light is turned on or turned off. In this case, the score is different, if the light is off, it gives 3000 points to the player, but if that light is on, then the player will earn 5000 points and in this example, be awarded a special or free game. The player also collects a bonus point that is collected at the loss of a ball as well as having the jackpot amount increased. Note there is a separate AddScoringEvent routine based on light state. This is to have different music/ light effect for each bulb state.

code:
```
Sub LeftOutLaneTrigger_Hit()
   IF LightLeftOutLaneTrigger.State=BulbOff THEN
  ' Look to see if light assigned is off first
      AddScore(3000)                    ' add some points
      AddJackpot(3000)                    ' Adds to
Jackpot total
      AddBonus(1)                     ' adds to Bonus
score
      AddScoringEvent "OutLaneUnlit_bally81"        '
run routine for music etc
   END IF
   IF LightLeftOutLaneTrigger.State=BulbOn THEN
  ' Look to see if light assigned is off first
      AddScore(5000)                    ' add some points
      AddJackpot(5000)                    ' Adds to
Jackpot total
      AddBonus(1)                     ' adds to Bonus
score
     ' NOTE: You need to add messages here (or just
leave it as a default) for the displays.
```

```
        Message(1)= "SPECIAL"
        Message(2)= "SPECIAL"
        Message(3)= "SPECIAL"
        Message(4)= "SPECIAL"
        AddScoringEvent "OutLaneSpecial"          '
Automatically Awards Special. BG/display/sounds is
preset in AddEngineEvent case "SPECIAL"
   End If
End Sub

Sub RightOutLaneTrigger_Hit()                  ' The
Right OutLane trigger has been Hit
   IF LightRightOutLaneTrigger.State=BulbOff THEN
  ' Look to see if light assigned is off first
        AddScore(3000)                  ' add some points
        AddJackpot(3000)                  ' Adds to
Jackpot total
        AddBonus(1)                  ' adds to Bonus
score
        AddScoringEvent "OutLaneUnlit_bally81"        '
run routine for music etc
   END IF
   IF LightRightOutLaneTrigger.State=BulbOn THEN
  ' Look to see if light assigned is off first
        AddScore(5000)                  ' add some points
        AddJackpot(5000)                  ' Adds to
Jackpot total
        AddBonus(1)                  ' adds to Bonus
score
     ' NOTE: You need to add messages here (or just
leave it as a default) for the displays.
        Message(1)= "SPECIAL"
        Message(2)= "SPECIAL"
        Message(3)= "SPECIAL"
        Message(4)= "SPECIAL"
        AddScoringEvent "OutLaneSpecial"          '
Automatically Awards Special. BG/display/sounds is
preset in AddEngineEvent case "SPECIAL"
   End If
End Sub
```

The Outlanes, by tradition, can award a special, or free game, if either the LightLeftOutLaneTrigger or LightRightOutLaneTrigger is lit. These use a very similar code to the Inlane code, and by default, will award that special if the ball rolls over a lighted drain lane. But because you only want to award a outlane special once per ball, there is a special AddScoringEvent just for outlanes.

`code:`
```
AddScoringEvent "OutLaneSpecial"      ' Automatically Awards Special.
BG/display/sounds is preset in AddEngineEvent case "SPECIAL"
```

Changing the Display Message

You can change the message in the quotes if you wish to display a different message. Just remember that you are limited to 9 characters per display. There are 4 message(x) to display for each lane, as the message is displayed in all 4 displays.

Turning on the OutLane Lights

As part of the Beginners tutorial, we also have a example trigger that contains the code to turn on the bulbs when a ball rolls over that trigger.

```
code:  Sub TriggerOutlaneOn_Hit()
             LightLeftOutLaneTrigger.State = BulbOn
             LightRightOutLaneTrigger.State = BulbOn
           AddScore(500)                     ' Add some points.
       a bit more than before
           AddBonus (1)                     ' Add to the Bonus
       Count when the ball is lost
           AddScoringEvent "trigger"        ' did you know you
       can mix and match AddScoringEvents?
       End Sub
```

This is just a sample code, but what it does is turn on the OutLane lights, give the score for rolling over that trigger, and play some music for that particular trigger being made.

Turning off the OutLane Lights

The engine will automatically turn off the OutLane lights at a loss of a ball, and have to be re lit by the player with his next ball. The code is found in the AddEngineEvent subroutine, and in the Case "NEW_BALL". AddEngineEvent are the hooks to the main portions of the engine, and are there so you can add your own engine code or what you wish the engine to do in one place, as opposed to going through thousands of lines of code. The "NEW_BALL" case within AddEngineEvent handles all the code for the start of each new ball, just before the ball is kicked out to the plunger. In this case, the lights are "switched off" at the start of the next ball.

Note: We only include the code related to the InLane/OutLane bulbs here, and have removed the other code related to a new player(s) start of a ball.

```
code:                   ' * Code for start of each ball, and
       what the system does
          Case "NEW_BALL"                          ' Called
       From  ResetForNewPlayerBall()
           ' THIS IS WHERE YOU ADD YOUR CODE FOR THE START
       OF EACH BALL !!!!
           '  Use this to turn on any lights, reset any
       variables you use, popup targets etc.
            ' this switches off the lights used in the
       OutLanes and Outlanes
               LightLeftOutLaneTrigger.State=BulbOff
               LightRightOutLaneTrigger.State=BulbOff
               LightLeftOutLaneTrigger.State=BulbOff
               LightRightOutLaneTrigger.State=BulbOff
```

A Note

Built into fpxEngine are the ability to Alternate the Inlane or OutLane Lights. This feature will alternate a lit Inlane or Outlane light as opposed to having the 2 lights for the Inlanes and Outlanes always on. The lanes "switch" sides when ever the ball strikes a powered slingshot, or if the ball strikes a Bumper.
If you want this feature, you will find the AddScoringEvent "AddAlternatingLanes" here

Engine Code

This feature Event is part of the main engine core, and has several objects that is part of the table. These objects must be present in the table, deleting any of these objects will cause a error message when you play the table. All other objects, such as ornaments, can be safely deleted.
Objects needed

- LightLeftOutLaneTrigger,LightRightOutLaneTrigger (round light object)
- LeftOutLaneTrigger,RightOutLaneTrigger (trigger objects, set to "wire")

The Main AddScoringEvent Code

```
code: case "OutLaneUnlit_bally81"
          AddMusicSet "outlaneunlit"
         set  LastSwitchHit = LeftOutLaneTrigger
              ' remember last trigger hit by the ball, it's
      good coding, but actually pretty useless at the
      moment
      case "OutLaneSpecial"
        ' special routine for Outlanes. This repoints to
      stock special code, and switches off special lights
      at outlane
           SetEventNum=10                              ' sets
      case for TimerSetEvent1. end of the routine, which
      calls  CreateNewBall()
         IF LockDisplay=1 THEN LockDisplay=0
          ' We need to check IF there is another Event
      going on first so we override it to run this next
      code
          AddEngineEvent "SPECIAL":AddMusicSet  "SPECIAL"
         IF  constAddDebug = 1 THEN AddDebugText
      "AddScoringEvent  Special"
         IF  nvCredits <  fpxMaxCredits THEN
            fpxSpecial_Hit()
         END IF
          ' Have to remember to swich those special lights
      off!
            LightLeftOutLaneTrigger.State=BulbOff
            LightRightOutLaneTrigger.State=BulbOff
```

Note that the engine has 2 routines for AddScoringEvent, one for each light state. case "OutLaneSpecial" has a lot of additional code, it points to the main user special routine (AddEngineEvent "SPECIAL")

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature uses "LockDisplay" when a special is awarded to override any other music, lights, and display routines until the time specified is finished. (mechanical sounds will still play) While the music is playing, no other music/display/lights will be used, as this is considered a main feature award and should have priority over anything else. If the light state is off, then this feature will be overridden by any high priority feature, such as Extra Ball or another special, but the player will still receive credit if this feature is made during that time.

SpecialIsLit

SpecialIsLit
AddScoringEvent ››

Special Is Lit

Sometimes you just want to tell the player a special is lit to be collected (like at the outlanes) so fpxEngine has code just for that...

code:
```
Sub fpxMyTrigger_Hit
AddScoringEvent "SpecialIsLit"            ' Tell Player a special can be collected
End Sub
```

All this does is flash the text defined, play the music and then resets back to regular scoring afterwards. Sub TriggerOutlaneOn_Hit() in the script has a example included already, but here's a copy of that code.

code:
```
Sub TriggerOutlaneOn_Hit()
```

```
  LightLeftOutLaneTrigger.State = BulbOn
  LightRightOutLaneTrigger.State = BulbOn
  AddScore(500)                          ' Add some points. a bit more than
before
  AddBonus (1)                           ' Add to the Bonus Count when the
ball is lost
  AddScoringEvent "trigger"                  ' did you know you can mix
and match AddScoringEvents?
  AddScoringEvent "SpecialIsLit"                 ' Tell Player a special
can be collected
End Sub
```

The Engine will automatically add the display message notifying the player that a special has been lit. That code is found within the AddScoringEvent routine.

Engine Code

There is none!

25KAward

SpecialIsLit
AddScoringEvent »

25k Award

Sometimes you just want to award the player a special score for advancing past a certain point. The fpxEngine has code just for that...
This just awards 25,000 points to the players score, 25,000 is added to the Jackpot value, and 1 bonus is added to the Bonus count routine when the player loses his ball. This is a pretty common feature with the arcade games, you see variations of this code, for example the bally table "Space Invaders" plays a (very) annoying sound and turns on a light for a single Drop Target on the right side for a limited amount of time. If you manage to hit that target during the time period, you will receive a special score.

```
code: Sub fpxMyTrigger_Hit
      AddScoringEvent "25kAward"                 ' Awards a
      25,000 point Bonus
      End Sub
```

AddScoringEvent "25kAward" just awards you 25.000 points.Usually, you can have a light turn on for that so the player knows it's a special feature he should be trying to

```
make. This is used by the Vault system as well, a good example
is the Drop Target Bank Vault item.
```

Engine Code

```
AddScoringEvent "25kAward"                    ' Awards a 25,000
point Bonus
```

AddAlternatingLanes

Alternating Lanes

If You Have Never Coded Before

We are going to be dealing with working with a light to your table, and also now what the script needs to do if that light is turned off, or turned on.

I would first recommend that you quickly review the light section in the fp manual first, as it gives you a lot of very helpful information on some of the settings, and explains them quite well. Just press the F1 key, or click on "Help" at the top menu in the editor, and "Open Manual". Go all the way down to "Table Components", click on that to expand the section, and then select "Lights" and finally "Round Lights" Whew, it's actually harder to use the help file than it is to use the fpx engine so far isn't it...

Built into fpxEngine are the ability to Alternate the Inlane or OutLane Lights. The Inlane is the path feeding a falling ball from the playfield to the flippers, usually behind a slingshot. Some people refer to the Inlanes as the "flipper Return Lanes". The Outlanes are the "drain" lanes, to the side of the Inlanes. This feature will alternate a lit Inlane or Outlane light as opposed to having the 2 lights for the Inlanes and Outlanes always on. The lanes "switch" sides when ever the ball strikes a powered slingshot, or if the ball strikes a Bumper.

Hit Code

To use this feature, you need to turn on either a Inlane or Outlane light and then call the AddScoringEvent routine.
The fpxEngine will then handle the routine, as well as automatically alternate the lights when ever a slingshot or Bumper is struck by the ball.

```
code:   LightLeftInLaneTrigger.State=BulbOn   ' If this is not on already, Turns on
       a Inlane Light
        AddScoringEvent "AddAlternatingLanes"   ' Runs initial alternate light
```

code

As part of the Beginners tutorial, we also have a example trigger that contains the code to turn on the bulbs when a ball rolls over that trigger.

Pinsettings

In the user settings, you can turn "On" or "Off" the ability to have alternating Inlane or Alternating Outlane lights. By default, Alternating the lane lights are turned "on".
As well, you can set if you want those lights (if lit) to "carry over" to that players next ball, or if you require the player to start over. By default, this feature is turned "off".

code:
```
' * Pinsettings
' You can set fpx to remember which lights were On or flashing at the loss
of a ball and restore those lights at the start
' of thats player new ball. This saves these values for each player. 0 is off
(no lights at next ball) 1 is On (restores lights
' at the next ball)
' *** Altermate Inlanes/Outlanes
 fpxAlternateInLanes = 1            ' Alternate inlane light lens if on
 fpxAlternateOutLanes = 1          ' Alternate Outlane light lens if on
 fpxMemoryInlanes=0               ' save light state of Inlane Lights for players
next ball
 fpxMemoryOutLanes=0             ' save light state of Outlane Lights for
players next ball
```

Inlanes

The code needed for scoring is already within the fpxEngine template, as part of the hit sections for both the Inlanes and Outlanes. We first include the code related to the Inlane Lights here, but the Outlane lights are very similar. You can have separate scoring for the light state, it that particular Inlane Light is turned "off", then the scoring is generally a lot lower than if that light is on. In the script, we look for the Off state first, and if that light is actually turned off, we score 1000 points, add a bonus to the bonus count at the loss of a ball, and play a AddScoringEvent routine for a unlit light.
Next, if that light is actually on, we score 5000 points, add a bonus to the bonus count at the loss of a ball, and play a AddScoringEvent routine for a "lit" light.

code:
```
' Return lane to left flipper
Sub LeftInLaneTrigger_Hit()
  IF LightLeftInLaneTrigger.State=BulbOff THEN       ' Look to see if light
assigned is off first
   AddScore(1000)              ' Add some points. a bit more than before
   AddBonus (1)              ' Add to the Bonus Count when the ball is lost
   AddScoringEvent "InLaneUnlit_bally81"       ' Go to routine for music etc
  END IF                ' I'm finished telling you what to do if the bulb is off
  ' If the bulb is lit, lets reward the player with a better score
  IF LightLeftInLaneTrigger.State=BulbOn THEN       ' Look to see if light
assigned is on first
   AddScore(5000)              ' Add some points. quite a lot to make the
light on worthwhile
   AddBonus (1)              ' Add to the Bonus Count when the ball is lost
   AddScoringEvent "InLaneIslit_bally81"       ' Go to routine for music etc
  END IF                ' I'm finished telling you what to do if the bulb is on
End Sub
```

```
' Okay, lets do the other inlane now...
' Return lane to right flipper
Sub RightInLaneTrigger_Hit()
  IF LightRightInLaneTrigger.State=BulbOff THEN      ' Look to see if light
assigned is off first
   AddScore(1000)           ' Add some points. a bit more than before
   AddBonus (1)           ' Add to the Bonus Count when the ball is lost

   AddScoringEvent "InLaneUnlit_bally81"      ' Go to routine for music etc
  END IF              ' I'm finished telling you what to do if the bulb is off
  ' If the bulb is lit, lets reward the player with a better score
  IF LightRightInLaneTrigger.State=BulbOn THEN      ' Look to see if light
assigned is on first
   AddScore(5000)           ' Add some points. quite a lot to make the
light on worthwhile
   AddBonus (1)           ' Add to the Bonus Count when the ball is lost
   AddScoringEvent "InLaneIslit_bally81"      ' Go to routine for music etc
  END IF              ' I'm finished telling you what to do if the bulb is on
End Sub
```

If you need more information, this section (AddScoringEvent) also has separate pages for both Inlanes

Outlanes

The Outlanes, by tradition, can award a special, or free game, if either the LightLeftOutLaneTrigger or LightRightOutLaneTrigger is lit. These use a very similar code to the Inlane code, and by default, will award that special if the ball rolls over a lighted drain lane. But because you only want to award a outlane special once per ball, there is a special AddScoringEvent just for outlanes.

```
code:  AddScoringEvent "OutLaneSpecial"      ' Automatically Awards Special.
       BG/display/sounds is preset in AddEngineEvent case "SPECIAL"
```

This code bypasses the normal special routines, and is completely separate. It has it's own free game code, as it is used just for the Outlane specials, and needs to turn off the outlane lights.

Here is the sample code for the Hit_Events of the Left and Right Outlanes:

```
code:  Sub LeftOutLaneTrigger_Hit()
         IF LightLeftOutLaneTrigger.State=BulbOff THEN      ' Look to see if light
       assigned is off first
          AddScore(3000)           ' add some points
          AddJackpot(3000)           ' Adds to Jackpot total
          AddBonus(1)           ' adds to Bonus score
          AddScoringEvent "OutLaneUnlit_bally81"      ' run routine for music etc
         END IF
         IF LightLeftOutLaneTrigger.State=BulbOn THEN      ' Look to see if light
       assigned is off first
          AddScore(5000)           ' add some points
          AddJackpot(5000)           ' Adds to Jackpot total
          AddBonus(1)           ' adds to Bonus score
          ' NOTE: You need to add messages here (or just leave it as a default)
       for the displays.
```

```
    Message(1)= "SPECIAL"
    Message(2)= "SPECIAL"
    Message(3)= "SPECIAL"
    Message(4)= "SPECIAL"
    AddScoringEvent "OutLaneSpecial"        ' Automatically Awards
Special. BG/display/sounds is preset in AddEngineEvent case
"SPECIAL"
  End If
End Sub
Sub RightOutLaneTrigger_Hit()           ' The Right OutLane trigger has
been Hit
  IF LightRightOutLaneTrigger.State=BulbOff THEN      ' Look to see if light
assigned is off first
    AddScore(3000)             ' add some points
    AddJackpot(3000)             ' Adds to Jackpot total
    AddBonus(1)            ' adds to Bonus score
    AddScoringEvent "OutLaneUnlit_bally81"       ' run routine for music etc
  END IF
  IF LightRightOutLaneTrigger.State=BulbOn THEN      ' Look to see if light
assigned is off first
    AddScore(5000)             ' add some points
    AddJackpot(5000)             ' Adds to Jackpot total
    AddBonus(1)            ' adds to Bonus score
    ' NOTE: You need to add messages here (or just leave it as a default)
for the displays.
    Message(1)= "SPECIAL"
    Message(2)= "SPECIAL"
    Message(3)= "SPECIAL"
    Message(4)= "SPECIAL"
    AddScoringEvent "OutLaneSpecial"        ' Automatically Awards
Special. BG/display/sounds is preset in AddEngineEvent case
"SPECIAL"
  End If
End Sub
```

Like the Inlanes above, you need to turn on one of the Outlane lights as part of another Hit_event subroutine, and then start the alternating lanes routine within the AddScoringEvent system

```
code: LightLeftOutLaneTrigger.State=BulbOn   ' Turns on a Outlane Light
      AddScoringEvent "AddAlternatingLanes"   ' Runs initial alternate light
      code
```

But the fpxEngine has a additional option you can use, a special AddScoringEvent routine that announces to the player that the special is in fact now lit, with a special display/music and lighting routine. This is very useful for those situations where the game does not award a special immediately.

```
code: LightLeftOutLaneTrigger.State=BulbOn   ' Turns on a Outlane Light
      AddScoringEvent "AddAlternatingLanes"   ' Runs initial alternate light
      code
      AddScoringEvent "SpecialIsLit"   ' prompt to inform the player a special is
      lit.
```

Engine Code

This feature Event is part of the main engine core, and has several objects that is part of the table. These objects must be present in the table, deleting any of these objects will cause a error message when you play the table. All other objects, such as ornaments, can be safely deleted.
Objects needed

- LightLeftInLaneTrigger,LightRightInLaneTrigger (round light object)
- LeftInLaneTrigger,RightInLaneTrigger (trigger objects, set to "wire")
- LightLeftOutLaneTrigger,LightRightOutLaneTrigger (round light object)
- LeftOutLaneTrigger,RightOutLaneTrigger (trigger objects, set to "wire")

The Main AddScoringEvent Code

code:
```
case "AddAlternatingLanes"
 IF LightLeftInLaneTrigger.State = BulbOn OR
LightRightInLaneTrigger.State = BulbOn THEN
   If fpxAlternateInLanes = 1 THEN
    IF LightLeftInLaneTrigger.State = BulbOn THEN
     LightLeftInLaneTrigger.State=BulbOff:LightRightInLaneTrigger.State =
BulbOn
    ELSE
     LightLeftInLaneTrigger.State=BulbOn:LightRightInLaneTrigger.State =
BulbOff
    END IF
   END IF
  END IF
  IF LightLeftOutLaneTrigger.State = BulbOn OR
LightRightOutLaneTrigger.State = BulbOn THEN
   IF fpxAlternateOutLanes = 1 THEN
   ' run check so inlane and outlane lights will be on different sides. Set the
initial light to the same side as the outlane
    IF LightLeftInLaneTrigger.State = BulbOn THEN
LightLeftOutLaneTrigger.State=BulbOn:LightRightOutLaneTrigger.State =
BulbOff:END IF
    IF LightRightInLaneTrigger.State = BulbOn THEN
LightLeftOutLaneTrigger.State=BulbOff:LightRightOutLaneTrigger.State =
BulbOn:END IF
    ' Then switch that light to the other side
    IF LightLeftOutLaneTrigger.State = BulbOn THEN
     LightLeftOutLaneTrigger.State=BulbOff:LightRightOutLaneTrigger.State
= BulbOn
    ELSE
     LightLeftOutLaneTrigger.State=BulbOn:LightRightOutLaneTrigger.State
= BulbOff
    END IF
   END IF
  END IF
```

Note that the engine has 2 routines within this AddScoringEvent, one for each light state. As well, the one AddScoringEvent handles both Inlanes and Outlanes at the same time, and was written so if a Inlane is lit, and a Outlane is also lit, then those lights are set to be on oppose sides.

Music/Light/Display Code

This feature uses presets, so you do not have to change any music, light, or display code. Because fpxEngine supports different music sets, the routines are different based on the company era, the music is different, and the lightseq and type of flashing display may be different between the various music sets. All routines are based on the interval (the total time of the main music file in milliseconds) when this feature is made in game.

NOTE: This feature will be overridden by any high priority feature, such as Extra Ball or a special, but the player will still receive credit if this feature is made during that time.

Pages still being written

Pages still being written

Manual pages still in progress

The pages within this section are being worked on so they can be a bit rough. As I go along, I update the information in these pages and when the pages are finished, moved to the regular sections within the manual.

fpxAdvanceScore

fpxAdvanceScore
Pages still being written »

fpxAdvance Score

A very common scoring feature (especially on Bally Solid State games) in which a ball rolls over a trigger, or hits a target, and the player is awarded a increasing score depending on how many times the player has hit that target. Bally (and other companies) usually had the advance score feature give a increasing amount of points (most notably Centaur and Embryon) but they also used this feature to increase the Bonus Multiplier, or light additional scoring features like "turn on" the Inlane lights.

The fpxEngine example uses a increasing value, with awarding a extra ball and special if the player advances the lit value high enough.

How it works

In the vault item, we use the design from eightball. A ball sent over the Star Rollover advances the lite value, and awards the points based on the Lens Light Chain. It then increases the next value and turns on the appropriate light for that value. You can have up to 10 separate values (depending on the setting within the script) and is based on stock Bally code that starts at 10,000 points, and increases the points by 10,000 to 50,000, a extra ball,a special, and then 50,000 points. The additional target when hit will add a bonus to the advance score as well. The Star Rollover will also flash the bulb lights for a set time as well (why not?)

This script also has a "pin setting" that is adjustable, to either have no lens lights on, and a rollover awards 5000 points and then lights the first lens light for 10,000, or you can have the engine light any lens light as it's starting value when ever a new ball is created.

You can also set the code to reset the Advance Value back to it's initial first state if the player(s) loses a ball and then starts a new ball in play, or you can "carry over" that value to his next ball. The script will store this value by each player, and "remember" what that player made in his previous ball and then restore that value with his next ball.

How to use this Vault item in your fpx table.

(not ready to tell you yet)

Modifying the code for your table

Arcade pinball machines have "pin settings", adjustable settings that allow the operator to fine tune the difficulty level of the game. fpxEngine has several settings built in to emulate those pin settings, and are based on the most common settings. Because of FP's limitations in the amount of settings that can be saved when a table is first loaded from the editor, fpx instead has these settings at the very top of the Vault script as part of the Hit Code section.

The big advantage to this is these settings are "script-able" which means you can turn on or off within your custom code. The actual code is just like pretty much everything within fpx, is a single statement, and all pin settings built within fpx are just 2 values, either a "0" (zero) to switch a feature Off, or a "1" to switch a feature on.

- **vfpxAVOn = 1** - This turns on this Vault item, so any code within this Vault item will execute when added to the fpxEngine core.
- **vfpxAVCaseStart = 1** - a pin setting, this tells the script which light and the value associated with that light to turn on first at the start of a game or with the start of every new ball.
- **vfpxMemory = 1** - Instead of starting over from scratch at the start of a player(s) new ball, this will remember the last state of Advance Value at the loss of a ball, and restore that value when that player starts a new ball in play. This will store separate states for 4 players

  AddUserfpxAV()

The very nature of the fpxEngine is to allow absolute beginners to build and create complex tables with as little actual experience as possible, but you still need to learn a tiny bit of coding sometimes. Most of the Vault items will be based on the actual arcade table it came from, so it's just a quick copy and paste the design elements, the code, and then adding lights in the LightList manager. This Vault item though is the Master template I use to create those items from, so if you decide to use this vault item, you can change the scoring quite easily by using the AddUserfpxAV() subroutine.

How the code works is once a Hit event is triggered, the code goes to AddfpxAV(), which handles all the hard stuff, like lights and sound. AddfpxAV() then will send the script to look at AddUserfpxAV() for the actual scoring. In this example, the score increases by 10000 usually, awards a extra ball and a special, and then adds 50,000 points afterwards.

code:
```
' Allows you to set the scoring. Everything else is
done for you within the engine. You can add/change
anything you want
' like a multiplier or turning on Inlane lights etc.
See Beginners Guide in manual
Sub AddUserfpxAV()
 Select Case vfpxAVCase
  Case 0
    AddScore(1000):AddJackpot(1000)
  Case 1
    AddScore(5000):AddJackpot(5000)
  Case 2
    AddScore(10000):AddJackpot(10000)
  Case 3
    AddScore(20000):AddJackpot(20000)
  Case 4
    AddScore(30000):AddJackpot(30000)
  Case 5
    AddScore(40000):AddJackpot(40000)
  Case 6
    AddScore(50000):AddJackpot(50000)
  Case 7
    AddScoringEvent "ExtraBall"
  Case 8
     fpxSpecial_Hit()
  Case 9
    AddScore(50000):AddJackpot(50000)
 End Select
End Sub
```

If you went through the Beginners Guide, you see it's pretty basic code, and is very easy to change or to add to. You can make the score anything you want, and you can even add the preset features from AddScoringEvent(), like turning on the inlanes, or increasing the Bonus Multiplier. Lets add a example (if you wish to use this instead, you can, just copy this code and delete the code in the script and replace it with this one)

code:
```
' Allows you to set the scoring. Everything else is
```

```
done for you within the engine. You can add/change
anything you want
' like a multiplier or turning on Inlane lights etc.
See Beginners Guide in manual
Sub AddUserfpxAV()
AddDebugText "AddUserfpxAV(): vfpxAVCase =
" & (vfpxAVCase)
' Normally, we put the inlane/outlane lights in case
2, but fpx will turn those lights off at the loss of
a ball,
' and not turn them back on again,
' so what we do is see if the case is 2 or more, and
then turn those lights back on on the next trigger
rollover when a
' new ball is in play.
 IF vfpxAVCase>1 THEN
    LightLeftInLaneTriggerState = BulbOn
    LightRightInLaneTriggerState = BulbOn
 END IF
 Select Case vfpxAVCase
  Case 0
    AddScore(1000):AddJackpot(1000)
  Case 1
    AddScore(5000):AddJackpot(5000)
  Case 2
    AddScore(5000):AddJackpot(5000)
 AddDebugText "    - Inlanes Lit"
  Case 3
    AddScore(5000):AddJackpot(5000)
    Message(1)= "BONUS   "
    Message(2)= "MULTI   "
    AddScoringEvent "AddMultiplier"
 AddDebugText "    - Bonus Multiplier"
  Case 4
    Message(1)= "Mystery  "
    Message(2)= "Mystery  "
    AddScoringEvent "Mystery"
 AddDebugText "    - Mystery"
  Case 5
    AddScore(25000):AddJackpot(25000)
 AddDebugText "    - 25,000"
  Case 6
    Message(1)= "JACKPOT"
    Message(3)= "JACKPOT"
    AddScoringEvent "Jackpot"
 AddDebugText "    - Jackpot"
  Case 7
    AddScore(5000):AddJackpot(5000)
    AddScoringEvent "ExtraBall"
 AddDebugText "    - Extra Ball"
```

```
    Case 8
      AddScore(5000):AddJackpot(5000)
        LightLeftOutLaneTrigger.State = BulbOn
        LightRightOutLaneTrigger.State = BulbOn
 Message(1)= "SPECIAL":Message(2)= " IS LIT
":Message(3)= "SPECIAL":Message(4)= " IS LIT " ' Add
Messages for the display
  AddScoringEvent "SpecialIsLit"                     '
Tell Player a special can be collected
  AddDebugText "    - Outlanes Lit"
' Just for me actually...
    Case 9
      Message(1)= " SUPER   "
      Message(2)= "JACKPOT  "
      AddScoringEvent "SuperJackpot"
  AddDebugText "    - Super Jackpot"
 End Select
End Sub
```

Default scoring is 5000 points for this example, but with several things, we don't need to add the default scoring, as features like the Mystery have their own scoring.

- Case 2 would normally turn on the Inlane Lights. Instead, we tell the script to turn on the lights every time the ball rolls over the trigger, and `vfpxAVCase` is a value of 2 or greater. fpx will automatically turn off those lights at the loss of a ball, so you would want to turn them back on at the start of a new ball and the ball rolls over the trigger during that next ball.
- Case 3 adds a multiplier. You do have message code for this, so you need to have it in. This advance the multiplier by one.
- Case 4 awards a Mystery which is a random score. You do have message code for this, so you need to have it in.
- Case 5 awards 25000 points (common with Bally games) There are no special routines like AddScoringEvent or AddMusicSet, this just adds 25,000 points.
- Case 6 awards a Jackpot. We removed the AddScore code, as we are getting points from a super jackpot instead. You do have message code for this, so you need to have it in
- Case 8 we replace the special feature that will automatically award a special and instead light the outlanes instead to award a special. We also add a 'prompt'' message to tell the player there's a special at the Outlanes to be rewarded with `AddScoringEvent "SpecialIsLit"`)
- Case 9 awards a Super Jackpot. We removed the AddScore code, as we are getting points from a super jackpot instead. You do have message code for this, so you need to have it in

  Of course, as the engine matures, there will be a lot more preset features added to AddScoringEvent, this is just a example. Advanced coders can of course add their own code.

  The two Rubbers...

  In the eightball design, there are two rubbers, one at the top to form the right side of the guide lanes (RubberfpxAV1) and a second rubber (RubberfpxAV2) just below the single target to form the left side of the loop, and to give some bounce for the Bumpers. Both rubbers have single leaf switches included, as the Bally arcade table scored 10 points on a hit to either rubbers. This code is not included within this vault item, but you can add it in if you wish. These just add 10 points to your score. Down the road, when support is added in, you can also alternate the Inlane and Outlane lights as well.

```
code: Sub RubberfpxAV1_Hit()
      AddScore (10)
      AddScoringEvent "InLaneUnlit_bally81"
      End Sub
      Sub RubberfpxAV2_Hit()
      AddScore (10)
      AddScoringEvent "InLaneUnlit_bally81"
      End Sub
```

Note that we just use the same code for a unlit Inlane rollover, (`AddScoringEvent "InLaneUnlit_bally81"`) as it has the music and mechanical sound already in place. Most Solid State games had a very limited sound and music set, so the sounds were reused for multiple scoring objects

List of Objects used for this Vault Item

A Note To Coders

Even though this Vault item comes complete with a design based on a actual pinball game, you can remove the excess objects in your editor and use the scripted objects within a new or different design. If though you remove/delete a scripted object from the editor and then run the game, you will get a error, so you have to keep all the scripted objects in the editor.

Scoring objects

- TriggerfpxAV (scores Advance Bonus)
- TargetfpxAV (secondary target, lights TriggerfpxAV to add bonus

Light lens

- LightfpxAVsr (used as a star rollover with TriggerfpxAV)
- LightfpxAVtarg (target lens light)
- LightfpxAV1, LightfpxAV2, LightfpxAV3, LightfpxAV4, LightfpxAV5, LightfpxAVeb, LightfpxAVsp (light lens to show the present Advance Score value, from 10,000 to 50,000, extra ball and Special
- BulbfpxAV1, BulbfpxAV2, BulbfpxAV3, BulbfpxAV4 - play field bulbs
- RubberfpxAV1, RubberfpxAV2 - rubbers with leaf switches to score points. NOTE: these are not scripted, you have to add the code yourself (see above) so you can safely delete these rubbers if you don't want to use them.

Variables

- **vfpxAVOn**

vfpxAVOn is mostly used for the engine, and is used to prevent errors in game play if there are objects missing. By default as a vault item, vfpxAVOn is set to be active (vfpxAVOn=1). This is strictly fpxEngine code, when vfpxAVOn is set to), or it's not active or not to be used, the engine will bypass any routines within the engine core, and any errors as well if you decide to not use this vault item. You can though, by adding in your own code, switch this vault item on or off, just don't forget to change it back afterwards.

- **vfpxAVCaseStart**
  vfpxAVCaseStart is the main variable used to determine which lens light is turned on, and what will score with that light. A very common arcade pin setting was the ability for the operator to set whether the first lens light in the chain would be turned on at the start of a game or with each new ball, or no light and require the player to make a additional shot to start the light chain (Bally's Star Trek is a perfect example). vfpxAVCaseStart though can start at ANY value, so having vfpxAVCaseStart=2 means the 20k light will be lite, or vfpxAVCaseStart=5 means the 50k light will be lite. Again, this can be script_able within your custom code, like having the extra ball light in the chain lit on the last ball if the player doesn't do very well in his game. Case advances by 1 every time till maxed out at 9, which then constantly repeats case 9 until it is reset.

- **vfpxMemory**
  Another very common feature with pin settings was the ability to have previously made shots "carry over" to the next players ball when it becomes his turn to play. By setting vfpxMemory=1, this feature will store the Advance Value shots made, and restore those values for each player that each player had made with their next ball. Setting this to) will automatically set the Advance Bonus to it's beginning game starting points (vfpxAVCaseStart) and for every ball, no matter how many players are playing a game.

- **vfpxAVCase**
  The name used for the Select Case in ClosefpxAV(). This handles the light and bulb routines when fpxAdvanceScore is reset or it's initial starting values when a game is started.

- **vfpxAVLights(9,9),vfpxAVBulbs(9,9)**
  A simple coding trick, used with `vfpfAVLightCount` and `vfpx AVBulbCount` to handle turning on or off all the lights or bulbs using a for/next loop. The big advantage is it's automated, and you can add more or less lights if you need to, as you use the SetLight statements, and both variables are just the total number for lights and bulbs. This example will switch off all the lens lights, with vfpxAVLightCount set to 9

```
code: FOR x = 1 TO (vfpxAVLightCount)
      vfpxAVLights(1,x).State=BulbOff
      NEXT
```

You will notice 2 numeric values are defined, the first is for groups, the second number is for the number of the light/bulb within that group, so you can have 10 groups (0 to 9) each with 10 lights or bulbs in them. This is really more for future Vault items.

- **vfpxAVLightCount**
  Is the total amount of lights. This is used mostly for for/next code, as it makes adding and deleting more lights very easy. The lens lights are defined within a SetLight

routine (vfpxAVLights)

- **vfpxAVBulbCount**
  Is the total amount of playfield bulbs. This is used mostly for for/next code, as it makes adding and deleting more lights very easy. The bulbs are defined within a SetLight routine (vfpxAVBulbs)

- **fpxAVp1,fpxAVp2,fpxAVp3,fpxAVp4**
  These are simple memory variables for each player, that holds the amount of advances a player has made. There is one for each player, and each one is seperate from the others. We generally just use this for multiplayer games, and only if vfpxMemory=1

  Subroutines

- **TriggerfpxAV_Hit()** - Hit event - scores Advance Value
- **AddUserfpxAV()** - The scoring routine completely adjustable. Allows you to set the scoring to what you want.
- **TargetfpxAV_Hit()** - Secondary target adds bonus to TriggerfpxAV
- **AddfpxAV()** - Main Advance Value Routine
- **ClosefpxAV()** - handles the lights and bulbs during advance value, and is also used to reset the lights at game start/new ball/game end
- **fpxResetVaultBallReset()** - reset routine for each new ball in play
- **fpxResetVaultGameReset()** - resets routine so start of each game, clears main variables
- **vfpxAVMemorySave()** - at loss of ball, saves to player memory
- **vfpxAVMemoryLoad()** - at start of a ball, restores any collected values and lights for that player (if enabled and only if vfpxMemory=1)

Hooking up to the Engine

NOTE: for the time being, we use AddEngineEvent() to hook up code, and not "hard coded" directly in the main engine script. This depends on the various Vault scripts, and how much coding is needed in the future to have them automatically recognized in your Hit Code.

You need to tell the script what to do at the start of a game, a new ball, and at the end of a game, so for this we use `AddEngineEvent()` and it's preset hooks as it's a lot simpler and generally fool proof to use as opposed to going through hundreds of lines of very advanced code. (If you need to review AddEngineEvent again, you will find a good basic tutorial in the Beginners Guide). It's actually pretty simple to add, only 3 calls, one to Start a game, one for each ball, and one for the end of game. The code is already in for you, this example is for those who wish to use this code in non-fpx tables or to translate for the vp editor.

This example is just the case settings and the single line to point to this routine, with everything else stripped out.

```
code:  Sub AddEngineEvent(SetEvent)
       Select Case SetEvent
```

```
' * Before the first ball pops out when a game is
started
Case "START_GAME"
' * VAULT
fpxResetVaultGameReset()  ' resets all Vault Items
by game

' * Code for start of each ball, and what the system
does
Case "NEW_BALL"
' * Vault
fpxResetVaultBallReset()  ' Vault code to restore
all vault items to initial defaults or saved memory
state

' * When all balls done, match over and the game is
finished.
Case "END_GAME"
' * Vault
fpxResetVaultGameReset()  'Reset vault items

End Select
End Sub
```

So `fpxResetVaultGameReset()` needs to be added at the start and end of a game, and `fpxResetVaultBallReset()` needs to be added for when a new ball is started. If you use this for other tables, you may have to adjust accordingly and add one of these 2 subroutine calls to other areas like tilt restore etc.

Code

Copy of basic Advance Scoring code

code:

```
' *** fpx Vault Advance Score ***
' * User adjustable and scriptable settings
 ' Turns on or off this vault item.If set to 0 then
this code is disabled.
 ' Set to 1 to use this vault item complete with
code and objects
 vfpxAVOn=1
 ' Initial light to be lit in vfpxAVLights for start
of each ball
 ' Set to 1 to start at 5000pts, 2 to start at
10000pts etc (9 max)
  vfpxAVCaseStart = 1
 ' Handles the Memory feature for Advance Score
(common pin setting)
 ' Set to one if you want the players Advance Score
carried over to his next ball.
```

```
 ' Set to 0 to have Advance Score reset back to
beginning with each new ball.
 vfpxMemory=1' Variable to hold Advance Score Value
from ball to Ball (for each player)
' Star Rollover trigger advances lit value
Sub TriggerfpxAV_Hit()
  PlaySound "trigger"
 If vfpxAVOn=1 THEN                ' Only if this
feature is set to "1" and nothing else
   AddfpxAV()
  If LightfpxAVsr.State=BulbOn THEN AddBonus(1)
 END IF
 set  LastSwitchHit = TriggerfpxAV
End Sub
' Allows you to set the scoring. Everything else is
done for you within the engine. You can add/change
anything you want
' like a multiplier or turning on Inlane lights etc.
See Beginners Guide in manual
Sub AddUserfpxAV()
 Select Case vfpxAVCase
  Case 0
    AddScore(1000):AddJackpot(1000)
  Case 1
    AddScore(5000):AddJackpot(5000)
  Case 2
    AddScore(10000):AddJackpot(10000)
  Case 3
    AddScore(20000):AddJackpot(20000)
  Case 4
    AddScore(30000):AddJackpot(30000)
  Case 5
    AddScore(40000):AddJackpot(40000)
  Case 6
    AddScore(50000):AddJackpot(50000)
  Case 7
    AddScoringEvent "ExtraBall"
  Case 8
     fpxSpecial_Hit()
  Case 9
    AddScore(50000):AddJackpot(50000)
 End Select
End Sub

' Additional Target scores points, lits TriggerfpxAV
for bonus
Sub TargetfpxAV_Hit()
 IF (fpTilted
= TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 If vfpxAVOn=1 THEN                ' Only if this
```

```
feature is set to "1" and nothing else
  IF LightfpxAVtarg.State=BulbOff THEN
    AddScore(5000):AddJackpot(5000):AddBonus
(1):AddMusicSet "trigger"
  ELSE
    AddScore(1000):AddJackpot(1000):AddBonus
(1):AddMusicSet "inlaneislit"
  END IF

LightfpxAVtarg
.State=BulbOn:LightfpxAVsr.State=BulbOn
 END IF
 set  LastSwitchHit = TargetfpxAV
End Sub
' *** Engine Code DO NOT CHANGE, DO NOT DELETE OR
CHANGE  ***
'
*****************************************************
*****************
' **                              Vault
                **
'
*****************************************************
*****************
' *** fpxAddValue ***
Dim vfpxAVOn,vfpxAVCaseStart,vfpxMemory
 ' User adjustable settings (pin settings)
Dim vfpxAVCase,vfpxAVLights(9,9),vfpxAVBulbs(9,9)
       ' control case for Advance Value.
Dim vfpxAVLightCount,vfpxAVBulbCount
' Used in for/next loops so you can have as many
lights as you want with changing the script.
Dim fpxAVp1,fpxAVp2,fpxAVp3,fpxAVp4                '
variables to hold aV for each player (vfpxMemory)
' Single Lens Lights
Set vfpxAVLights(1,1)
=LightfpxAV1:Set vfpxAVLights(1,2)
=LightfpxAV2:Set vfpxAVLights(1,3)
=LightfpxAV3:Set vfpxAVLights(1,4)=LightfpxAV4
Set vfpxAVLights(1,5)
=LightfpxAV5:Set vfpxAVLights(1,6)
=LightfpxAVeb:Set vfpxAVLights(1,7)=LightfpxAVsp
' bulbs
Set vfpxAVBulbs(1,1)=BulbfpxAV1:Set vfpxAVBulbs(1,2)
=BulbfpxAV2:Set vfpxAVBulbs(1,3)=BulbfpxAV3
Set vfpxAVBulbs(1,4)=BulbfpxAV4
' Total amount of bulbs/lights in this group (used
for scriptwide For/Next loops)
vfpxAVLightCount = 7:vfpxAVBulbCount = 4
' Main scoring routine.
' On trigger rollover, vfpxAVCase increases by one,
then runs that number in the matching case
Sub AddfpxAV()
 IF (fpTilted
= TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 If vfpxAVOn=1 THEN                   ' Only if
this feature is set to "1" and nothing else
  FOR x = 1
To

(vfpxAVLightCount):vfpxAVLights
(1,x).State=BulbOff:NEXT   ' All lens lights off
first
   vfpxAVCase = vfpxAVCase+1                   '
Increases value by 1 for select case routines
  IF  vfpxAVCase > 9 THEN  vfpxAVCase = 9:END IF
     ' Prevents case from going beyond 9, because no
scoring
   TimerCloseScoringEventCase = 1:LockDisplay=0
     ' Points to the closing routines after the
timers are done
  Select Case vfpxAVCase
   Case 0                     ' Error catcher
     AddMusicSet "guidetrigger"
    ' Flash bulbs. This uses the timer interval in
AddMusicScore and resets to on afterwards
    FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
   Case 1                  ' No light on, scores 5000
     AddMusicSet "guidetrigger"
    FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
   Case 2                  ' scores 10000
     AddMusicSet
"guidetrigger"
:DisplayBlinkInterval=(FlashForMSBlinkInterval+40)
     vfpxAVLights(1,1).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval),BulbOn
    ' Need to put code here as it uses the
musicInterval timer from AddMusicSet
    FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
   Case 3
     AddMusicSet
"guidetrigger"
```

```
:DisplayBlinkInterval=(FlashForMSBlinkInterval+30)
     vfpxAVLights(1,2).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval),BulbOn
    FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
   Case 4
     AddMusicSet
"guidetrigger"
:DisplayBlinkInterval=(FlashForMSBlinkInterval+20)
     vfpxAVLights(1,3).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval),BulbOn
    FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
   Case 5
     AddMusicSet
"guidetrigger"
:DisplayBlinkInterval=(FlashForMSBlinkInterval+10)
     vfpxAVLights(1,4).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval),BulbOn
    FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
   Case 6
     AddMusicSet
"guidetrigger"
:DisplayBlinkInterval=(FlashForMSBlinkInterval)
     vfpxAVLights(1,5).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval),BulbOn
    FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
   Case 7
     FlushDisplay()                          ' Clear display
or no messages will appear
     Message(1)= "EXTRA    ":Message(2)= "BALL
":Message(3)= "EXTRA    ":Message(4)= "BALL     "
    FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
   Case 8
     FlushDisplay()                          ' Clear display
or no messages will appear
     vfpxAVLights(1,5).State=BulbOn
    FOR x = 1
```

```
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
   Case 9
     AddMusicSet
"guidetrigger"
:DisplayBlinkInterval=(FlashForMSBlinkInterval)
     vfpxAVLights(1,5).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval),BulbOn
    FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).FlashForMs
(MusicIntervalTime),
 (FlashForMSBlinkInterval), BulbOn:NEXT
  End Select
     AddUserfpxAV()                           ' Goto scoring
code as set by developer
     vfpxAVMemorySave()                         ' Save
vfpxAVCase for this player if vfpxMemory=1
 ELSE                            ' This Vault item is
disabled so just leave the subroutine
  Exit Sub
 END IF
End Sub
' runs the correct light routine and restores proper
light if vfpxMemory=1
Sub ClosefpxAV()
AddDebugText "ClosefpxAV() "
 If vfpxAVOn=1 THEN                         ' Only if
this feature is set to "1" and nothing else
  FOR x = 1
To (vfpxAVLightCount): vfpxAVLights(1,x).State
= BulbOff:NEXT  ' we turn off all the lights first
before we update the scoring
  FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).State
= BulbOn:NEXT
  Select Case vfpxAVCase                        ' Sets
next light to display
   Case 0: Exit Sub
   Case 1:vfpxAVLights(1,1).State
= BulbOn:vfpxAVLights(1,1).FlashForMs
(MusicIntervalTime
/2), (FlashForMSBlinkInterval),BulbOn  ' 1k, light
10k light for next case
   Case 2:vfpxAVLights(1,2).State
= BulbOn:vfpxAVLights(1,2).FlashForMs
(MusicIntervalTime
/2), (FlashForMSBlinkInterval),BulbOn  ' 10
   Case 3:vfpxAVLights(1,3).State
= BulbOn:vfpxAVLights(1,3).FlashForMs
```

```
(MusicIntervalTime
/2), (FlashForMSBlinkInterval),BulbOn   ' 20
   Case 4:vfpxAVLights(1,4).State
= BulbOn:vfpxAVLights(1,4).FlashForMs
(MusicIntervalTime
/2), (FlashForMSBlinkInterval),BulbOn   ' 30
   Case 5:vfpxAVLights(1,5).State
= BulbOn:vfpxAVLights(1,5).FlashForMs
(MusicIntervalTime
/2), (FlashForMSBlinkInterval),BulbOn   ' 40
   Case 6:vfpxAVLights(1,6).State
= BulbOn:vfpxAVLights(1,6).FlashForMs
(MusicIntervalTime
/2), (FlashForMSBlinkInterval),BulbOn   ' 50
   Case 7:vfpxAVLights(1,7).State
= BulbOn:vfpxAVLights(1,7).FlashForMs
(MusicIntervalTime
/2), (FlashForMSBlinkInterval),BulbOn   ' Extra Ball
   Case 8:vfpxAVLights(1,5).State
= BulbOn:vfpxAVLights(1,5).FlashForMs
(MusicIntervalTime
/2), (FlashForMSBlinkInterval),BulbOn   ' Special
   Case 9:vfpxAVLights(1,5).State=BulbOn ' No need
to add blinking light effect here as done in
AddfpxAV()
  END SELECT
 END IF
End Sub
' Blanket reset used for tilt, new ball, Startup or
game over
Sub fpxResetVaultBallReset()
AddDebugText "fpxResetVaultBallReset()  "
 ' Advance Count
 IF vfpxAVOn=1 THEN
  If vfpxMemory=1 Then                    ' Look to
see if player memory feature is on
     vfpxAVMemoryLoad()                 ' Load in
last vfpxAVCaseStart made from loss of previous ball
  ELSE
    vfpxAVCase=(vfpxAVCaseStart)                 '
reset vfpxAVCase back to the beginning
  END IF
  ' fpx will use the last interval time played, so
we need to set the interval time here instead.
    MusicIntervalTime
= 250
:ClosefpxAV():LightfpxAVsr
.State=BulbOff:LightfpxAVtarg.State=BulbOff

 END IF
End Sub
```

```
' runs at game start and game end.
' This resets the variables or the next game will
continue on with the scoring made from previous game
Sub fpxResetVaultGameReset()
 IF vfpxAVOn=1 THEN
  FOR x = 1
To (vfpxAVLightCount): vfpxAVLights(1,x).State
= BulbOff:NEXT  ' we turn off all the lights first
before we update the scoring
  FOR x = 1
TO vfpxAVBulbCount:vfpxAVBulbs(1,x).State
= BulbOn:NEXT

vfpxAVCase=(vfpxAVCaseStart):fpxAVp1=(vfpxAVCaseStar
t):fpxAVp2=(vfpxAVCaseStart):fpxAVp3=(vfpxAVCaseStar
t):fpxAVp4=(vfpxAVCaseStart)
 END IF
End Sub
' Saves the vfpxAVCase settings for each player at
the loss of a ball if vfpxMemory=1
Sub vfpxAVMemorySave()
 IF vfpxAVOn=1 THEN
  Select Case CurrentPlayer
   Case 1:fpxAVp1=vfpxAVCase
   Case 2:fpxAVp2=vfpxAVCase
   Case 3:fpxAVp3=vfpxAVCase
   Case 4:fpxAVp4=vfpxAVCase
  End Select
 END IF
End Sub
' Loads the vfpxAVCase settings for each player at
the start of a ball, and restores that value back on
his next ball if vfpxMemory=1
Sub vfpxAVMemoryLoad()
 IF vfpxAVOn=1 THEN
  Select Case CurrentPlayer
   Case 1:vfpxAVCase=fpxAVp1
   Case 2:vfpxAVCase=fpxAVp2
   Case 3:vfpxAVCase=fpxAVp3
   Case 4:vfpxAVCase=fpxAVp4
  End Select
 END IF
end sub
```

More fpxEngine Presets

More fpxEngine Presets
Pages still being written »

More fpxEngine Presets

The Vault

The Vault

The Vault system is the true magic behind the fpxEngine. These are a collection of designed and coded templates of small design elements that you can use within your table design to build a pinball table from scratch. All you need to do is select which Vault templates you wish to use, copy and paste the design and the script into the master template, and you are pretty much done. A average simple design should take about 1/2 hour to do, and it will work as soon as you press play. As time goes on, more and more vault templates will be added.

Each vault page in this manual shows a image for how that vault will look, and gives a complete description of what the vault does, how it scores, and how to modify the vault for your own set of rules. Also incuded is a copy of the master script, so you can study how it all works and reuse the master script (refered to as "worksheets") for your own code, or to duplicate.

- How to use a vault item
- How to use the worksheets, complete with a tutorial for advanced coders.

A Note To Coders

The Vault is a very powerful system, it is recommended that you go through the Using the Vault Worksheets page in this manual before you modify or make your own vault items. This details all the variables and subroutines used for the vault, how to name those, as well as information on a simple way to hook them up into the fpxEngine.

This section details all the individual Vault items, and explains what they do. This also gives you the information you need to change any "pinsettings", as well as a description of the main scoring subroutines and a list of the objects needed for the script. Everything related to these features will be placed in their page, including (in future versions) alternate code examples, additional commands you can use, and sections on modifying the code to do more advanced routines. As well, with future releases of fpxEngine, new scoring features, such as multiball or new features written for the vault will be added.

Vault Items

Drop targets	**StandUp targets**	**Triggers**
• vault_fpxDropTargetBank1 • vault_fpxDropTargetBank2	• vault_fpxTarget3Bank1 • vault_fpxTarget3Bank2	• vault_fpxAV1 (Advance value)
Kickers	**Plastics and Spare Parts**	
• vault_fpxKicker1	• vault_fpxPlastics1	

How to use the Vault

Vault - How To use the Vault Items

Vault - Using the Vault items

How to use a Vault Item in the fpxEngine

Using this Vault item and including it in your table design is very simple. It's just a couple copy and pastes from one fpt file to another!

- Open up the fpxEngine Template fpt file in your Future Pinball Folder
- Open up and select the Vault item fpt file from the Vault folder included with the fpxEngine program
- Make sure **ALL** the layers are visible in the editor. Go to the top menu, Select "***Edit***",then select "***Select All***". All the table elements should highlight
- Then Select "***Edit***" again, and click "***Copy***". Select "***Edit***",Then select "***Select All***"
- Go back to the fpxEngine, go to the top menu, Select "***Edit***",then select "***Paste***". All the table elements should be added to your design.
- Go back to the Vault fpt file, Click "***Script***" on the left hand menu bar. The script Editor will now appear. Click on the first empty line, and while holding down your Left Mouse Button, move the mouse downwards till the entire text within the script is highlighted.
- Click the Right Mouse Button, and select "***Copy***" from the drop down menu. Return to the fpxEngine template, open up the script editor.
- Find in the script where the Hit Section Begins, put your mouse cursor on a EMPTY line below, right click your mouse and select "***Paste***". The Vault script is now in in your fpxEngine.
- You need to add any lights,plastics and bulbs to your LightList Manager in your fpx table.
    - In the fpxEngine Template, go to the top menu, select "**Table**" and then select " ***LightList Manager***". A small popup box will appear
    - Look for the names listed on the left side in the LightList Manager to have

"***light***", "***bulb***" or "***plastic***" as fpx will use these words at the beginning of the object name for all Vault Items.

- Select the light/bulbs/plastic file names, and then press the Include button. Those items you selected will transfer over to the right hand side (you need to scroll down, the lights will be at the bottom of that list.)
- Once you have selected all the lights and bulbs and transferred them over, select "***Okay***", and then "***Okay***" again.

- Press Play!
- If you do not know how to transfer the lights, the Future Pinball manual explains how to do this. Also, visit the fpxEngine YouTube for demonstrations

Using the Vault Worksheets

Vault - Using the Vault items

Where the Vault Worksheet Templates are

Vault worksheets (the master copy of the code I use to create Vault files) can be found in the fpxEngine main folder as a subfolder called Vault Worksheets. These worksheets filenames will be either in .txt or .vbs.
You will need a good text editor to modify these worksheets, as everything within the sheets can be changed with a simple find and replace.
NEVER use the FP script editor to do any main modifications to a worksheet, the find/replace command is very unreliable, and too simplistic to use. I recommend Notepad++, a free version is available and gives you far more capabilities, like color coding by script language and the find/replace is first rate.
You can get a copy here at the Notepad++ website

Introduction

In developing the Vault system for the fpxEngine, a script template is used to create the main code for each Vault item, and then modifications are made to that code depending on what objects are used, and scoring features. As easy as using the Vault system is for nearly everyone, coding on the other hand is very time consuming and can take days of writing the code, testing and debugging the code, and even writing a manual page for it. The template worksheets are just a way for me to create Vault items in a fast manner while removing a lot of hours of work per Vault on my end.

Still, even though in time, you can have a unlimited amount of vault items, some people who are a bit more comfortable with coding will want to customize or even create their own vault files, so this page details the basic structure of the Vault worksheets, and how to change some settings, like adding more case settings, that go beyond what a typical user with no coding experience can do. These worksheets are very expandable, and extremely flexible, yet still retain their ease of use for even the most beginner of coder.

A NOTE about this section

Because there will be so many vault items down the road, all with different table names and objects used, we are using **fpxKicker1** as our code example. fpx is the "table name" and Kicker1 is our "object name". With our main variables listing though, this might cause confusion, so we instead use (table name)(object name) as opposed to a existing vault item name. If you are duplicating or modifying a vault worksheet, using the find and replace method above will also, and automatically, change the main variables to your table and object name

The Keywords

The worksheets were carefully written to make duplication of the worksheets as simple to use and as fast to change as possible, by using set keywords within the variable, subroutine, and object names.
This allows you to select a keyword, and with a simple find/replace, change it on mass so your new vault item will not cause conflicts with other Vault items.
There are 3 keywords used within the worksheet template:

- **Table vault name keyword** - This is the main vault name, and is used for vault items based on a actual table. For the basic vault example code we use the name "*fpx*" as our "*(table name)"*
- **Object vault name keyword** - This is the main type of object we use. For the basic vault example code we use the name "*Kicker1*" as our "*(object name)*" I usually add a number behind the object name, and change that number if I want a second duplicate vault item without causing conflicts or errors. as a example one vault "*(object name)*" is "*Kicker1*", the second duplicated vault "*(object name)*" would be called "*Kicker2*".
- **Remark keyword** - This just replaces the remarks of the vault item. As a example fpxVaultKicker1

Did You Know?

Not only are table names and object used as keywords within a vault item, but we also use unique keywords in every vault as part of our subroutines and variable names. This makes organizing our script structure very easy, and also, because those names have the same keyword no matter what vault file script we are looking at, it means we instantly know what that variable does and makes tracking, modifing and debugging code a lot easier.
For example, for the variable that sets a starting case value, every main Vault file will always have "CaseStart" in that variable name,. Subroutines are the same, they always have a keyword as well. As a example, for the subroutines that control when a game is started, the subroutine name will always have "GameReset" in that subroutine name, and at the end of that subroutine name.

⚙ Changeable Variables

- Handles all the scoring using case statements.
- Uses a set keyword

v_(*table name***)(***object name***)Case**
This is the main variable used to handle multiple scoring when a bank is made. In the script, this is handled by the Subroutine : AddVault`objecttable`() .
This increases by 1 every time a bank is being made, or a trigger is rolled over.

v_(*table name***)(***object name***)CaseStart**
This is adjustable by both the developer and the person playing the game. This sets *v_(table name)(object name)Case* to a starting value when a game is started.   This is a common "pinsetting" with the arcade games, and within the majority of the fpx Vault items, determines whether to require the player to make a extra case before a feature will be lit, or to "jump" ahead to a higher case setting at the start of a new game.
In the case settings, if you wish to have the player go through a extra case within the case settings, there is always a case 0, which usually just gives points, so *v_(table name)(object name)CaseStart* would be set to 0. This would be a "hard" or "conservative" setting with the arcade tables.*v_(table name)(object name) Case* will start at this point at a new game.
If the player memory feature is switched off, (*v_(table name)(object name) Memory=0*), every new ball in play will reset back to the value set in *v_(table name)(object name)CaseStart* for every player

v_(*table name***)(***object name***)CaseEnd**
Because the case routines are set up to handle 10 case settings internally, you will need to define the case setting you want as the Ending case, and not continue on to a higher case. For example, most drop targets have 4 main case settings, usually scoring, a special feature, a extra ball and a special, plus a 5th case setting to repeat afterwards. This variable allows you to define more case settings (they have to be in numeric order) and then set the "end" point for the case settings, so the case settings will never go beyond this maximum range. This way, you have the option to expand or decrease the cases used for your modified vault file as you see fit.

v_(*table name***)(***object name***)CaseRepeat**
This is used to tell the script what to do once you have reached the maximum case setting in v*_(table name)(object name)*CaseEnd. All you need to do is input a value that is then used by *v_(table name)(object name)Case*
If you wish to start over the case settings back to the beginning, you can just input the same case number you used for v*_(table name)(object name)*CaseStart .
If you want the case to repeat the case set to v*_(table name)(object name)* CaseEnd, then just set this number to be the same as v*_(table name)(object name)*CaseEnd.
You can also set this number to any case value you want. If you set this number to a case setting beyond v*_(table name)(object name)*CaseEnd, the system will just repeat that case number till the player loses his ball.

If you set this value lower than v_*(table name)(object name)*CaseEnd, the script will start at that value and increase the cases till it gets back to v_*(table name)(object name)*CaseEnd. This allows you to repeat the scoring routines over and over till the player loses his ball in play.

Expanding or reducing the amount of Lens Lights needed

- fpxEngine uses a unique scripting structure, that allows both the light lens and bulbs to be included in a minutes amount of work, without causing errors in the script. We take advantage of the SET command, and using control variables instead of hard coded numbers, to expand (or contract) the amount of Light lens and playfield bulbs, that is used automatically in the script.

NOTE: For these code examples, ***fpx*** *is our "table name" and* ***Kicker1*** *is our "object name".*

v_(*table name***)(***object name***)Lights(9,9)**

- This is the main variable used to set lens lights within the script. We use TWO variables here, the first number is the "group" number, and the second value is the Object ID within that group. You can have a maximum of 9 "groups" with a maximum of 9 objects per group. This way, the script can handle as much as 81 objects, but is mainly used to split a collection of light lens to 2 groups so we can run different routines at the same time.
- The big advantage is using this method for For/Next loops, especially with blinking effects, but this method also makes it easier for other coding tricks and aspects.
- You will notice that everything is exactly the same way with the naming, this just makes things easier for coding, as you can understand the code a lot better when looking at one Vault item or another
- You need to define each light within a SET command using this variable, and assign a specific value for each light object.

code:
```
Set v_fpxKicker1Lights(1,1)=LightKicker1fpx1:Set v_fpxKicker1Lights(1,2)=LightKicker1fpx2
Set v_fpxKicker1Lights(1,3)=LightKicker1fpx3:Set v_fpxKicker1Lights(1,4)=LightKicker1fpx4
```

The values assigned MUST be in numeric order, or it will cause errors or certain lights "out of sequence" to not work. You will notice we use group 1, but we increase the object ID for that group by one and then assign a light lens already created in the editor to each ID
To expand this code, as a example adding a 5th lens light, you would just type this, then add the object in the editor and make sure it is called LightKicker1fpx5 as it's file name

code:
```
Set v_fpxKicker1Lights(1,5)=LightKicker1fpx5
```

v_(*table name***)(***object name***)LightCount**

Once you have done adding lights, you need to also tell the script how many lights you have, so the script will automatically handle the routines, especially for FOR/NEXT routines.

code:
```
v_fpxKicker1LightCount = 4  ' Number of Lens Lights used (in set code) in FOR NEXT loops
```

Since you just added a new light, this means you now have 5 lens lights, not 4 like before, so you need to update the total

code:
```
v_fpxKicker1LightCount = 5  ' Number of Lens Lights used (in set
code) in FOR NEXT loops
```

And here is a sample code within the vault script that uses v_fpxKicker1LightCount. This type of code is used every time in all Vault items, so you no longer need to search and replace any numbers, just the one time in v_fpxKicker1LightCount .
The coding was written in such a way, that you can have multiple lights, and after you get that set up, just change the one number in v_fpxKicker1LightCount to match the amount, and that's it. The engine will handle everything else. This code example switches off all the lights in this group at once.

code:
```
FOR x = 1 To
(v_fpxKicker1LightCount):v_fpxKicker1Lights
(1,x).State=BulbOff:NEXT  ' All lens lights off first
```

NOTE: we have just 4 lights already defined, so the new light lens object name has a 5 at the end in it.

Adding the lights to BAM Lighting code
We do not get into details about BAM, as there is enough to do for me as it is, but BAM does have the ability to make a major improvement to lighting. The simplest way is to duplicate a existing line and changing the name of the duplicated object to the newly created object name you set in the editor. Here's the original lines of the lighting BAM code (4 light objects):

code:
```
' light lens
 LightKicker1fpx1EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx1EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx2EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx2EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx3EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx3EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx4EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
```

and here's the code with the added new light (now with 5 light objects):

code:
```
' light lens
 LightKicker1fpx1EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx1EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx2EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx2EXT
```

```
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx3EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx3EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx4EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx5EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx5EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx5EXT
.GlowBrightness=(fpxLensGlowBrightness)
```

You will notice that "EXT" has to be included after the file name. This is a BAM thing. If you do not include EXT after the object file name, or have a object name that is present or actually in the editor, you will get a error message and your table will not work. With fpx, you will notice that everything is named the same way, and done the same way, including the lights, which is a increasing numeric number at the end.

Expanding or reducing the amount of Bulbs/Plastics needed

v_(*table name*)(*object name*)Bulbs(9,9)

- This is the main variable used to set play field bulbs AND plastics within the script. We use TWO variables here, the first number is the "group" number, and the second value is the Object ID within that group. You can have a maximum of 9 "groups" with a maximum of 9 objects per group. This way, the script can handle as much as 81 objects, but is mainly used to split a collection of bulbs/plastics to 2 groups so we can run different routines at the same time.
- This is done exactly the same way as the lights above. fpx always tries as much as possible to have one way, and only one way to do things. This makes it far easier to use, as you only need to learn that one way to understand the script.
- The big advantage is using this method for For/Next loops, especially with blinking bulb/plastic effects, but this method also makes it easier for other coding tricks and aspects. We combine the bulbs and plastics into one here as they work in unison anyway.
- You will notice that everything is exactly the same way with the naming, this just makes things easier for coding, as you can understand the code a lot better when looking at one Vault item or another
- You need to define each bulb/plastic within a SET command using this variable, and assign a specific value for each light.

`code:`
```
Set v_fpxKicker1Bulbs(1,1)=BulbKicker1fpx1:Set
v_fpxKicker1Bulbs(1,2)=BulbKicker1fpx2:Set
v_fpxKicker1Bulbs(1,3)=PlasticKicker1fpx1 ' Defines Bulb and
Plastics
```

The values assigned MUST be in numeric order, or it will cause errors or certain lights "out of sequence" to not work. You will notice we use group 1, but we increase the object ID for that group by one and then assign a bulb already created in the editor to each ID

To expand this code, as a example adding a 4th playfield bulb, you would just type this

code:
```
v_fpxKicker1Bulbs(1,4)=BulbKicker1fpx3
```

NOTE: we have just 2 bulbs already defined, plus a plastic, so the new bulb object name has a 3 at the end in it.

v_(*table name***)(***object name***)BulbCount**

Once you have done adding bulbs and plastics, you need to also tell the script how many you have, so the script will automatically handle the routines, especially for FOR/NEXT routines.

code:
```
v_fpxKicker1BulbCount = 3  ' Number of bulbs used (in set code) in
FOR NEXT loops
```

Since you just added a new light, this means you now have 4 bulbs and plastics, not 3 like before, so you need to update the total

code:
```
v_fpxKicker1BulbCount = 4  ' Number of bulbs used (in set code) in
FOR NEXT loops
```

and here is a sample code within the vault script. This type of code is used every time in all Vault items, so you no longer need to search and replace anything. This code switches off all the lights in this group at once.

code:
```
FOR x = 1 To
(v_fpxKicker1LightCount):v_fpxKicker1Lights
(1,x).State=BulbOff:NEXT  ' All lens lights off first
```

Adding the Bulbs and Plastic lights to BAM Lighting code

We do not get into details about BAM, as there is enough to do for me as it is, but BAM does have the ability to make a major improvement to lighting. The simplest way is to duplicate a existing line and changing the name of the duplicated object to the newly created object name you set in the editor. Here's the original lines of the Bulb BAM code (2 bulb objects and 1 plastic object):

code:
```
' BAM bulb
 BulbKicker1fpx1EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbKicker1fpx2EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx2EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx2EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticKicker1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticKicker1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticKicker1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
```

and here's the code with the added new light (now with 3 bulb objects and 1 plastic object):

code:
```
' BAM bulb
 BulbKicker1fpx1EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbKicker1fpx2EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx2EXT
```

```
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx2EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticKicker1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticKicker1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticKicker1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
BulbKicker1fpx3EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx3EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx3EXT
.GlowBrightness=(fpxBulbGlowBrightness)
```

You will notice that "EXT" has to be included after the file name. This is a BAM thing. If you do not include EXT after the object file name, or have a object name that is present or actually in the editor, you will get a error message and your table will not work. With fpx, you will notice that everything is named the same way, and done the same way, including the Bulbs, which is a increasing numeric number at the end.

v_(*table name*)(*object name*)CaseEnd
Because the case routines are set up to handle 10 case settings internally, you will need to define the case setting you want as the Ending case, and not continue on to a higher case. For example, most drop targets have 4 main case settings, usually scoring, a special feature, a extra ball and a special, plus a 5th case setting to repeat afterwards. This variable allows you to set the "end" point for the case settings, so the case settings will never go beyond this maximum range. This way, you have the option to expand or decrease the cases used for your modified vault file as you see fit.

v_(*table name*)(*object name*)CaseRepeat
This is used to tell the script what to do once you have reached the maximum case setting in v_fpxKicker1CaseEnd. All you need to do is input a value that is then used by **v_fpxKicker1Case**
If you wish to start over the case settings back to the beginning, you can just input the same case number you used for **v_fpxKicker1CaseStart** .
If you want the case to repeat the case set to v_fpxKicker1CaseEnd, then just set this number to be the same as v_fpxKicker1CaseEnd.
You can also set this number to any case value you want. If you set this number to a case setting beyond v_fpxKicker1CaseEnd, the system will just repeat that case number till the player loses his ball.
If you set this value lower than v_fpxKicker1CaseEnd, the script will start at that value and increase the cases till it gets back to v_fpxKicker1CaseEnd. This allows you to repeat the scoring routines over and over till the player loses his ball in play.

The Next Step: Adding more case scoring

Once you have changed the variable values above, you can now add additional cases to your scoring. As a example, we will use the Kicker1 example. This awards points for the first 2 cases (case 0 (no light lens) and case 1 (first light lens)), 25000 points for case 2(2nd light lens), a Extra Ball for case 3 (3rd light lens) and finally a special for case 4 (4th light lens). For this example, v_fpxKicker1CaseStart = 1 (starting case for a new game or new ball in play) v_fpxKicker1CaseEnd=4 (the maximum amount of case) and v_fpxKicker1CaseRepeat=4 (it will repeat case 4 and award a special every time)

If we wanted to change this, and instead to not have a special awarded more than once, but to have it repeat 25000 points instead afterwards, we would need to add another defined case setting, and create any new objects needed.
We need to first add the items and code changes and additions in the very top or where the DIM and SET code is. You can find this code by searching for "Individual Vault Routines" in the script for the fpxEngine.fpt file
That code would look like this:

code:
```
Const VaultKicker1fpxDebug=0                    ' Dev Debug. Set to
1 to generate debug text for this vault item
Dim
v_fpxKicker1On
,v_fpxKicker1Memory
,v_fpxKicker1Lights
(
9
,9),v_fpxKicker1Bulbs(9,9),v_fpxKicker1LightCount,v_fpxKicker1BulbCount
         ' Variables
Dim
v_fpxKicker1Case
,v_fpxKicker1CaseEnd,v_fpxKicker1CaseStart,v_fpxKicker1CaseRepeat
                    ' Scoring variables
Dim m_fpxKicker1p1,m_fpxKicker1p2,m_fpxKicker1p3,m_fpxKicker1p4
                    ' Player(s) memory
If v_fpxKicker1On=1 THEN                       ' Checks if vault item is
being used
 Set v_fpxKicker1Lights(1,1)=LightKicker1fpx1:Set v_fpxKicker1Lights(1,2)
=LightKicker1fpx2                 ' Defines Light Lens
 Set v_fpxKicker1Lights(1,3)=LightKicker1fpx3:Set v_fpxKicker1Lights(1,4)
=LightKicker1fpx4
 Set v_fpxKicker1Bulbs(1,1)=BulbKicker1fpx1:Set v_fpxKicker1Bulbs(1,2)
=BulbKicker1fpx2:Set v_fpxKicker1Bulbs(1,3)=PlasticKicker1fpx1          '
Defines Bulb and Plastics
' BAM bulb
 BulbKicker1fpx1EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbKicker1fpx2EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx2EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx2EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticKicker1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticKicker1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticKicker1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
' light lens
 LightKicker1fpx1EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx1EXT
```

```
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx2EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx2EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx3EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx3EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx4EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
 v_fpxKicker1LightCount = 4                          ' Number of Lens Lights
used (in set code) in FOR NEXT loops
 v_fpxKicker1BulbCount = 3                           ' Number of bulbs used
(in set code) in FOR NEXT loops
 v_fpxKicker1CaseEnd=4                              ' Last case number for
scoring, used to reset case to beginning if over at next ball
 v_fpxKicker1CaseRepeat=4                              ' If
v_fpxKicker1Case>v_fpxKicker1CaseStart. This forces either a repeat of
last case, or resets the case back to the beginning
End If
```

Since we only want to add one more case setting, we first change v_fpxKicker1CaseRepeat from 4 to 5

code:
```
v_fpxKicker1CaseRepeat=5
```

Then we need to create a new light lens object and maybe a extra playfield bulb. We already did this as a example above, so lets show you that code again.

code:
```
Set v_fpxKicker1Lights(1,5)=LightKicker1fpx5
Set v_fpxKicker1Bulbs(1,4)=BulbKicker1fpx3
```

and make sure we tell the script the proper amount of bulbs/plastics and light lens we are using, also explained above as a example.

code:
```
v_fpxKicker1LightCount = 5  ' Number of Lens Lights used (in set code) in
FOR NEXT loops
v_fpxKicker1BulbCount = 4  ' Number of bulbs used (in set code) in FOR
NEXT loops
```

And finally, add the BAM lighting code:

code:
```
LightKicker1fpx5EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx5EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx5EXT
.GlowBrightness=(fpxLensGlowBrightness)
BulbKicker1fpx3EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx3EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx3EXT
.GlowBrightness=(fpxBulbGlowBrightness)
```

So now, that code with the new changes will look like this:

code:
```
Const VaultKicker1fpxDebug=0                          ' Dev Debug. Set to
1 to generate debug text for this vault item
```

```
Dim
v_fpxKicker1On
,v_fpxKicker1Memory
,v_fpxKicker1Lights
(
9
,9),v_fpxKicker1Bulbs(9,9),v_fpxKicker1LightCount,v_fpxKicker1BulbCount
          ' Variables
Dim
v_fpxKicker1Case
,v_fpxKicker1CaseEnd,v_fpxKicker1CaseStart,v_fpxKicker1CaseRepeat
                      ' Scoring variables
Dim m_fpxKicker1p1,m_fpxKicker1p2,m_fpxKicker1p3,m_fpxKicker1p4
                       ' Player(s) memory
If v_fpxKicker1On=1 THEN                              ' Checks if vault item is
being used
 Set v_fpxKicker1Lights(1,1)=LightKicker1fpx1:Set v_fpxKicker1Lights(1,2)
=LightKicker1fpx2                   ' Defines Light Lens
 Set v_fpxKicker1Lights(1,3)=LightKicker1fpx3:Set v_fpxKicker1Lights(1,4)
=LightKicker1fpx4
Set v_fpxKicker1Lights(1,5)=LightKicker1fpx5
                                                                        ' ***
THIS IS NEW ADDITION ***
 Set v_fpxKicker1Bulbs(1,1)=BulbKicker1fpx1:Set v_fpxKicker1Bulbs(1,2)
=BulbKicker1fpx2:Set v_fpxKicker1Bulbs(1,3)=PlasticKicker1fpx1          '
Defines Bulb and Plastics
Set v_fpxKicker1Bulbs(1,4)=BulbKicker1fpx3
                                                                       ' ***
THIS IS NEW ADDITION ***
' BAM bulb
 BulbKicker1fpx1EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbKicker1fpx2EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx2EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx2EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticKicker1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticKicker1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticKicker1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
BulbKicker1fpx3EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx3EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx3EXT
.GlowBrightness=(fpxBulbGlowBrightness)   ' *** THIS IS NEW ADDITION
***
' light lens
 LightKicker1fpx1EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx1EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
```

```
 LightKicker1fpx2EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx2EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx3EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx3EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx4EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
LightKicker1fpx5EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx5EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx5EXT
.GlowBrightness=(fpxLensGlowBrightness)   ' *** THIS IS NEW ADDITION
***
 v_fpxKicker1LightCount = 5                          ' *** THIS HAS BEEN
CHANGED *** Number of Lens Lights used (in set code) in FOR NEXT
loops
 v_fpxKicker1BulbCount = 4                           ' *** THIS HAS BEEN
CHANGED ***Number of bulbs used (in set code) in FOR NEXT loops
 v_fpxKicker1CaseEnd=4                              ' Last case number for
scoring, used to reset case to beginning if over at next ball
 v_fpxKicker1CaseRepeat=5                            ' *** THIS HAS BEEN
CHANGED ***If v_fpxKicker1Case>v_fpxKicker1CaseStart. This forces
either a repeat of last case, or resets the case back to the beginning
End If
```

Now, we need to add the new scoring code, this is done in the same subroutine that most users would use to change the scoring, (Sub AddVault**(***object name***)(***table name***)**) but first this is what the code would look like:

code:
```
Sub AddVaultScoreKicker1fpx1()
 'IF VaultDebug=1 THEN AddDebugText "AddVaultScoreKicker1fpx1() "
          '  Dev Debug Code
 If v_fpxKicker1On=1 THEN
   Select Case v_fpxKicker1Case
     Case 0
         ' This is used as the starting TargetBank score if
v_fpxKicker1CaseStart = 0.
    AddScore(5000)
         '  Adds the value within the brackets to your player(s) score
    AddJackpot(1000)
          '  Adds the value within the brackets to the Jackpot value which can
be collected later on in the game
    AddBonus(1)
         '  Adds one bonus within the brackets to the End-Of-Ball Bonus
Countdown routine
   Case 1
        ' This is used as the starting TargetBank score if
v_fpxKicker1CaseStart = 1.
    AddScore(10000)
```

```
     AddJackpot(10000)
     AddBonus(1)
    Case 2
     AddScoringEvent "25kAward"
    Case 3
     AddScoringEvent "ExtraBall"
       ' Adds a Extra Ball
    Case 4
     AddScoringEvent "Special"
       ' Adds a credit (free game)
    Case 5
     ' See the manual to add more cases for scoring
    Case 6
     ' See the manual to add more cases for scoring
    Case 7
     ' See the manual to add more cases for scoring
    Case 8
     ' See the manual to add more cases for scoring
    Case 9
     ' See the manual to add more cases for scoring
    Case Else
      ' Case else will score if case is higher than 5, and will keep repeating
     AddScoringEvent "25kAward"
   End Select
  End If
End Sub
```

Now, we just need to add the new scoring code to case 5 in this scoring subroutine.

`code:`

```
Sub AddVaultScoreKicker1fpx1()
 'IF VaultDebug=1 THEN AddDebugText "AddVaultScoreKicker1fpx1() "
        '  Dev Debug Code
 If v_fpxKicker1On=1 THEN
   Select Case v_fpxKicker1Case
     Case 0
        ' This is used as the starting TargetBank score if
v_fpxKicker1CaseStart = 0.
     AddScore(5000)
        '  Adds the value within the brackets to your player(s) score
     AddJackpot(1000)
         '  Adds the value within the brackets to the Jackpot value which can
be collected later on in the game
     AddBonus(1)
        '  Adds one bonus within the brackets to the End-Of-Ball Bonus
Countdown routine
    Case 1
       ' This is used as the starting TargetBank score if
v_fpxKicker1CaseStart = 1.
     AddScore(10000)
     AddJackpot(10000)
     AddBonus(1)
    Case 2
     AddScoringEvent "25kAward"
```

```
            Case 3
             AddScoringEvent "ExtraBall"
                ' Adds a Extra Ball
            Case 4
             AddScoringEvent "Special"
                ' Adds a credit (free game)
            Case 5
            AddScoringEvent "25kAward"
            Case 6
             ' See the manual to add more cases for scoring
            Case 7
             ' See the manual to add more cases for scoring
            Case 8
             ' See the manual to add more cases for scoring
            Case 9
             ' See the manual to add more cases for scoring
            Case Else
              ' Case else will score if case is higher than 5, and will keep repeating
             AddScoringEvent "25kAward"
           End Select
          End If
         End Sub
```

fpxEngine will automatically handle the extra lights and objects we just added. Note that v_fpxKicker1CaseEnd=4 is still the same, the script is set to start repeating after the final case ending value is done. You can have up to 10 cases scoring, with any case values over Case 9 repeating by having v_fpxKicker1CaseRepeat
set to "10" (v_fpxKicker1CaseRepeat = 10)
You can also use v_fpxKicker1CaseRepeat to "rollover" and go back to a earlier case setting and start over the entire process. For example, after your special, you want to go back to Case 1. You set v_fpxKicker1CaseRepeat to 1 (v_fpxKicker1CaseRepeat = 1) and after the special, the case value will reset back to 1, and start over and repeat all the following case scoring.

Vault - Drop Targets

vault_fpxDropTargetBank1

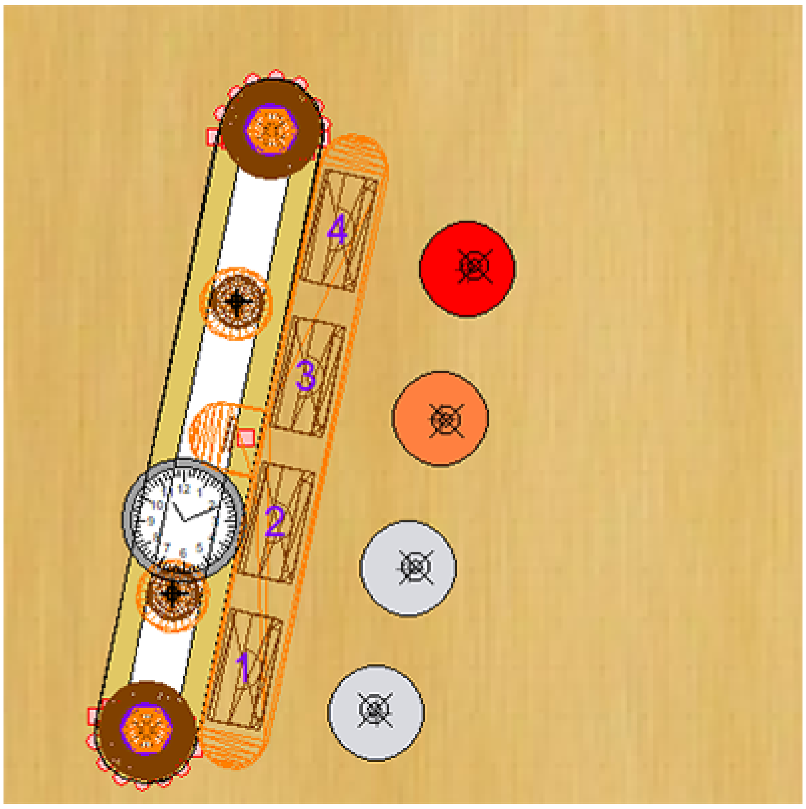
Vault - Drop Targets - vault_fpxDropTargetBank1

Vault - Drop Targets - vault_fpxDropTargetBank1

How to use this Vault Item in the fpxEngine

Using this Vault item and including it in your table design is very simple. It's just a couple copy and pastes from one fpt file to another!
Don't know How? Just click here.

vault_fpxDropTargetBank1



How it works

This Vault item has 5 stages of scoring, plus the ability to set player memory so a player can carry on through the stages with his next ball. The script is set to automatically reset the stages if the player reaches the 5th stage (25000 pts) in his previous ball so that player can have the chance to score a Extra Ball and a free game a second time.

User adjustments (pinsettings)

v_fpxDTB1CaseStart = 1
Initial Stage (Light) to be lit in this vault item for start of each ball,
- Set to 0 for no lights at start or "hard". (You need to complete the Target bank routine once before the first light will be lit)
- Set to 1 to have the first light on at the start of a game

v_fpxDTB1Memory=1
Handles the Memory feature for

DropTarget Score by each player
- Set to 1 if you want the player(s) DropTarget Bank made total per game in play carried over to his next ball in play.
- Set to 0 to have the DropTarget Bank Made total per game in play reset back to beginning with each new ball.

⚙ How to Change Scoring (Bank)

Sub AddVaultScoreDTB1fpx1()
This subroutine handles scoring when a bank is made. you can modify each stage by modifying the code with each case setting by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement fpxEngine AddScoringEvents routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.
- Case 0: 5000 points,1000 added to Jackpot value , 1 added to bonus count at loss of a ball
- Case 1: 10000 points,10000 added

to Jackpot value , 1 added to bonus count at loss of a ball
- Case 2: 25000 points (AddScoringEvent "25kAward")
- Case 3: Extra Ball (AddScoringEvent "ExtraBall")
- Case 4: Special (AddScoringEvent "Special")
- Case 5: 25000 points (AddScoringEvent "25kAward") This will repeat till loss of ball, then the case settings will reset back to the beginning case at the players next ball.

How to Change scoring (Single)

Sub DTB1fpx1_Hit()
This subroutine handles scoring when a single target is made. you can modify this just by changing the scoring code by adding different values between the brackets for higher scores
AddScore(1000)   ' Adds the value within the brackets to your player(s) score for Single Target Hit (not entire bank)
AddJackpot(1000)   ' Adds the value within the brackets to the Jackpot value which can be collected later on in the game
AddBonus(1)   ' Adds one bonus within the brackets to the End-Of-Ball Bonus Countdown routine

There is a rubber object behind the targets, which also generates points
Sub DTB1fpxRubber1_Hit()
AddScore(10)

List of Objects

These objects are needed for the entire script to work. If you accidentally delete one of these objects and then run the table, the table will throw up a error message telling you a object is missing. If you restore that object back, your table will run fine.

This is a list of all the objects needed:

Lights

LightDTB1fpx1
LightDTB1fpx2
LightDTB1fpx3
LightDTB1fpx4

Bulbs

BulbDTB1fpx1
BulbDTB1fpx2
PlasticDTB1fpx1

Objects

DTB1fpx1
DTB1fpxRubber1
TimerDTB1fpx1
t_DTB1fpx1

Additional Information

- All Drop Targets objects are set to Layer 1 in your editor. Each major group of Vault items are set to their own layer for easy moving and modification.
- Layer 9 are reserved for the top (header) and Bottom (slingshots/flippers/aprons)
- Layer 0 is reserved for the fpxEngine objects.
- All objects can be modified or rotated to fit your design. Consult the FP manual for making changes directly in the editor, or visit the Pinball Nirvana website if you get really stuck.

A Note

fpxEngine uses a shaped light as a "plastic", as well as a semi-transparent surface as a edge around it. This is set to a height of 32mm, to form a proper cover over posts/rubbers etc that the ball can roll underneath. The Plastic light is part of the bulb code, it will flash in unison with the bulbs underneath, and can not be deleted without causing a error message when you run the table. Plastics always have the word in front of them. Semi-transparent edges (surfaces) always has the "t_ in front of the name.

Vault Worksheet

The very nature of the fpxEngine is to allow absolute beginners to build and create complex tables with as little actual experience as possible, but you still need to learn a tiny bit of coding sometimes. Most of the Vault items will be based on the actual arcade table it came from, so it's just a quick copy and paste the design elements, the code, and then adding lights in the LightList manager.

More advanced coders though may wish to duplicate or modify this code to run different routines. This Vault item worksheet is the master template I use to create vault items from, so if you decide to duplicate or modify this vault item, you can change it easily using this worksheet. For more information, check the Using the Vault Worksheets page.

code:

```
' * Find and replace code keywords for duplication or new vault items
' replacement table vault name keyword: fpx
' replacement object keyword: DTB1
' Remark Keyword: DropTargetBank1
' Codeing Names Keywords
' v_ variables keyword
' m_ memory keyword
' * List of variables
' VaultDTB1fpxDebug
' v_fpxDTB1On
' v_fpxDTB1CaseStart
'  v_fpxDTB1CaseEnd
' v_fpxDTB1CaseRepeat
'  v_fpxDTB1Memory
' v_fpxDTB1Case
' v_fpxDTB1Lights(9,9)      ' For/Next Loops
' v_fpxDTB1Bulbs(9,9)       ' For/Next Loops
' v_fpxDTB1LightCount       ' Amount Lights in v_fpxDTB1Lights
' v_fpxDTB1BulbCount        ' Amount Bulbs in v_fpxDTB1Bulbs
' m_fpxDTB1p1
' m_fpxDTB1p2
' m_fpxDTB1p3
' m_fpxDTB1p4
' * Subroutines
' AddVaultDTB1fpx()       ' Main vault routine
' CloseVaultDTB1fpx()      ' Light control
' DTB1fpxGameReset()       ' New game
' DTB1fpxBallReset()      ' New ball in play
' DTB1fpxMemorySave()      ' saves players progress
' DTB1fpxMemoryLoad()      ' Restores players progress
' AddVaultScoreDTB1fpx
' fpxDTB1

' * Unique subroutines to this vault
' TimerTarget1fpx_Expired() ' Timer
' * Lights
' LightDTB1fpx1
' LightDTB1fpx2
' LightDTB1fpx3
' LightDTB1fpx4
' * Bulbs
' BulbDTB1fpx1
' BulbDTB1fpx2
' PlasticfpxDropTarget3Bank1
' * Objects
' DTB1fpx1
' DTB1fpxRubber1
' TimerDTB1fpx1
' t_fpxDropTarget3Bank1
' ----------------------------------------------------
' * fpx Vault DropTargetBank1 *
' ----------------------------------------------------
' For the fpxEngine.
' 1. Make sure all layers in the FP editor are "visible".
' Copy all the table elements in the editor from this fpt and then paste these
elements into your fpxEngine table.
' 2. Set User adjustments below to suit, then copy this entire script from this fpt
and paste into the script  for your fpx table.
```

```
' I recommend  pasting  in the HIT SECTION.
' 3. Add Lights,plastics and bulbs to your LightList Manager in your fpx table.
' - Look for the names listed on the left side in the LightList Manager to have
"light", "bulb" or "plastic" as fpx will use these words
' at the beginning of the object name for all Vault Items
'  - If you do not know how to transfere the lights, both the fpx manual (in the
vault pages) and the Future Pinball manual explains how to do this.
' 4. Press "play"!
' * User adjustments  *
' Initial light to be lit in this vault item for start of each ball, set to 0 for no lights at
start or "hard".
' (You need to complete the routine once before the first light will be lit)
' Set to 1 to have the first light on at the start of a game
 v_fpxDTB1CaseStart = 1                          ' Variable to hold DropTarget
Bank starting value
' Handles the Memory feature for DropTarget Score (common pin setting)
' Set to one if you want the player(s) DropTarget Bank made total per game in
play carried over to his next ball in play.
' Set to 0 to have the DropTarget Bank Made total per game in play reset back
to beginning with each new ball,
' as set by v_fpxDTB1CaseStart  above.
 v_fpxDTB1Memory=1                               ' Variable to hold DropTarget Bank
Score Value from ball to Ball (for each player)
' *** Scoring ***
' Allows you to set the scoring. Everything else is done for you within the engine.
You can add/change anything you want in here
' like adding a multiplier or turning on Inlane lights etc. See Beginners Guide in
manual
Sub AddVaultScoreDTB1fpx1()
 'IF VaultDebug=1 THEN AddDebugText "AddVaultScoreDTB1fpx1() "
          '  Dev Debug Code
 If v_fpxDTB1On=1 THEN
  Select Case v_fpxDTB1Case
   Case 0                         ' This is used as the starting TargetBank score
if v_fpxDTB1CaseStart = 0.
  AddScore(5000)                            '  Adds the value within the brackets to
your player(s) score
  AddJackpot(1000)                          '  Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
  AddBonus(1)                               '  Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
   Case 1                         ' This is used as the starting TargetBank score
if v_fpxDTB1CaseStart = 1.
  AddScore(10000)
  AddJackpot(10000)
  AddBonus(1)
   Case 2
  AddScoringEvent "25kAward"
   Case 3
  AddScoringEvent "ExtraBall"                          ' Adds a Extra Ball
```

```
   Case 4
  AddScoringEvent "Special"                          ' Adds a credit (free game)
   Case Else                       ' Case else will score if case is higher than 5,
and will keep repeating
  AddScoringEvent "25kAward"
  End Select
 End If
End Sub
' Drop Target Hit . You need to set a scoring value here for a single drop target
hit that doesn't complete a bank
Sub DTB1fpx1_Hit()
 'IF VaultDebug=1 THEN AddDebugText "DTB1_Hit() "                          ' Play
the mechanical sound. this will sound even if game is tilted
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub : END IF
     ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
 AddScore(1000)                             ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
 AddJackpot(1000)                             ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
 AddBonus(1)                           ' Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
 If DTB1fpx1.Dropped = False Then AddMusicSet "drop" : Exit Sub : END IF
             ' Look to see if all targets in the bank is down, if not, play drop
sound,exit this subroutine
 If v_fpxDTB1On=1 THEN AddVaultDTB1fpx() : END IF                        ' if all
targets in this bank are down, go to bank made scoring subroutine
 set LastSwitchHit = DTB1fpx1                             ' FP code we can use if
needed. Just good programing
End Sub
' Rubber hit behind targets
Sub DTB1fpxRubber1_Hit()
 AddScore(10)                             ' Adds the value within the brackets to
your player(s) score
 AddMusicSet "sling"                             ' The music routine
End Sub
' Engine code, do not change or modify. This is needed by the engine to
reconize this vault item. Changing this will cause errors.
 v_fpxDTB1On=1                             '  KEEP THIS SET TO 1.
'  /END fpx Vault DropTargetBank1
' Placed in main fpx Vault Routines ******
' ----------------------------------------------------------
' *** fpx Vault DropTargetBank1 ***
' ----------------------------------------------------------
Const VaultDTB1fpxDebug=1                             ' Dev Debug. Set to 1 to
generate debug text for this vault item
Dim
v_fpxDTB1On
,v_fpxDTB1Memory
,v_fpxDTB1Lights
(9,9),v_fpxDTB1Bulbs(9,9),v_fpxDTB1LightCount,v_fpxDTB1BulbCount           '
```

```
Variables
Dim
v_fpxDTB1Case
,v_fpxDTB1CaseEnd,v_fpxDTB1CaseStart,v_fpxDTB1CaseRepeat
      ' Scoring variables
Dim m_fpxDTB1p1,m_fpxDTB1p2,m_fpxDTB1p3,m_fpxDTB1p4
    ' Player(s) memory
If v_fpxDTB1On=1 THEN                      ' Checks if vault item is being
used
 Set v_fpxDTB1Lights(1,1)=LightDTB1fpx1:Set v_fpxDTB1Lights(1,2)
=LightDTB1fpx2:Set v_fpxDTB1Lights(1,3)=LightDTB1fpx3:Set
v_fpxDTB1Lights(1,4)=LightDTB1fpx4
 Set v_fpxDTB1Bulbs(1,1)=BulbDTB1fpx1:Set v_fpxDTB1Bulbs(1,2)
=BulbDTB1fpx2:Set v_fpxDTB1Bulbs(1,3)=PlasticDTB1fpx1        ' Bulb and
Plastics
' BAM bulb
 BulbDTB1fpx1EXT
.Brightness=(fpxBulbBrightness):BulbDTB1fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbDTB1fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbDTB1fpx2EXT
.Brightness=(fpxBulbBrightness):BulbDTB1fpx2EXT
.GlowRadius=(fpxBulbGlowRadius):BulbDTB1fpx2EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticDTB1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticDTB1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticDTB1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
' light lens
 LightDTB1fpx1EXT
.Brightness=(fpxLensBrightness):LightDTB1fpx1EXT
.GlowRadius=(fpxLensGlowRadius):LightDTB1fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightDTB1fpx2EXT
.Brightness=(fpxLensBrightness):LightDTB1fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightDTB1fpx2EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightDTB1fpx3EXT
.Brightness=(fpxLensBrightness):LightDTB1fpx3EXT
.GlowRadius=(fpxLensGlowRadius):LightDTB1fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightDTB1fpx4EXT
.Brightness=(fpxLensBrightness):LightDTB1fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightDTB1fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
 PlasticDTB1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticDTB1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticDTB1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
 v_fpxDTB1LightCount = 4                   ' Number of Lens Lights used
(in set code) in FOR NEXT loops
 v_fpxDTB1BulbCount = 3                    ' Number of bulbs used (in set
code) in FOR NEXT loops
 v_fpxDTB1CaseEnd=5
 v_fpxDTB1CaseRepeat=5                          ' Last case number for
scoring, used to reset case to beginning if over at next ball
End If
' Main scoring routine. On target bank, v_fpxDTB1Case scores then increases
by one
Sub AddVaultDTB1fpx()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
      ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
 If v_fpxDTB1On=1 THEN                                    ' Check if vault item is
being used
  IF VaultDTB1fpxDebug=1 THEN  AddDebugText "AddVaultDTB1fpx()"
        ' Dev Debug Code
  FOR x = 1 To
(v_fpxDTB1LightCount):v_fpxDTB1Lights(1,x).State=BulbOff:NEXT
' All lens lights off first
  LockDisplay=0                                ' Clears any music priorty code
  DisplayBlinkInterval=(FlashForMSBlinkInterval+40)                 ' Sets new
interval for DT lights
  FOR x = 1 TO v_fpxDTB1BulbCount:v_fpxDTB1Bulbs(1,x).FlashForMs
(MusicIntervalTime), (FlashForMSBlinkInterval), BulbOn:NEXT        ' Bulbs
behind object will Blink rapidly for time set
  AddMusicSet "dropbank"                          ' Note. Default music is
overwritten by any AddScoringEvent code, so you need this default in case
there is no special scoring feature
  Select Case v_fpxDTB1Case
   Case 0 :                              ' v_fpx CaseStart = 0. no light lens till next
case
   Case 1 :                              ' v_fpx CaseStart = 1. first light lens used
   Case 2 : v_fpxDTB1Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn                ' Second Light lens used
   Case 3 : FlushDisplay()                          ' Have to add flushdisplay here
before extra ball/special/jackpot routines
   Case 4 : FlushDisplay() : v_fpxDTB1Lights(1,2).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval),BulbOn           ' Last light lens, so turn
back on LightLens 2 to keep repeating till loss of ball
   Case 5 : v_fpxDTB1Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn                ' After last case, this just keeps
repeating for the ball in play till ball lost
  End Select
  AddVaultScoreDTB1fpx1()                                  ' Goto scoring code
as set by developer. Must be before increase case code! (geez shiva)
  v_fpxDTB1Case = v_fpxDTB1Case+1                                  ' Increases
value by 1 for select case routines
  IF v_fpxDTB1Case> v_fpxDTB1CaseEnd THEN
v_fpxDTB1Case=(v_fpxDTB1CaseEnd)                   ' Wraps back to caseStart if
over max CaseEnd
  TimerDTB1fpx1.Set True, 1000                          ' Resets targets (after set
delay time)
```

```
  DTB1fpxMemorySave()                                   ' Save to that players
memory
  CloseVaultDTB1fpx()                           ' Run closing light routine

 END IF
End Sub
' Timer to reset target bank after a delay.
Sub TimerDTB1fpx1_Expired()
 TimerDTB1fpx1.Enabled = False
 If v_fpxDTB1On=1 THEN                                   ' Check if vault item is
being used
  IF VaultDTB1fpxDebug=1 THEN  AddDebugText "TimerDTB1fpx1_Expired() "
                  ' Dev Debug Code
  DTB1fpx1.SolenoidPulse : PlaySound "DTargetReset"                      '
Resets drop target bank
 END IF
End sub
' runs the correct light routine and restores proper light if v_fpxDTB1Memory=1
Sub CloseVaultDTB1fpx()
  If v_fpxDTB1On=1 THEN                                  ' Only if this feature is
set to "1" and nothing else
   IF VaultDTB1fpxDebug=1 THEN  AddDebugText "CloseVaultDTB1fpx()
":AddDebugText  " ":END IF               ' Dev Debug Code
   FOR x = 1 To (v_fpxDTB1LightCount): v_fpxDTB1Lights(1,x).State =
BulbOff:NEXT
   Select Case v_fpxDTB1Case                                   ' Sets next light to
display based on Case
    Case 0: Exit Sub
    Case 1: v_fpxDTB1Lights(1,1).State = BulbOn:
v_fpxDTB1Lights(1,1).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn      ' 1k, light 10k light for next case
    Case 2: v_fpxDTB1Lights(1,2).State = BulbOn:
v_fpxDTB1Lights(1,2).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn      ' 20k
    Case 3: v_fpxDTB1Lights(1,3).State = BulbOn:
v_fpxDTB1Lights(1,3).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn      ' Extra Ball
    Case 4: v_fpxDTB1Lights(1,4).State = BulbOn:
v_fpxDTB1Lights(1,4).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn      ' Special
    Case 5:v_fpxDTB1Lights(1,2).State = BulbOn:
v_fpxDTB1Lights(1,2).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn     ' Any thing over
   END SELECT
  END IF
End Sub
' Game Start. This resets the variables. This is used by all Vault items and
called directly in the main engine code
Sub DTB1fpxGameReset()
 IF v_fpxDTB1On=1 THEN                             ' Check if vault item is being
used
```

```
  IF VaultDTB1fpxDebug=1 THEN  AddDebugText
"DTB1fpxGameReset()":AddDebugText  " ":END IF              ' Dev Debug Code
  TimerDTB1fpx1.Set True, 20                                ' Resets targets (after set
delay time)
  FOR x = 1 To (v_fpxDTB1LightCount): v_fpxDTB1Lights(1,x).State =
BulbOff:NEXT                     ' We turn off all the lights first before we update the
scoring
  FOR x = 1 TO v_fpxDTB1BulbCount:v_fpxDTB1Bulbs(1,x).State =
BulbOn:NEXT                      ' We turn on all the Bulb lights first before we update
the scoring
  v_fpxDTB1Case=(v_fpxDTB1CaseStart)                                 ' Resets scoring
case setting back to starting default (set in vault user options)
  m_fpxDTB1p1=(v_fpxDTB1CaseStart):m_fpxDTB1p2=(v_fpxDTB1CaseStart):
m_fpxDTB1p3=(v_fpxDTB1CaseStart):m_fpxDTB1p4=(v_fpxDTB1CaseStart)    '
Resets variables back to initial starting point
 END IF
End Sub
' Run at NewBall. Also used as a blanket reset used for tilt, startup or game
over. This is used by all Vault items and called directly in the main engine code
Sub DTB1fpxBallReset()
 IF v_fpxDTB1On=1 THEN                                   ' Check if vault item is being
used
  IF VaultDTB1fpxDebug=1 THEN  AddDebugText "CloseVaultDTB1fpx()
":AddDebugText  " ":END IF              ' Dev Debug Code
  TimerDTB1fpx1.Set True, 20                                ' Resets targets (after set
delay time) Note we do this here before the memory check
  If v_fpxDTB1Memory=1 Then                                        ' Look to see if
player memory feature is on if it is then...
   DTB1fpxMemoryLoad()                                        ' Load in last
v_fpxDTB1CaseStart made from loss of previous ball
  ELSE                                  ' or if player memory feature is off
   FOR x = 1 To (v_fpxDTB1LightCount): v_fpxDTB1Lights(1,x).State =
BulbOff:NEXT                    ' We turn off all the lights first before we update the
scoring
   FOR x = 1 TO v_fpxDTB1BulbCount:v_fpxDTB1Bulbs(1,x).State =
BulbOn:NEXT                     ' We turn on all the Bulb lights first before we update
the scoring
   IF v_fpxDTB1CaseStart > 1 THEN v_fpxDTB1CaseStart = 1                        '
error catcher and prevents people from cheezing
   v_fpxDTB1Case=(v_fpxDTB1CaseStart)                                     ' Reset
v_fpxDTB1Case back to the beginning(user selectable top of script)
   CloseVaultDTB1fpx()                              ' runs the correct light routine and
restores proper light if target bank player memory feature is set to 1
  END IF
 END IF
End Sub
' Saves the Case settings for each player at the loss of a ball (Selectable by
user). This is used by all Vault items and called directly in the main engine
code
Sub DTB1fpxMemorySave()
  IF v_fpxDTB1On=1 THEN
  Select Case CurrentPlayer                          ' we see which player  is
playing.
   Case 1:m_fpxDTB1p1=v_fpxDTB1Case
   Case 2:m_fpxDTB1p2=v_fpxDTB1Case
   Case 3:m_fpxDTB1p3=v_fpxDTB1Case
   Case 4:m_fpxDTB1p4=v_fpxDTB1Case
  End Select
  IF VaultDTB1fpxDebug=1 THEN                             ' Dev Debug Code
   AddDebugText "DTB1fpxMemorySave()":AddDebugText " - fpxDTB1Case =   "
& (v_fpxDTB1Case):AddDebugText " - fpxDTB1Case =   " & (v_fpxDTB1Case)
   AddDebugText " - fpxDTB1Case =   " & (v_fpxDTB1Case):AddDebugText " -
fpxDTB1Case =   " & (v_fpxDTB1Case)
  END IF
  END IF
End Sub
' Loads the "Case" settings for each player at the start of a ball, and restores
that value back on his next ball if Memory=1.
' This is used by all Vault items and called directly in the main engine code
Sub DTB1fpxMemoryLoad()
 IF v_fpxDTB1On=1 THEN
  FOR x = 1 To (v_fpxDTB1LightCount): v_fpxDTB1Lights(1,x).State =
BulbOff:NEXT                 ' We need to restore the light memory for each player,
so all lights off
  FOR x = 1 TO v_fpxDTB1BulbCount:v_fpxDTB1Bulbs(1,x).State =
BulbOn:NEXT                  ' PF bulb lights should be on, but lets make sure
  Select Case CurrentPlayer                          ' Now we look at the memory for
the player that is up to see which light should be turned back on
   Case 1:v_fpxDTB1Case=m_fpxDTB1p1
   Case 2:v_fpxDTB1Case=m_fpxDTB1p2
   Case 3:v_fpxDTB1Case=m_fpxDTB1p3
   Case 4:v_fpxDTB1Case=m_fpxDTB1p4
  End Select
  IF v_fpxDTB1Case=> v_fpxDTB1CaseEnd THEN
v_fpxDTB1Case=v_fpxDTB1CaseStart                 ' Wraps back to caseStart if
over max CaseEnd
  Select Case v_fpxDTB1Case                          ' Restore the proper light the
extra snazzy way
   Case 0
   Case 1:v_fpxDTB1Lights(1,1).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 2:v_fpxDTB1Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 3:v_fpxDTB1Lights(1,3).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 4:v_fpxDTB1Lights(1,4).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 5:v_fpxDTB1Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
  End Select
  IF VaultDTB1fpxDebug=1 THEN                             ' Dev Debug Code
   AddDebugText " - m_fpxDTB1p1 =   " & (m_fpxDTB1p1):AddDebugText " -
m_fpxDTB1p2 =   " & (m_fpxDTB1p2)
```

```
   AddDebugText " - m_fpxDTB1p3 =   " & (m_fpxDTB1p3):AddDebugText " -
m_fpxDTB1p4 =   " & (m_fpxDTB1p4)
  END IF
 END IF
End Sub
'  /END fpx Vault DropTargetBank1
' *******************************************************************
' **                         Vault                                 **
' *******************************************************************
' *** These subroutines are Master subroutines and are used by all Vault items
***
' These are for use by the fpxEngine. Any other templates these subroutines
must be linked to within that templates code
' Run at NewBall. Also used as a blanket reset used for tilt, startup or game
over. This is used by all Vault items and called directly in the main engine
code.
' ResetTiltedState(): ResetForNewPlayerBall()
Sub VaultBallReset()
 IF v_fpxDTB1On=1 THEN DTB1fpxBallReset()

End Sub
' This resets the variables for the start of a game. This is used by all Vault items
and called directly in the main engine code
' ResetForNewGame() : EndOfGame()
Sub VaultGameReset()
 IF v_fpxDTB1On=1 THEN DTB1fpxGameReset():END IF

End Sub
' Closes ScoringEvent code pointed to by background timer for additional
instructions.
' TimerCloseScoringEventCase is the control variable
Sub TimerCloseScoringEvent_Expired()
End Sub
' Saves the v_fpxAVCase settings for each player at the loss of a ball
(Selectable by user).
Sub VaultAVMemorySave()
 IF v_fpxDTB1On=1 THEN DTB1fpxMemorySave()

End Sub
Sub VaultAVMemoryLoad()
 IF v_fpxDTB1On=1 THEN DTB1fpxMemoryLoad():END IF

end sub
```

vault_fpxDropTargetBank2

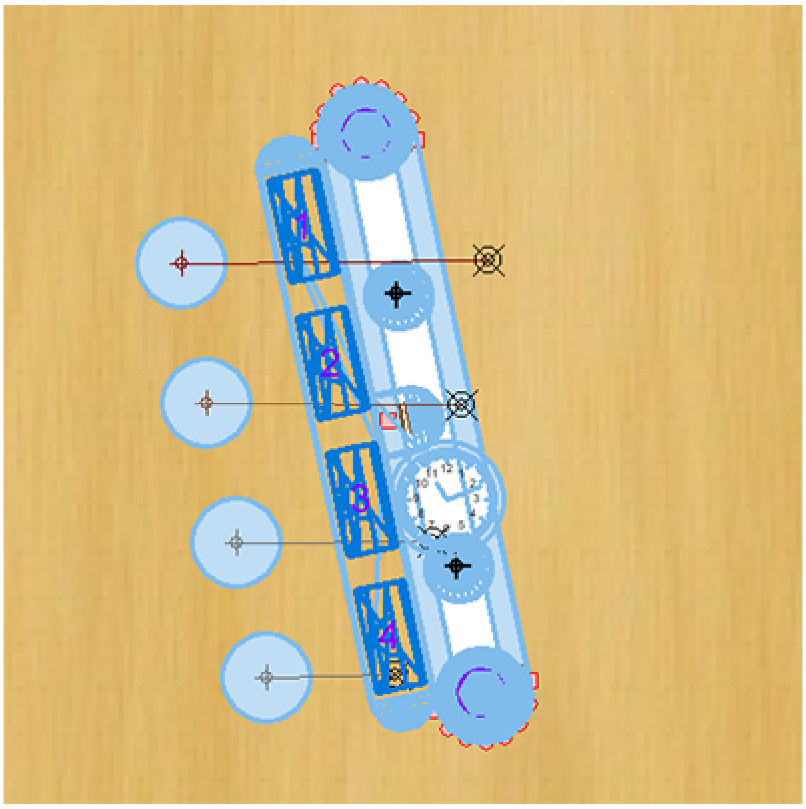
Vault - Drop Targets - vault_fpxDropTargetBank2

Vault - Drop Targets - vault_fpxDropTargetBank2

How to use this Vault Item in the fpxEngine

Using this Vault item and including it in your table design is very simple. It's just a couple copy and pastes from one fpt file to another!
Don't know How? Just click here.

vault_fpxDropTargetBank2



How it works

This Vault item has 5 stages of scoring, plus the ability to set player memory so a player can carry on through the stages with his next ball. The script is set to automatically reset the stages if the player reaches the 5th stage (25000 pts) in his previous ball so that player can have the chance to score a Extra Ball and a free game a second time.

User adjustments (pinsettings)

v_fpxDTB2CaseStart = 1
Initial Stage (Light) to be lit in this vault item for start of each ball,
- Set to 0 for no

lights at start or "hard". (You need to complete the Target bank routine once before the first light will be lit)
- Set to 1 to have the first light on at the start of a game

v_fpxDTB2Memory=1
Handles the Memory feature for DropTarget Score by each player
- Set to 1 if you want the player(s) DropTarget Bank made total per game in play carried over to his next ball in play.
- Set to 0 to have the DropTarget Bank Made total per game in play reset back to beginning with each new ball.

How to Change Scoring (Bank)

Sub AddVaultScoreDTB2fpx1()
This subroutine handles scoring when a bank is made. you can modify each stage by modifying the code with each case setting by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement fpxEngine AddScoringEvents

routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.
- Case 0: 5000 points,1000 added to Jackpot value , 1 added to bonus count at loss of a ball
- Case 1: 10000 points,10000 added to Jackpot value , 1 added to bonus count at loss of a ball
- Case 2: 25000 points (AddScoringEvent "25kAward")
- Case 3: Extra Ball (AddScoringEvent "ExtraBall")
- Case 4: Special (AddScoringEvent "Special")
- Case 5: 25000 points (AddScoringEvent "25kAward") This will repeat till loss of ball, then the case settings will reset back to the beginning case at the players next ball.

How to Change scoring (Single)

Sub DTB2fpx1_Hit()
This subroutine handles scoring when a single target is made. you can modify this just by changing the scoring code by adding different values between the brackets for higher scores
AddScore(1000)   ' Adds the value within the brackets to your player(s) score for Single Target Hit (not entire bank)
AddJackpot(1000)   ' Adds the value within the brackets to the Jackpot value which can be collected later on in the game
AddBonus(1)   ' Adds one bonus within the brackets to the End-Of-Ball Bonus Countdown routine

There is a rubber object behind the targets, which also generates points
Sub DTB2fpxRubber1_Hit()
AddScore(10)

List of Objects

These objects are needed for the entire script to work. If you accidentally delete one of these objects and then run the table, the table will throw up a error message telling you a object is missing. If you restore that object back, your table will run fine.

This is a list of all the objects needed:

Lights

LightDTB2fpx1
LightDTB2fpx2
LightDTB2fpx3
LightDTB2fpx4

Bulbs

BulbDTB2fpx1
BulbDTB2fpx2
PlasticDTB2fpx1

Objects

DTB2fpx1
DTB2fpxRubber1
TimerDTB2fpx1
t_DTB2fpx1

Additional Information

- All Drop Targets objects are set to Layer 1 in your editor. Each major group of Vault items are set to their own layer for easy moving and modification.
- Layer 9 are reserved for the top (header) and Bottom (slingshots/flippers/aprons)
- Layer 0 is reserved for the fpxEngine objects.
- All objects can be modified or rotated to fit your design. Consult the FP manual for making changes directly in the editor, or visit the Pinball Nirvana website if you get really stuck.

A Note

fpxEngine uses a shaped light as a "plastic", as well as a semi-transparent surface as a edge around it. This is set to a height of 32mm, to form a proper cover over posts/rubbers etc that the ball can roll underneath. The Plastic light is part of the bulb code, it will flash in unison with the bulbs underneath, and can not be deleted without causing a error message when you run the table. Plastics always have the word in front of them. Semi-transparent edges (surfaces) always has the "t_ in front of the name.

Vault Worksheet

The very nature of the fpxEngine is to allow absolute beginners to build and create complex tables with as little actual experience as possible, but you still need to learn a tiny bit of coding sometimes. Most of the Vault items will be based on the actual arcade table it came from, so it's just a quick copy and paste the design elements, the code, and then adding lights in the LightList manager.

More advanced coders though may wish to duplicate or modify this code to run different routines. This Vault item worksheet is the master template I use to create vault items from, so if you decide to duplicate or modify this vault item, you can change it easily using this worksheet. For more information, check the Using the Vault Worksheets page.

code:

```
' * Find and replace code keywords for duplication or new vault items
' replacement table vault name keyword: fpx
' replacement object keyword: DTB2
' Remark Keyword: DropTargetBank2
' Codeing Names Keywords
' v_ variables keyword
' m_ memory keyword
' * List of variables
' VaultDTB2fpxDebug
' v_fpxDTB2On
' v_fpxDTB2CaseStart
' v_fpxDTB2CaseEnd
' v_fpxDTB2CaseRepeat
' v_fpxDTB2Memory
' v_fpxDTB2Case
' v_fpxDTB2Lights(9,9)      ' For/Next Loops
' v_fpxDTB2Bulbs(9,9)       ' For/Next Loops
' v_fpxDTB2LightCount       ' Amount Lights in v_fpxDTB2Lights
' v_fpxDTB2BulbCount        ' Amount Bulbs in v_fpxDTB2Bulbs
' m_fpxDTB2p1
' m_fpxDTB2p2
' m_fpxDTB2p3
' m_fpxDTB2p4
' * Subroutines
' AddVaultDTB2fpx()       ' Main vault routine
' CloseVaultDTB2fpx()      ' Light control
' DTB2fpxGameReset()       ' New game
' DTB2fpxBallReset()      ' New ball in play
' DTB2fpxMemorySave()      ' saves players progress
' DTB2fpxMemoryLoad()      ' Restores players progress
' AddVaultScoreDTB2fpx
' fpxDTB2

' * Unique subroutines to this vault
' TimerTarget1fpx_Expired() ' Timer
' * Lights
' LightDTB2fpx1
' LightDTB2fpx2
' LightDTB2fpx3
' LightDTB2fpx4
' * Bulbs
' BulbDTB2fpx1
' BulbDTB2fpx2
' * Objects
' DTB2fpx1
' DTB2fpxRubber1
' TimerDTB2fpx1

' ---------------------------------------------------
' * fpx Vault DropTargetBank2 *
' ---------------------------------------------------
' For the fpxEngine.
' 1. Make sure all layers in the FP editor are "visible".
' Copy all the table elements in the editor from this fpt and then paste these
elements into your fpxEngine table.
' 2. Set User adjustments below to suit, then copy this entire script from this fpt
and paste into the script  for your fpx table.
'  I recommend  pasting  in the HIT SECTION.
' 3. Add Lights,plastics and bulbs to your LightList Manager in your fpx table.
' - Look for the names listed on the left side in the LightList Manager to have
"light", "bulb" or "plastic" as fpx will use these words
' at the beginning of the object name for all Vault Items
'  - If you do not know how to transfere the lights, both the fpx manual (in the
vault pages) and the Future Pinball manual explains how to do this.
' 4. Press "play"!
' * User adjustments  *
' Initial light to be lit in this vault item for start of each ball, set to 0 for no lights at
start or "hard".
' (You need to complete the routine once before the first light will be lit)
' Set to 1 to have the first light on at the start of a game
 v_fpxDTB2CaseStart = 1                              ' Variable to hold DropTarget
Bank starting value
' Handles the Memory feature for DropTarget Score (common pin setting)
' Set to one if you want the player(s) DropTarget Bank made total per game in
play carried over to his next ball in play.
' Set to 0 to have the DropTarget Bank Made total per game in play reset back
to beginning with each new ball,
' as set by v_fpxDTB2CaseStart  above.
 v_fpxDTB2Memory=1                                   ' Variable to hold DropTarget Bank
Score Value from ball to Ball (for each player)
' *** Scoring ***
' Allows you to set the scoring. Everything else is done for you within the engine.
You can add/change anything you want in here
' like adding a multiplier or turning on Inlane lights etc. See Beginners Guide in
manual
Sub AddVaultScoreDTB2fpx1()
 'IF VaultDebug=1 THEN AddDebugText "AddVaultScoreDTB2fpx1() "
          '  Dev Debug Code
 If v_fpxDTB2On=1 THEN
   Select Case v_fpxDTB2Case
    Case 0                           ' This is used as the starting TargetBank score
if v_fpxDTB2CaseStart = 0.
   AddScore(5000)                           '  Adds the value within the brackets to
your player(s) score
   AddJackpot(1000)                          '  Adds the value within the brackets to
```

```
the Jackpot value which can be collected later on in the game
  AddBonus(1)                              '  Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
   Case 1                             ' This is used as the starting TargetBank score
if v_fpxDTB2CaseStart = 1.
  AddScore(10000)
  AddJackpot(10000)
  AddBonus(1)
   Case 2
  AddScoringEvent "25kAward"
   Case 3
  AddScoringEvent "ExtraBall"                        ' Adds a Extra Ball
   Case 4
  AddScoringEvent "Special"                         ' Adds a credit (free game)
   Case Else                             ' Case else will score if case is higher than 5,
and will keep repeating
  AddScoringEvent "25kAward"
  End Select
 End If
End Sub
' Drop Target Hit . You need to set a scoring value here for a single drop target
hit that doesn't complete a bank
Sub DTB2fpx1_Hit()
 'IF VaultDebug=1 THEN AddDebugText "DTB2_Hit() "                    ' Play
the mechanical sound. this will sound even if game is tilted
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub : END IF
     ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
 AddScore(1000)                              ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
 AddJackpot(1000)                              ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
 AddBonus(1)                              ' Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
 If DTB2fpx1.Dropped = False Then AddMusicSet "drop" : Exit Sub : END IF
             ' Look to see if all targets in the bank is down, if not, play drop
sound,exit this subroutine
 If v_fpxDTB2On=1 THEN AddVaultDTB2fpx() : END IF                    ' if all
targets in this bank are down, go to bank made scoring subroutine
 set LastSwitchHit = DTB2fpx1                         ' FP code we can use if
needed. Just good programing
End Sub
' Rubber hit behind targets
Sub DTB2fpxRubber1_Hit()
 AddScore(10)                              ' Adds the value within the brackets to
your player(s) score
 AddMusicSet "sling"                          ' The music routine
End Sub
' Engine code, do not change or modify. This is needed by the engine to
reconize this vault item. Changing this will cause errors.
 v_fpxDTB2On=1                              '  KEEP THIS SET TO 1.
```

```
'  /END fpx Vault DropTargetBank2
' Placed in main fpx Vault Routines ******
' -----------------------------------------------------
' *** fpx Vault DropTargetBank2 ***
' -----------------------------------------------------
Const VaultDTB2fpxDebug=1                              ' Dev Debug. Set to 1 to
generate debug text for this vault item
Dim
v_fpxDTB2On
,v_fpxDTB2Memory
,v_fpxDTB2Lights
(9,9),v_fpxDTB2Bulbs(9,9),v_fpxDTB2LightCount,v_fpxDTB2BulbCount           '
Variables
Dim
v_fpxDTB2Case
,v_fpxDTB2CaseEnd,v_fpxDTB2CaseStart,v_fpxDTB2CaseRepeat
      ' Scoring variables
Dim m_fpxDTB2p1,m_fpxDTB2p2,m_fpxDTB2p3,m_fpxDTB2p4
     ' Player(s) memory
If v_fpxDTB2On=1 THEN                              ' Checks if vault item is being
used
 Set v_fpxDTB2Lights(1,1)=LightDTB2fpx1:Set v_fpxDTB2Lights(1,2)
=LightDTB2fpx2:Set v_fpxDTB2Lights(1,3)=LightDTB2fpx3:Set
v_fpxDTB2Lights(1,4)=LightDTB2fpx4
 Set v_fpxDTB2Bulbs(1,1)=BulbDTB2fpx1:Set v_fpxDTB2Bulbs(1,2)
=BulbDTB2fpx2:Set v_fpxDTB2Bulbs(1,3)=PlasticDTB2fpx1
' BAM bulb
 BulbDTB2fpx1EXT
.Brightness=(fpxBulbBrightness):BulbDTB2fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbDTB2fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbDTB2fpx2EXT
.Brightness=(fpxBulbBrightness):BulbDTB2fpx2EXT
.GlowRadius=(fpxBulbGlowRadius):BulbDTB2fpx2EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticDTB2fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticDTB2fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticDTB2fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
' light lens
 LightDTB2fpx1EXT
.Brightness=(fpxLensBrightness):LightDTB2fpx1EXT
.GlowRadius=(fpxLensGlowRadius):LightDTB2fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightDTB2fpx2EXT
.Brightness=(fpxLensBrightness):LightDTB2fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightDTB2fpx2EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightDTB2fpx3EXT
.Brightness=(fpxLensBrightness):LightDTB2fpx3EXT
.GlowRadius=(fpxLensGlowRadius):LightDTB2fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
```

```
 LightDTB2fpx4EXT
.Brightness=(fpxLensBrightness):LightDTB2fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightDTB2fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
 v_fpxDTB2LightCount = 4                        ' Number of Lens Lights used
(in set code) in FOR NEXT loops
 v_fpxDTB2BulbCount = 3                         ' Number of bulbs used (in set
code) in FOR NEXT loops
 v_fpxDTB2CaseEnd=5
 v_fpxDTB2CaseRepeat=5                           ' Last case number for
scoring, used to reset case to beginning if over at next ball
End If
' Main scoring routine. On target bank, v_fpxDTB2Case scores then increases
by one
Sub AddVaultDTB2fpx()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
      ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
 If v_fpxDTB2On=1 THEN                                       ' Check if vault item is
being used
  IF VaultDTB2fpxDebug=1 THEN  AddDebugText "AddVaultDTB2fpx()"
        ' Dev Debug Code
  FOR x = 1 To
(v_fpxDTB2LightCount):v_fpxDTB2Lights(1,x).State=BulbOff:NEXT
' All lens lights off first
  LockDisplay=0                                ' Clears any music priorty code
  DisplayBlinkInterval=(FlashForMSBlinkInterval+40)                      ' Sets new
interval for DT lights
  FOR x = 1 TO v_fpxDTB2BulbCount:v_fpxDTB2Bulbs(1,x).FlashForMs
(MusicIntervalTime), (FlashForMSBlinkInterval), BulbOn:NEXT         ' Bulbs
behind object will Blink rapidly for time set
  AddMusicSet "dropbank"                         ' Note. Default music is
overwritten by any AddScoringEvent code, so you need this default in case
there is no special scoring feature
  Select Case v_fpxDTB2Case
   Case 0 :                            ' v_fpx CaseStart = 0. no light lens till next
case
   Case 1 :                            ' v_fpx CaseStart = 1. first light lens used
   Case 2 : v_fpxDTB2Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn                ' Second Light lens used
   Case 3 : FlushDisplay()                            ' Have to add flushdisplay here
before extra ball/special/jackpot routines
   Case 4 : FlushDisplay() : v_fpxDTB2Lights(1,2).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval),BulbOn           ' Last light lens, so turn
back on LightLens 2 to keep repeating till loss of ball
   Case 5 : v_fpxDTB2Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn                ' After last case, this just keeps
repeating for the ball in play till ball lost
  End Select
  AddVaultScoreDTB2fpx1()                                       ' Goto scoring code
as set by developer. Must be before increase case code! (geez shiva)
```

```
  v_fpxDTB2Case = v_fpxDTB2Case+1                              ' Increases
value by 1 for select case routines
  IF v_fpxDTB2Case> v_fpxDTB2CaseEnd THEN
v_fpxDTB2Case=(v_fpxDTB2CaseEnd)                 ' Wraps back to caseStart if
over max CaseEnd
  TimerDTB2fpx1.Set True, 1000                    ' Resets targets (after set
delay time)
  DTB2fpxMemorySave()                              ' Save to that players
memory
  CloseVaultDTB2fpx()                        ' Run closing light routine

 END IF
End Sub
' Timer to reset target bank after a delay.
Sub TimerDTB2fpx1_Expired()
 TimerDTB2fpx1.Enabled = False
 If v_fpxDTB2On=1 THEN                                   ' Check if vault item is
being used
  IF VaultDTB2fpxDebug=1 THEN  AddDebugText "TimerDTB2fpx1_Expired() "
                 ' Dev Debug Code
  DTB2fpx1.SolenoidPulse : PlaySound "DTargetReset"                       '
Resets drop target bank
 END IF
End sub
' runs the correct light routine and restores proper light if v_fpxDTB2Memory=1
Sub CloseVaultDTB2fpx()
  If v_fpxDTB2On=1 THEN                                  ' Only if this feature is
set to "1" and nothing else
   IF VaultDTB2fpxDebug=1 THEN  AddDebugText "CloseVaultDTB2fpx()
":AddDebugText  " ":END IF               ' Dev Debug Code
   FOR x = 1 To (v_fpxDTB2LightCount): v_fpxDTB2Lights(1,x).State =
BulbOff:NEXT
   Select Case v_fpxDTB2Case                                  ' Sets next light to
display based on Case
    Case 0: Exit Sub
    Case 1: v_fpxDTB2Lights(1,1).State = BulbOn:
v_fpxDTB2Lights(1,1).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn       ' 1k, light 10k light for next case
    Case 2: v_fpxDTB2Lights(1,2).State = BulbOn:
v_fpxDTB2Lights(1,2).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn       ' 20k
    Case 3: v_fpxDTB2Lights(1,3).State = BulbOn:
v_fpxDTB2Lights(1,3).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn       ' Extra Ball
    Case 4: v_fpxDTB2Lights(1,4).State = BulbOn:
v_fpxDTB2Lights(1,4).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn       ' Special
    Case 5:v_fpxDTB2Lights(1,2).State = BulbOn:
v_fpxDTB2Lights(1,2).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn      ' Any thing over
   END SELECT
  END IF
End Sub
' Game Start. This resets the variables. This is used by all Vault items and
called directly in the main engine code
Sub DTB2fpxGameReset()
 IF v_fpxDTB2On=1 THEN                               ' Check if vault item is being
used
  IF VaultDTB2fpxDebug=1 THEN  AddDebugText
"DTB2fpxGameReset()":AddDebugText  " ":END IF              ' Dev Debug Code
  TimerDTB2fpx1.Set True, 20                              ' Resets targets (after set
delay time)
  FOR x = 1 To (v_fpxDTB2LightCount): v_fpxDTB2Lights(1,x).State =
BulbOff:NEXT                    ' We turn off all the lights first before we update the
scoring
  FOR x = 1 TO v_fpxDTB2BulbCount:v_fpxDTB2Bulbs(1,x).State =
BulbOn:NEXT                     ' We turn on all the Bulb lights first before we update
the scoring
  v_fpxDTB2Case=(v_fpxDTB2CaseStart)                             ' Resets scoring
case setting back to starting default (set in vault user options)
  m_fpxDTB2p1=(v_fpxDTB2CaseStart):m_fpxDTB2p2=(v_fpxDTB2CaseStart):
m_fpxDTB2p3=(v_fpxDTB2CaseStart):m_fpxDTB2p4=(v_fpxDTB2CaseStart)    '
Resets variables back to initial starting point
 END IF
End Sub
' Run at NewBall. Also used as a blanket reset used for tilt, startup or game
over. This is used by all Vault items and called directly in the main engine code
Sub DTB2fpxBallReset()
 IF v_fpxDTB2On=1 THEN                               ' Check if vault item is being
used
  IF VaultDTB2fpxDebug=1 THEN  AddDebugText "CloseVaultDTB2fpx()
":AddDebugText  " ":END IF              ' Dev Debug Code
  TimerDTB2fpx1.Set True, 20                              ' Resets targets (after set
delay time) Note we do this here before the memory check
  If v_fpxDTB2Memory=1 Then                                        ' Look to see if
player memory feature is on if it is then...
   DTB2fpxMemoryLoad()                                    ' Load in last
v_fpxDTB2CaseStart made from loss of previous ball
  ELSE                                 ' or if player memory feature is off
   FOR x = 1 To (v_fpxDTB2LightCount): v_fpxDTB2Lights(1,x).State =
BulbOff:NEXT                    ' We turn off all the lights first before we update the
scoring
   FOR x = 1 TO v_fpxDTB2BulbCount:v_fpxDTB2Bulbs(1,x).State =
BulbOn:NEXT                     ' We turn on all the Bulb lights first before we update
the scoring
   IF v_fpxDTB2CaseStart > 1 THEN v_fpxDTB2CaseStart = 1                  '
error catcher and prevents people from cheezing
   v_fpxDTB2Case=(v_fpxDTB2CaseStart)                                 ' Reset
v_fpxDTB2Case back to the beginning(user selectable top of script)
   CloseVaultDTB2fpx()                              ' runs the correct light routine and
restores proper light if target bank player memory feature is set to 1
  END IF
 END IF
End Sub
' Saves the Case settings for each player at the loss of a ball (Selectable by
user). This is used by all Vault items and called directly in the main engine
code
Sub DTB2fpxMemorySave()
  IF v_fpxDTB2On=1 THEN
  Select Case CurrentPlayer                                ' we see which player  is
playing.
   Case 1:m_fpxDTB2p1=v_fpxDTB2Case
   Case 2:m_fpxDTB2p2=v_fpxDTB2Case
   Case 3:m_fpxDTB2p3=v_fpxDTB2Case
   Case 4:m_fpxDTB2p4=v_fpxDTB2Case
  End Select
  IF VaultDTB2fpxDebug=1 THEN                              ' Dev Debug Code
   AddDebugText "DTB2fpxMemorySave()":AddDebugText " - fpxDTB2Case =   "
& (v_fpxDTB2Case):AddDebugText " - fpxDTB2Case =   " & (v_fpxDTB2Case)
   AddDebugText " - fpxDTB2Case =   " & (v_fpxDTB2Case):AddDebugText " -
fpxDTB2Case =   " & (v_fpxDTB2Case)
  END IF
  END IF
End Sub
' Loads the "Case" settings for each player at the start of a ball, and restores
that value back on his next ball if Memory=1.
' This is used by all Vault items and called directly in the main engine code
Sub DTB2fpxMemoryLoad()
 IF v_fpxDTB2On=1 THEN
  FOR x = 1 To (v_fpxDTB2LightCount): v_fpxDTB2Lights(1,x).State =
BulbOff:NEXT                   ' We need to restore the light memory for each player,
so all lights off
  FOR x = 1 TO v_fpxDTB2BulbCount:v_fpxDTB2Bulbs(1,x).State =
BulbOn:NEXT                    ' PF bulb lights should be on, but lets make sure
  Select Case CurrentPlayer                                ' Now we look at the memory for
the player that is up to see which light should be turned back on
   Case 1:v_fpxDTB2Case=m_fpxDTB2p1
   Case 2:v_fpxDTB2Case=m_fpxDTB2p2
   Case 3:v_fpxDTB2Case=m_fpxDTB2p3
   Case 4:v_fpxDTB2Case=m_fpxDTB2p4
  End Select
  IF v_fpxDTB2Case=> v_fpxDTB2CaseEnd THEN
v_fpxDTB2Case=v_fpxDTB2CaseStart                  ' Wraps back to caseStart if
over max CaseEnd
  Select Case v_fpxDTB2Case                               ' Restore the proper light the
extra snazzy way
   Case 0
   Case 1:v_fpxDTB2Lights(1,1).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 2:v_fpxDTB2Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 3:v_fpxDTB2Lights(1,3).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 4:v_fpxDTB2Lights(1,4).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 5:v_fpxDTB2Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
  End Select
  IF VaultDTB2fpxDebug=1 THEN                                   ' Dev Debug Code
   AddDebugText " - m_fpxDTB2p1 =   " & (m_fpxDTB2p1):AddDebugText " -
m_fpxDTB2p2 =   " & (m_fpxDTB2p2)
   AddDebugText " - m_fpxDTB2p3 =   " & (m_fpxDTB2p3):AddDebugText " -
m_fpxDTB2p4 =   " & (m_fpxDTB2p4)
  END IF
 END IF
End Sub
'  /END fpx Vault DropTargetBank2
' ********************************************************************
' **                              Vault                             **
' ********************************************************************
' *** These subroutines are Master subroutines and are used by all Vault items
***
' These are for use by the fpxEngine. Any other templates these subroutines
must be linked to within that templates code
' Run at NewBall. Also used as a blanket reset used for tilt, startup or game
over. This is used by all Vault items and called directly in the main engine
code.
' ResetTiltedState(): ResetForNewPlayerBall()
Sub VaultBallReset()
 IF v_fpxDTB2On=1 THEN DTB2fpxBallReset()

End Sub
' This resets the variables for the start of a game. This is used by all Vault items
and called directly in the main engine code
' ResetForNewGame() : EndOfGame()
Sub VaultGameReset()
 IF v_fpxDTB2On=1 THEN DTB2fpxGameReset():END IF

End Sub
' Closes ScoringEvent code pointed to by background timer for additional
instructions.
' TimerCloseScoringEventCase is the control variable
Sub TimerCloseScoringEvent_Expired()
End Sub
' Saves the v_fpxAVCase settings for each player at the loss of a ball
(Selectable by user).
Sub VaultAVMemorySave()
 IF v_fpxDTB2On=1 THEN DTB2fpxMemorySave()

End Sub
Sub VaultAVMemoryLoad()
 IF v_fpxDTB2On=1 THEN DTB2fpxMemoryLoad():END IF

end sub
```

Vault - Stand Up Targets

vault_fpxTarget3Bank1

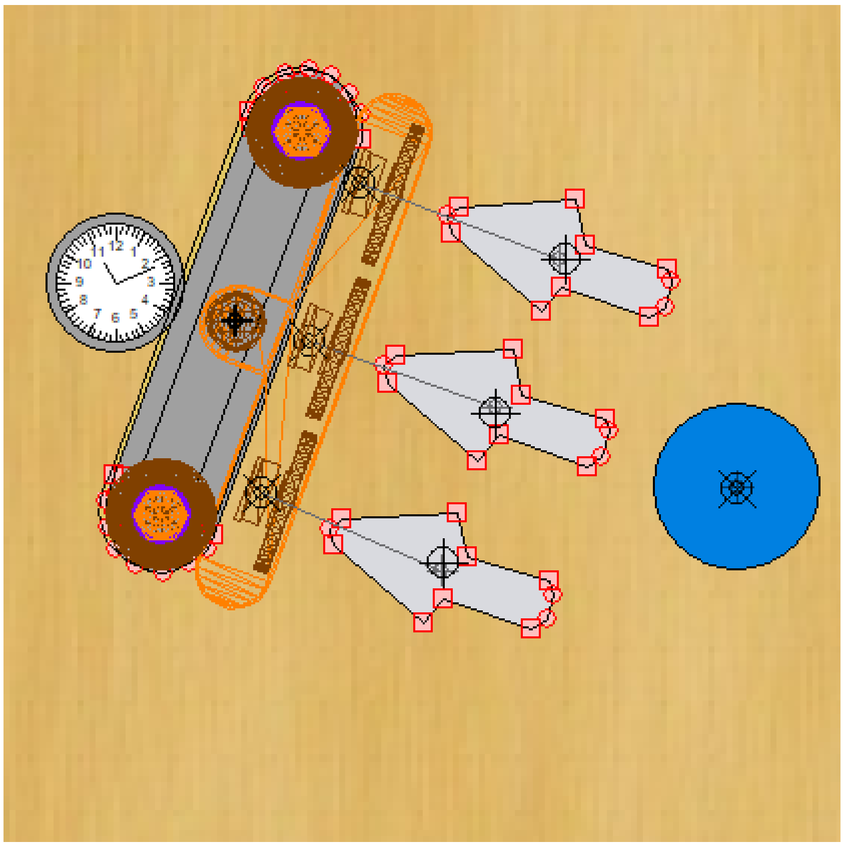
Vault - Stand Up Targets - vault_fpxTarget3Bank1

Vault - Stand Up Targets - vault_fpxTarget3Bank1

How to use this Vault Item in the fpxEngine

Using this Vault item and including it in your table design is very simple. It's just a couple copy and pastes from one fpt file to another!
Don't know How? Just click here.

vault_fpxTarget3Bank1



How it works

This Vault item has 2 stages of scoring, plus the ability to set player memory so a player can carry on through the stages with his next ball. This Vault item also has 2 different light displays, a flashing light lens routine for the target lights (Williams style) or a off state that turns on with a target hit.(Bally style)

User adjustments (pinsettings)

v_fpxTarget3Bank1Memory=1
- Handles the Memory feature for Target Score (common pin setting)
- Set to one if you want the player(s) Target Bank made total per game in play carried over to his next ball in play.
- Set to 0 to have the Target Bank Made total per game in play reset back to beginning with each new ball,

v_fpxTarget3Bank1CaseStart = 1
- Initial light to be lit in this vault item

for start of each ball
- Set to 0 for no lights at start or "hard". (You need to complete the routine once before the first light will be lit)
- Set to 1 to have the first light on at the start of a game

How to Change Scoring (Bank)

Sub AddVaultScoreTarget3Bank1fpx()
This subroutine handles scoring when a bank is made. you can modify each stage by modifying the code with each case setting by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement fpxEngine AddScoringEvents routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.
- Case 0: 5000 points,1000 added to Jackpot value , 1 added to

	bonus count at loss of a ball - Case 1: 10000 points,10000 added to Jackpot value , 1 added to bonus count at loss of a ball - Case 2: Awards Jackpot Value (AddScoringEvent "Jackpot") - Case Else: This is a error catcher, you should just leave this alone.

How to Change scoring (Single)

Sub AddVaultScoreTarget3Bank1fpx()
This subroutine handles scoring when a bank is made. you can modify each stage by modifying the code with each case setting by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement fpxEngine AddScoringEvents routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.
- Case 1: 5000 points, 5000 added to Jackpot, and 1 bonus added to the Bonus Countdown when the player loses his ball.
- Case 2: 10000 points, 10000 added to Jackpot, and 1 bonus added to the Bonus Countdown when the player loses his ball. The player is also awarded a Jackpot value.

Sub Target3Bank1fpx1_Hit(),Sub Target3Bank1fpx2_Hit(),Sub Target3Bank1fpx3_Hit()
This subroutine handles scoring when a single target is made. you can modify this just by changing the scoring code by adding different values between the brackets for higher scores

This is for when the target is not made, or is not "lit".
```
AddScore(1000) ' Adds the value within the brackets to your player(s) score for Single Target Hit (not entire bank)
AddJackpot(1000)        ' Adds the value within the brackets to the Jackpot value which can be collected later on in the game
AddBonus(1)             ' Adds one bonus within the brackets to the End-Of-Ball Bonus Countdown routine
```

When a target is already made or "lit", the scoring will be different
```
AddScore(5000) ' Adds the value within the brackets to your player(s) score for Single Target Hit (not entire bank)
AddJackpot(5000)        ' Adds the value within the brackets to the Jackpot value which can be collected later on in the game
AddBonus(1)             ' Adds one bonus within the brackets to the End-Of-Ball Bonus Countdown routine
```

List of Objects

These objects are needed for the entire script to work. If you accidentally delete one of these objects and then run the table, the table will throw up a error message telling you a object is missing. If you restore that object back, your table will run fine.

This is a list of all the objects needed:

Lights

LightTarget3Bank1fpx1
LightTarget3Bank1fpx2
LightTarget3Bank1fpx3
LightTarget3Bank1fpx4

Bulbs

BulbTarget3Bank1fpx1
PlasticTarget3Bank1fpx1

Objects

Target3Bank1fpx1
Target3Bank1fpx2
Target3Bank1fpx3
TimerTarget3Bank1fpx
t_Target3Bank1fpx1

Additional Information

- All Stand Up Targets objects are set to Layer 2 in your editor. Each major group of Vault items are set to their own layer for easy moving and modification.
- Layer 9 are reserved for the top (header) and Bottom (slingshots/flippers/aprons)
- Layer 0 is reserved for the fpxEngine objects.
- All objects can be modified or rotated to fit your design. Consult the FP manual for making changes directly in the editor, or visit the Pinball Nirvana website if you get really stuck.

A Note

fpxEngine uses a shaped light as a "plastic", as well as a semi-transparent surface as a edge around it. This is set to a height of 32mm, to form a proper cover over posts/rubbers etc that the ball can roll underneath. The Plastic light is part of the bulb code, it will flash in unison with the bulbs underneath, and can not be deleted without causing a error message when you run the table. Plastics always have the word in front of the file name. Semi-transparent edges (surfaces) always has the "t_ in front of the name.

Vault Worksheet

The very nature of the fpxEngine is to allow absolute beginners to build and create complex tables with as little actual experience as possible, but you still need to learn a tiny bit of coding sometimes. Most of the Vault items will be based on the actual arcade table it came from, so it's just a quick copy and paste the design elements, the code, and then adding lights in the LightList manager.

More advanced coders though may wish to duplicate or modify this code to run different routines. This Vault item worksheet is the master template I use to create vault items from, so if you decide to duplicate or modify this vault item, you can change it easily using this worksheet. For more information, check the Using the Vault Worksheets page.

code:

```
' * Find and replace code keywords for duplication or new vault items
' replacement table vault name keyword: fpx
' replacement object keyword: Target3Bank1
' Remark Keyword: Target3Bank1
' Coding Names Keywords
' v_ variables keyword
' m_ memory keyword
' VaultTarget3Bank1fpxDebug      ' Debug
' v_fpxTarget3Bank1On       ' Turns on or off this vault item. (Prevents errors if no objects on the table)
' v_fpxTarget3Bank1CaseStart     ' Initial starting case(position) for Bank made
' v_fpxTarget3Bank1Memory      ' Handles memory for each player
' v_fpxTarget3Bank1Case       ' Bank scoring control case
' v_fpxTarget3Bank1UseWilliamsLights   ' Flashing chaser lights (wms) or no flashing (bally) lights
' v_fpxTarget3Bank1BlinkIntervals    ' Interval of flashing lights
' v_fpxTarget3Bank1CaseEnd      ' Last case (position) for each bank made
' v_fpxTarget3Bank1CaseRepeat
' v_fpxTarget3Bank1Lights(9,9)     ' Use in FOR/NEXT loops. 9 groups with 9 objects in each group max.
' v_fpxTarget3Bank1Bulbs(9,9)     ' Use in FOR/NEXT loops. 9 groups with 9 objects in each group max.
' v_fpxTarget3Bank1LightCount     ' Total amount of lights used
' v_fpxTarget3Bank1BulbCount     ' Total amount of bulbs used
' m_fpxTarget3Bank1p1       ' Variable for Player 1 memory
' m_fpxTarget3Bank1p2       ' Variable for Player 2 memory
' m_fpxTarget3Bank1p3       ' Variable for Player 3 memory
' m_fpxTarget3Bank1p4       ' Variable for Player 4 memory
' * Subroutines
' AddVaultScoreTarget3Bank1fpx()    ' Main Bank Scoring (user adjustable)
' Target3Bank1fpx1_Hit()      ' Hit routine for object (user adjustable)
' Target3Bank1fpx2_Hit()      ' Hit routine for object (user adjustable)
' Target3Bank1fpx3_Hit()      ' Hit routine for object (user adjustable)
' AddVaultTarget3Bank1fpx()      ' Main bank scoring routine (Vault Engine)
' CloseVaultTarget3Bank1fpx()     ' Used in player memory to restore the proper lights
' Target3Bank1fpxLightControl()     ' Main lighting routine
' Target3Bank1fpxGameReset()     ' Start game routine
' Target3Bank1fpxBallReset()     ' New ball routine
' Target3Bank1fpxMemoryLoad()     ' Load player memory
' Target3Bank1fpxMemorySave()     ' Saves player memory
' * Unique subroutines to this vault
' TimerTarget3Bank1fpx_Expired()    ' Timer to reset after set delay.
' * Lights
' LightTarget3Bank1fpx1
' LightTarget3Bank1fpx2
' LightTarget3Bank1fpx3
' LightTarget3Bank1fpx4
' * Bulbs
' BulbTarget3Bank1fpx1
' PlasticTarget3Bank1fpx1
' * Objects
' Target3Bank1fpx1
' Target3Bank1fpx2
' Target3Bank1fpx3
' TimerTarget3Bank1fpx
 t_Target3Bank1fpx1
'
' * fpx Vault Target3Bank1 *
' --------------------------
' For the fpxEngine.
' 1. Make sure all layers in the FP editor are "visible".
' Copy all the table elements in the editor from this fpt and then paste these
elements into your fpxEngine table.
' 2. Set User adjustments below to suit, then copy this entire script from this fpt
and paste into the script  for your fpx table.
'  I recommend  pasting  in the HIT SECTION.
' 3. Add Lights,plastics and bulbs to your LightList Manager in your fpx table.
' - Look for the names listed on the left side in the LightList Manager to have
"light", "bulb" or "plastic" as fpx will use these words
' at the beginning of the object name for all Vault Items
'  - If you do not know how to transfere the lights, both the fpx manual (in the
vault pages) and the Future Pinball manual explains how to do this.
' 4. Press "play"!
' * User adjustments  *
' Initial light to be lit in this vault item for start of each ball, set to 0 for no lights at
start or "hard".
' (You need to complete the routine once before the first light will be lit)
' Set to 1 to have the first light on at the start of a game
  v_fpxTarget3Bank1CaseStart = 1                              ' Variable to hold
DropTarget Bank starting value
' Handles the Memory feature for Target Score (common pin setting)
' Set to one if you want the player(s) Target Bank made total per game in play
carried over to his next ball in play.
' Set to 0 to have the Target Bank Made total per game in play reset back to
beginning with each new ball,
' as set by v_fpxTarget3Bank1CaseStart  above.
  v_fpxTarget3Bank1Memory=1                                   ' Variable to hold Target
Bank Score Value from ball to Ball (for each player)
' Each of the main companies in the 1980's producing pinball tables had a
unique style to their games. Williams with their targets had the "chaser" lights
' in front of their targets blinking rapidly that turned solid (or "on") when a ball
struck a target (like in Firepower), while Bally had their lights turned completely
' Off and then turned on the lights with a target hit. This pinsetting simulates
both styles, by default this is set to Williams style
  v_fpxTarget3Bank1UseWilliamsLights=1                        ' 0=Bally(no light on
before hit) 1=Williams (light is blinking before hit)
' *** Scoring ***
' Allows you to set the scoring. Everything else is done for you within the engine.
You can add/change anything you want in here
' like adding a multiplier or turning on Inlane lights etc. See Beginners Guide in
manual
Sub AddVaultScoreTarget3Bank1fpx()
 If v_fpxTarget3Bank1On=1 THEN
   Select Case v_fpxTarget3Bank1Case
    Case 0                              ' This is used as the starting TargetBank score
if v_fpxTarget3Bank1CaseStart = 0.
   AddScore(5000)                             '  Adds the value within the brackets to
your player(s) score
   AddJackpot(1000)                             '  Adds the value within the brackets
to the Jackpot value which can be collected later on in the game
   AddBonus(1)                          '  Adds one bonus within the brackets to
the End-Of-Ball Bonus Countdown routine
    Case 1                              ' This is used as the starting TargetBank score
if v_fpxTarget3Bank1CaseStart = 1.
   AddScore(10000)
   AddJackpot(10000)
   AddBonus(1)
    Case 2
   AddScoringEvent "Jackpot"                         ' Collects the Jackpot value

    Case Else                              ' Case else will score if case is higher than
5, and will keep repeating
   v_fpxTarget3Bank1Case=1 : AddVaultScoreTarget3Bank1fpx()                  '
Error catcher, this loops back to case 1
   End Select
 End If
End Sub
' Target Hit . You need to set a scoring value here for a single target hit that
doesn't complete a bank
Sub Target3Bank1fpx1_Hit()
 If v_fpxTarget3Bank1On=1 THEN
  IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub : END IF
      ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
  IF LightTarget3Bank1fpx1.State=BulbOff OR
LightTarget3Bank1fpx1.State=BulbBlink Then                ' Note we check for light
state for both Bally and wms styles
   LightTarget3Bank1fpx1.State=BulbOn                         ' Need to turn on the
light first so AddVaultTarget3Bank1fpx() will work if all 3 lights are set to BulbOn
   AddScore(1000)                             ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
   AddJackpot(1000)                             ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
   AddMusicSet "drop"
  Else                              ' If the target light is already lit
   AddScore(5000)                             ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
   AddJackpot(5000)                             ' Adds the value within the brackets to
```

```
the Jackpot value which can be collected later on in the game
   AddBonus(1)                              ' Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
   AddMusicSet "drop"
  END IF
  v_fpxTarget3Bank1Lights(1,1).FlashForMs (MusicIntervalTime),
(v_fpxTarget3Bank1BlinkIntervals/4),BulbOn            ' Fancy blinking when hit
then bulb is lit
  AddVaultTarget3Bank1fpx()                             ' Routine to check if Bank is
made.
  set LastSwitchHit = Target3Bank1fpx1                          ' FP code we can use
if needed. Just good programing
 END IF
End Sub
Sub Target3Bank1fpx2_Hit()
 If v_fpxTarget3Bank1On=1 THEN
  IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub : END IF
       ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
  IF LightTarget3Bank1fpx2.State=BulbOff OR
LightTarget3Bank1fpx2.State=BulbBlink Then              ' Note we check for light
state for both Bally and wms styles
   LightTarget3Bank1fpx2.State=BulbOn                         ' Need to turn on the
light first so AddVaultTarget3Bank1fpx() will work if all 3 lights are set to BulbOn
   AddScore(1000)                             ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
   AddJackpot(1000)                            ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
   AddMusicSet "drop"
  Else                          ' If the target light is already lit
   AddScore(5000)                             ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
   AddJackpot(5000)                            ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
   AddBonus(1)                              ' Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
   AddMusicSet "drop"
  END IF
  v_fpxTarget3Bank1Lights(1,2).FlashForMs (MusicIntervalTime),
(v_fpxTarget3Bank1BlinkIntervals/4),BulbOn            ' Fancy blinking when hit
then bulb is lit
  AddVaultTarget3Bank1fpx()                             ' Routine to check if Bank is
made.
  set LastSwitchHit = Target3Bank1fpx2                          ' FP code we can use
if needed. Just good programing
 END IF
End Sub
Sub Target3Bank1fpx3_Hit()
 If v_fpxTarget3Bank1On=1 THEN
  IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub : END IF
       ' See if in tilt state or no game in progress to exit the subroutine and stop
```

```
any scoring
  IF LightTarget3Bank1fpx3.State=BulbOff OR
LightTarget3Bank1fpx3.State=BulbBlink Then              ' Note we check for light
state for both Bally and wms styles
   LightTarget3Bank1fpx3.State=BulbOn                      ' Need to turn on the
light first so AddVaultTarget3Bank1fpx() will work if all 3 lights are set to BulbOn
   AddScore(1000)                                 ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
   AddJackpot(1000)                                ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
   AddMusicSet "drop"
  Else                             ' If the target light is already lit
   AddScore(5000)                                 ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
   AddJackpot(5000)                                ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
   AddBonus(1)                                ' Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
   AddMusicSet "drop"
  END IF
  v_fpxTarget3Bank1Lights(1,3).FlashForMs (MusicIntervalTime),
(v_fpxTarget3Bank1BlinkIntervals/4),BulbOn           ' Fancy blinking when hit
then bulb is lit
  AddVaultTarget3Bank1fpx()                              ' Routine to check if Bank is
made.
  set LastSwitchHit = Target3Bank1fpx3                        ' FP code we can use
if needed. Just good programing
 END IF
End Sub
' Engine code, do not change or modify. This is needed by the engine to
reconize this vault item. Changing this will cause errors.
 v_fpxTarget3Bank1On=1                             ' KEEP THIS SET TO 1.
' ----------------------------
'  END fpx Vault Target3Bank1
' ----------------------------
' -------------------------------------------------------
' *** fpx Vault Target(3)Bank1 ***
' -------------------------------------------------------
Const VaultTarget3Bank1fpxDebug=0
Dim
v_fpxTarget3Bank1On
,v_fpxTarget3Bank1CaseStart
,v_fpxTarget3Bank1Memory
,v_fpxTarget3Bank1Case,v_fpxTarget3Bank1UseWilliamsLights     ' Variables
Dim
v_fpxTarget3Bank1BlinkIntervals
,v_fpxTarget3Bank1CaseEnd,v_fpxTarget3Bank1CaseRepeat
Dim
v_fpxTarget3Bank1Lights
(
9
,
9),v_fpxTarget3Bank1Bulbs
(9,9),v_fpxTarget3Bank1LightCount,v_fpxTarget3Bank1BulbCount          '
Variables used in For/Next loops
Dim
m_fpxTarget3Bank1p1
,m_fpxTarget3Bank1p2,m_fpxTarget3Bank1p3,m_fpxTarget3Bank1p4
' Variables for Player(s) memory
If v_fpxTarget3Bank1On=1 THEN                              ' Checks if vault item is
being used
 Set v_fpxTarget3Bank1Lights(1,1)=LightTarget3Bank1fpx1:Set
v_fpxTarget3Bank1Lights(1,2)=LightTarget3Bank1fpx2:Set
v_fpxTarget3Bank1Lights(1,3)=LightTarget3Bank1fpx3 ' Define lights used in
FOR/NEXT loops
 Set v_fpxTarget3Bank1Lights(1,4)=LightTarget3Bank1fpx4
 Set v_fpxTarget3Bank1Bulbs(1,1)=BulbTarget3Bank1fpx1
 Set v_fpxTarget3Bank1Bulbs(1,2)=PlasticTarget3Bank1fpx1                     '
Define Bulbs used in FOR/NEXT loops
 v_fpxTarget3Bank1LightCount = 4:v_fpxTarget3Bank1BulbCount = 2
   ' Number of lights and number of bulbs used in FOR NEXT loops
 v_fpxTarget3Bank1CaseEnd =2
 v_fpxTarget3Bank1CaseRepeat=1                              ' Amount of scoring
cases before wraps to CaseStart
' BAM bulb
 BulbTarget3Bank1fpx1EXT
.Brightness=(fpxBulbBrightness):BulbTarget3Bank1fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbTarget3Bank1fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticTarget3Bank1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticTarget3Bank1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticTarget3Bank1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
' light lens
 LightTarget3Bank1fpx1EXT
.Brightness=(fpxLensBrightness):LightTarget3Bank1fpx1EXT
.GlowRadius=(fpxLensGlowRadius):LightTarget3Bank1fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightTarget3Bank1fpx2EXT
.Brightness=(fpxLensBrightness):LightTarget3Bank1fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightTarget3Bank1fpx2EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightTarget3Bank1fpx3EXT
.Brightness=(fpxLensBrightness):LightTarget3Bank1fpx3EXT
.GlowRadius=(fpxLensGlowRadius):LightTarget3Bank1fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightTarget3Bank1fpx4EXT
.Brightness=(fpxLensBrightness):LightTarget3Bank1fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightTarget3Bank1fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
End If
' Main scoring routine. On target bank, v_fpxTarget3Bank1Case increases by
one, then runs that number in the matching case
```

```
Sub AddVaultTarget3Bank1fpx()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
     ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
 If v_fpxTarget3Bank1On=1 THEN                                        ' Check if vault
item is being used
  IF VaultTarget3Bank1fpxDebug=1 THEN  AddDebugText
"AddVaultTarget3Bank1fpx()"                   ' Dev Debug Code
  If LightTarget3Bank1fpx1.State=BulbOn AND
LightTarget3Bank1fpx2.State=BulbOn AND
LightTarget3Bank1fpx3.State=BulbOn THEN
   FOR x = 1 To
(v_fpxTarget3Bank1LightCount):v_fpxTarget3Bank1Lights
(1,x).State=BulbOff:NEXT               ' All lens lights off first
   LockDisplay=0                                  ' Clears any music priorty code
   FOR x = 1 TO
v_fpxTarget3Bank1BulbCount:v_fpxTarget3Bank1Bulbs(1,x).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval), BulbOn:NEXT   ' Bulbs behind object
will Blink rapidly for time set
  ' Note. Default music is overwritten by any AddScoringEvent code, so you
need this default in case there is no special scoring feature
   AddMusicSet "dropbank"
   Target3Bank1fpxLightControl()                           ' Reset lights
   Select Case v_fpxTarget3Bank1Case
    Case 0 :                                ' v_fpx CaseStart = 0. no light lens till next
case
    Case 1 :  v_fpxTarget3Bank1Lights(1,4).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn           ' First bank made, Turn on Jackpot light
    Case 2 :                                ' Second Bank made
    Case Else : v_fpxTarget3Bank1Case=(v_fpxTarget3Bank1CaseStart)
       ' error catcher
   End Select
   AddVaultScoreTarget3Bank1fpx()                                        ' Goto scoring
code as set by user. Must be before increase case code! (geez shiva)
   v_fpxTarget3Bank1Case = v_fpxTarget3Bank1Case+1                                    '
Increases value by 1 for select case routines
   IF v_fpxTarget3Bank1Case> v_fpxTarget3Bank1CaseEnd THEN
v_fpxTarget3Bank1Case=(v_fpxTarget3Bank1CaseStart)         ' Wraps back to
CaseStart if over max CaseEnd
   Target3Bank1fpxMemorySave()                                      ' Save Case Value
to that players memory
  END IF
 END IF
End Sub
' Timer to reset target bank after a delay.
Sub TimerTarget3Bank1fpx_Expired()
 TimerTarget3Bank1fpx.Enabled = False
 If v_fpxTarget3Bank1On=1 THEN
  IF VaultTarget3Bank1fpxDebug=1 THEN  AddDebugText
"TimerTarget3Bank1fpx_Expired()"                ' Dev Debug Code
  FOR x = 1 To (v_fpxTarget3Bank1LightCount):
```

```
v_fpxTarget3Bank1Lights(1,x).State = BulbOff:NEXT              ' We need to
restore the light memory for each player, so all lights off
  IF v_fpxTarget3Bank1Case=2 THEN
v_fpxTarget3Bank1Lights(1,4).State=BulbOn                 ' Jackpot Light blinks
rapidly for time set
  Target3Bank1fpxLightControl()                          ' Restore the target lights to
start again
 END IF
End sub
' runs the correct light routine and restores proper light if
v_fpxTarget3Bank1Memory=1
Sub CloseVaultTarget3Bank1fpx()
 If v_fpxTarget3Bank1On=1 THEN                                        ' Only if this
feature is set to "1" and nothing else
  IF VaultTarget3Bank1fpxDebug=1 THEN  AddDebugText
"CloseVaultTarget3Bank1fpx()"                 ' Dev Debug Code
  Target3Bank1fpxLightControl()                          ' Restore the target lights to
start again
  Select Case v_fpxTarget3Bank1Case                                     ' Sets next
light to display based on Case
   Case 0:
   Case 1:
   Case 2: v_fpxTarget3Bank1Lights(1,4).State = BulbOn                    ' Turn on
Jackpot Light
   Case Else : v_fpxTarget3Bank1Case=1                               ' error catcher
  END SELECT
  END IF
End Sub
' Main bank scoring routine
Sub Target3Bank1fpxLightControl()
 If v_fpxTarget3Bank1On=1 THEN
  IF VaultTarget3Bank1fpxDebug=1 THEN  AddDebugText
"Target3Bank1fpxLightControl()"                ' Dev Debug Code
                   ' Only if this feature is set to "1" and nothing else
  v_fpxTarget3Bank1BlinkIntervals=150                              ' Blink Intervals
  LightTarget3Bank1fpx1.BlinkInterval =
(v_fpxTarget3Bank1BlinkIntervals):LightTarget3Bank1fpx2.BlinkInterval =
(v_fpxTarget3Bank1BlinkIntervals)
  LightTarget3Bank1fpx3.BlinkInterval =
(v_fpxTarget3Bank1BlinkIntervals):LightTarget3Bank1fpx4.BlinkInterval =
(v_fpxTarget3Bank1BlinkIntervals)
  LightTarget3Bank1fpx1.BlinkPattern =
"100":LightTarget3Bank1fpx2.BlinkPattern =
"010":LightTarget3Bank1fpx3.BlinkPattern =
"001":LightTarget3Bank1fpx4.BlinkPattern = "010" ' Blink Pattern
  IF v_fpxTarget3Bank1UseWilliamsLights THEN
   FOR x = 1 To (v_fpxTarget3Bank1LightCount-1):
v_fpxTarget3Bank1Lights(1,x).State = BulbBlink:NEXT            ' (WMS) all lights
Blink
  ELSE
   FOR x = 1 To (v_fpxTarget3Bank1LightCount-1):
```

```
v_fpxTarget3Bank1Lights(1,x).State = BulbOff:NEXT            ' (Bally) Turns off
lights
  END IF
 END IF
End Sub
' Start Game Routine
Sub Target3Bank1fpxGameReset()
 IF v_fpxTarget3Bank1On=1 THEN                          ' Check if vault item is
being used
  IF VaultTarget3Bank1fpxDebug=1 THEN  AddDebugText
"Target3Bank1fpxGameReset()"                  ' Dev Debug Code
  FOR x = 1 To (v_fpxTarget3Bank1LightCount):
v_fpxTarget3Bank1Lights(1,x).State = BulbOff:NEXT            ' We turn off all the
Bulb lights first before we update the scoring
  v_fpxTarget3Bank1Case=(v_fpxTarget3Bank1CaseStart)                        '
Resets scoring case setting back to starting default (set in vault user options)
  m_fpxTarget3Bank1p1=(v_fpxTarget3Bank1CaseStart):m_fpxTarget3Bank1p2=
(v_fpxTarget3Bank1CaseStart)          ' Resets variables back to initial starting
point
  m_fpxTarget3Bank1p3=(v_fpxTarget3Bank1CaseStart):m_fpxTarget3Bank1p4=
(v_fpxTarget3Bank1CaseStart)
 END IF
End Sub

' Run from NewBall(). Also used as a blanket reset called from tilt, startup and
game over subroutines.
Sub Target3Bank1fpxBallReset()
 IF v_fpxTarget3Bank1On=1 THEN                          ' Check if vault item is
being used
  IF VaultTarget3Bank1fpxDebug=1 THEN  AddDebugText
"Target3Bank1fpxBallReset()"                  ' Dev Debug Code
  TimerTarget3Bank1fpx.Set True, 20                          ' Resets targets (after
set delay time) Note we do this here before the memory check
  If v_fpxTarget3Bank1Memory=1 Then                                    ' Look to see if
player memory feature is on if it is then...
   Target3Bank1fpxMemoryLoad()                                   ' Load in last
v_fpxTarget3Bank1CaseStart made from loss of previous ball
  ELSE                              ' or if player memory feature is off
   FOR x = 1 To (v_fpxTarget3Bank1LightCount):
v_fpxTarget3Bank1Lights(1,x).State = BulbOff:NEXT            ' We turn off all the
lights first before we update the scoring
   FOR x = 1 TO
v_fpxTarget3Bank1BulbCount:v_fpxTarget3Bank1Bulbs(1,x).State =
BulbOn:NEXT              ' We turn on all the Bulb lights first before we update the
scoring
   IF v_fpxTarget3Bank1CaseStart > 1 THEN v_fpxTarget3Bank1CaseStart = 1
              ' error catcher and prevents people from cheezing
   v_fpxTarget3Bank1Case=(v_fpxTarget3Bank1CaseStart)
  ' Reset v_fpxTarget3Bank1Case back to the beginning(user selectable top of
script)
   CloseVaultTarget3Bank1fpx()                         ' runs the correct light routine
```

```
and restores proper light if target bank player memory feature is set to 1
  END IF
 END IF
End Sub
' Only called from Target3Bank1fpxBallReset() but keep here in case future
vaults need this
Sub Target3Bank1fpxMemoryLoad()
 IF v_fpxTarget3Bank1On=1 THEN
  IF VaultTarget3Bank1fpxDebug=1 THEN  AddDebugText
"Target3Bank1fpxMemoryLoad()"                ' Dev Debug Code
  FOR x = 1 TO v_fpxTarget3Bank1BulbCount:v_fpxTarget3Bank1Bulbs(1,x).State
= BulbOn:NEXT               ' PF bulb lights should be on, but lets make sure
  Select Case CurrentPlayer                         ' Now we look at the memory for
the player that is up to see which light should be turned back on
   Case 1:v_fpxTarget3Bank1Case=m_fpxTarget3Bank1p1
   Case 2:v_fpxTarget3Bank1Case=m_fpxTarget3Bank1p2
   Case 3:v_fpxTarget3Bank1Case=m_fpxTarget3Bank1p3
   Case 4:v_fpxTarget3Bank1Case=m_fpxTarget3Bank1p4
  End Select
  IF v_fpxTarget3Bank1Case> v_fpxTarget3Bank1CaseEnd THEN
v_fpxTarget3Bank1Case=(v_fpxTarget3Bank1CaseStart)         ' Wraps back to
caseStart if over max CaseEnd
  Select Case v_fpxTarget3Bank1Case                          ' Restore the proper
light the extra snazzy way
   Case 0 : FOR x = 1 To
(v_fpxTarget3Bank1LightCount):v_fpxTarget3Bank1Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxTarget3Bank1BlinkIntervals/4),BulbOn:NEXT
   Case 1 : FOR x = 1 To
(v_fpxTarget3Bank1LightCount):v_fpxTarget3Bank1Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxTarget3Bank1BlinkIntervals/4),BulbOn:NEXT
   Case 2 : FOR x = 1 To
(v_fpxTarget3Bank1LightCount):v_fpxTarget3Bank1Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxTarget3Bank1BlinkIntervals/4),BulbOn:NEXT
   Case Else:FOR x = 1 To
(v_fpxTarget3Bank1LightCount):v_fpxTarget3Bank1Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxTarget3Bank1BlinkIntervals/4),BulbOn:NEXT
  End Select
  IF VaultTarget3Bank1fpxDebug=1 AND v_fpxTarget3Bank1On=1 THEN
              ' Dev Debug Code
   AddDebugText " - m_fpxTarget3Bank1p1 =   " &
(m_fpxTarget3Bank1p1):AddDebugText " - m_fpxTarget3Bank1p2 =   " &
(m_fpxTarget3Bank1p2):AddDebugText " - m_fpxTarget3Bank1p3 =   " &
(m_fpxTarget3Bank1p3):AddDebugText " - m_fpxTarget3Bank1p4 =   " &
(m_fpxTarget3Bank1p4)
  END IF
 END IF
End Sub
' Only called from AddVaultTarget3Bank1fpx() but keep here in case future
vaults need this
Sub Target3Bank1fpxMemorySave()
  IF v_fpxTarget3Bank1On=1 THEN                              ' Check if vault item is
being used
```

```
  IF VaultTarget3Bank1fpxDebug=1 THEN  AddDebugText
"Target3Bank1fpxMemorySave()"                ' Dev Debug Code
  Select Case CurrentPlayer                           ' we see which player  is
playing.
   Case 1:m_fpxTarget3Bank1p1=v_fpxTarget3Bank1Case
   Case 2:m_fpxTarget3Bank1p2=v_fpxTarget3Bank1Case
   Case 3:m_fpxTarget3Bank1p3=v_fpxTarget3Bank1Case
   Case 4:m_fpxTarget3Bank1p4=v_fpxTarget3Bank1Case
  End Select
  END IF
End Sub
'  /END fpx Vault Target(3)Bank1
' ********************************************************************
' **                               Vault                                    **
' ********************************************************************
' *** These subroutines are Master subroutines and are used by all Vault items
***
' Run at NewBall. Also used as a blanket reset used for tilt, startup or game
over. This is used by all Vault items and called directly in the main engine code
Sub VaultBallReset()
 ' Standup Targets
 IF v_fpxTarget3Bank1On=1 THEN Target3Bank1fpxBallReset() : END IF
            ' v_fpxTarget3Bank1On
End Sub
' Game Start. This resets the variables. This is used by all Vault items and
called directly in the main engine code
Sub VaultGameReset()
 ' StandupTargets
 IF v_fpxTarget3Bank1On=1 THEN Target3Bank1fpxGameReset():END IF
End Sub
' Closes ScoringEvent code pointed to by background timer for additional
instructions. This is used by all Vault items and called directly in the main
engine code
' TimerCloseScoringEventCase is the control variable
Sub TimerCloseScoringEvent_Expired()                       ' Restore the
scoring display
End Sub
' Saves the v_fpxAVCase settings for each player at the loss of a ball
(Selectable by user). This is used by all Vault items and called directly in the
main engine code
Sub VaultAVMemorySave()
 '  StandupTargets
 IF v_fpxTarget3Bank1On=1 THEN Target3Bank1fpxMemorySave()
End Sub
' Loads the v_fpx "Case" settings for each player at the start of a ball, and
restores that value back on his next ball if Memory=1.
' This is used by all Vault items and called directly in the main engine code
Sub VaultAVMemoryLoad()
 '  StandupTargets
 IF v_fpxTarget3Bank1On=1 THEN Target3Bank1fpxMemoryLoad():END IF
end sub
```

vault_fpxTarget3Bank2

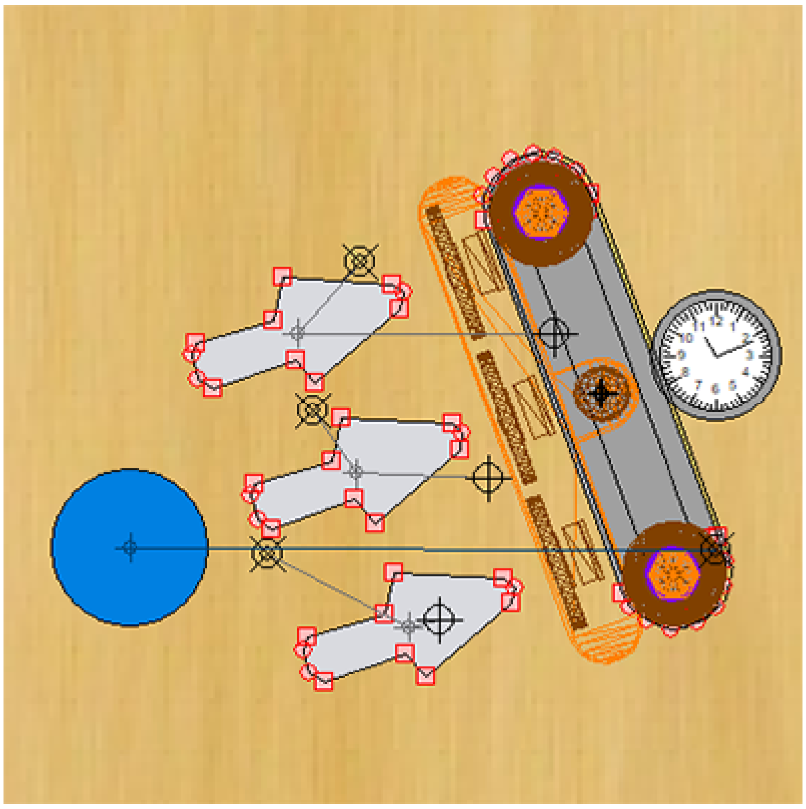
Vault - Stand Up Targets - vault_fpxTarget3Bank2

Vault - Stand Up Targets - vault_fpxTarget3Bank2

How to use this Vault Item in the fpxEngine

Using this Vault item and including it in your table design is very simple. It's just a couple copy and pastes from one fpt file to another!
Don't know How? Just click here.

vault_fpxTarget3Bank2



How it works

This Vault item has 2 stages of scoring, plus the ability to set player memory so a player can carry on through the stages with his next ball. This Vault item also has 2 different light displays, a flashing light lens routine for the target lights (Williams style) or a off state that turns on with a target hit.(Bally style)

User adjustments (pinsettings)

v_fpxTarget3Bank2Memory=1
- Handles the

Memory feature for Target Score (common pin setting)
- Set to one if you want the player(s) Target Bank made total per game in play carried over to his next ball in play.
- Set to 0 to have the Target Bank Made total per game in play reset back to beginning with each new ball,

v_fpxTarget3Bank2CaseStart = 1
- Initial light to be lit in this vault item for start of each ball
- Set to 0 for no lights at start or "hard". (You need to complete the routine once before the first light will be lit)
- Set to 1 to have the first light on at the start of a game

How to Change Scoring (Bank)

Sub AddVaultScoreTarget3Bank2fpx()
This subroutine handles scoring when a bank is made. you can modify each stage by modifying the code with each case setting by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement

fpxEngine AddScoringEvents routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.
- Case 0: 5000 points,1000 added to Jackpot value , 1 added to bonus count at loss of a ball
- Case 1: 10000 points,10000 added to Jackpot value , 1 added to bonus count at loss of a ball
- Case 2: Awards Jackpot Value (AddScoringEvent "Jackpot")
- Case Else: This is a error catcher, you should just leave this alone.

⚙ How to Change scoring (Single)

Sub AddVaultScoreTarget3Bank2fpx()

This subroutine handles scoring when a bank is made. you can modify each stage by modifying the code with each case setting by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement fpxEngine AddScoringEvents routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.
- Case 1: 5000 points, 5000 added to Jackpot, and 1 bonus added to the Bonus Countdown when the player loses his ball.
- Case 2: 10000 points, 10000 added to Jackpot, and 1 bonus added to the Bonus Countdown when the player loses his ball. The player is also awarded a Jackpot value.

Sub Target3Bank1fpx1_Hit(),Sub Target3Bank2fpx2_Hit(),Sub Target3Bank1fpx3_Hit()

This subroutine handles scoring when a single target is made. you can modify this just by changing the scoring code by adding different values between the brackets for higher scores

This is for when the target is not made, or is not "lit".

AddScore(1000) ' Adds the value within the brackets to your player(s) score for Single Target Hit (not entire bank)
AddJackpot(1000)   ' Adds the value within the brackets to the Jackpot value which can be collected later on in the game
AddBonus(1)   ' Adds one bonus within the brackets to the End-Of-Ball Bonus Countdown routine

When a target is already made or "lit", the scoring will be different

AddScore(5000) ' Adds the value within the brackets to your player(s) score for Single Target Hit (not entire bank)

AddJackpot(5000) ' Adds the value within the brackets to the Jackpot value which can be collected later on in the game

AddBonus(1) ' Adds one bonus within the brackets to the End-Of-Ball Bonus Countdown routine

List of Objects

These objects are needed for the entire script to work. If you accidentally delete one of these objects and then run the table, the table will throw up a error message telling you a object is missing. If you restore that object back, your table will run fine.

This is a list of all the objects needed:

Lights

LightTarget3Bank2fpx1
LightTarget3Bank2fpx2
LightTarget3Bank2fpx3
LightTarget3Bank2fpx4

Bulbs

BulbTarget3Bank2fpx1
PlasticTarget3Bank2fpx1

Objects

Target3Bank2fpx1
Target3Bank2fpx2
Target3Bank2fpx3
TimerTarget3Bank2fpx
t_Target3Bank2fpx1

Additional Information

- All Stand Up Targets objects are set to Layer 2 in your editor. Each major group of Vault items are set to their own layer for easy moving and modification.
- Layer 9 are reserved for the top (header) and Bottom (slingshots/flippers/aprons)
- Layer 0 is reserved for the fpxEngine objects.
- All objects can be modified or rotated to fit your design. Consult the FP manual for making changes directly in the editor, or visit the Pinball Nirvana website if you get really stuck.

A Note

fpxEngine uses a shaped light as a "plastic", as well as a semi-transparent surface as a edge around it. This is set to a height of 32mm, to form a proper cover over posts/rubbers etc that the ball can roll underneath. The Plastic light is part of the bulb code, it will flash in unison with the bulbs underneath, and can not be deleted without causing a error message when you run the table. Plastics always have the word in front of the file name. Semi-transparent edges (surfaces) always has the "t_ in front of the name.

Vault Worksheet

The very nature of the fpxEngine is to allow absolute beginners to build and create complex tables with as little actual experience as possible, but you still need to learn a tiny bit of coding sometimes. Most of the Vault items will be based on the actual arcade table it came from, so it's just a quick copy and paste the design elements, the code, and then adding lights in the LightList manager.

More advanced coders though may wish to duplicate or modify this code to run different routines. This Vault item worksheet is the master template I use to create vault items from, so if you decide to duplicate or modify this vault item, you can change it easily using this worksheet. For more information, check the Using the Vault Worksheets page.

code:

```
' * Find and replace code keywords for duplication or new vault items
' replacement table vault name keyword: fpx
' replacement object keyword: Target3Bank1
' Remark Keyword: Target3Bank2
' Codeing Names Keywords
' v_ variables keyword
' m_ memory keyword
' VaultTarget3Bank2fpxDebug      ' Debug
' v_fpxTarget3Bank2On       ' Turns on or off this vault item. (Prevents errors if no objects on the table)
' v_fpxTarget3Bank2CaseStart     ' Initial starting case(position) for Bank made
' v_fpxTarget3Bank2Memory      ' Handles memory for each player
' v_fpxTarget3Bank2Case      ' Bank scoring control case
' v_fpxTarget3Bank2UseWilliamsLights   ' Flashing chaser lights (wms) or no flashing (bally) lights
' v_fpxTarget3Bank2BlinkIntervals    ' Interval of flashing lights
' v_fpxTarget3Bank2CaseEnd      ' Last case (position) for each bank made
' v_fpxTarget3Bank2CaseRepeat
' v_fpxTarget3Bank2Lights(9,9)     ' Use in FOR/NEXT loops. 9 groups with 9 objects in each group max.
' v_fpxTarget3Bank2Bulbs(9,9)     ' Use in FOR/NEXT loops. 9 groups with 9 objects in each group max.
' v_fpxTarget3Bank2LightCount     ' Total amount of lights used
' v_fpxTarget3Bank2BulbCount     ' Total amount of bulbs used
' m_fpxTarget3Bank2p1       ' Variable for Player 1 memory
' m_fpxTarget3Bank2p2       ' Variable for Player 2 memory
' m_fpxTarget3Bank2p3       ' Variable for Player 3 memory
' m_fpxTarget3Bank2p4        ' Variable for Player 4 memory
' * Subroutines
' AddVaultScoreTarget3Bank2fpx()    ' Main Bank Scoring (user adjustable)
' Target3Bank2fpx1_Hit()      ' Hit routine for object (user adjustable)
' Target3Bank2fpx2_Hit()      ' Hit routine for object (user adjustable)
' Target3Bank2fpx3_Hit()      ' Hit routine for object (user adjustable)
' AddVaultTarget3Bank2fpx()      ' Main bank scoring routine (Vault Engine)
' CloseVaultTarget3Bank2fpx()     ' Used in player memory to restore the proper lights
' Target3Bank2fpxLightControl()     ' Main lighting routine
' Target3Bank2fpxGameReset()     ' Start game routine
' Target3Bank2fpxBallReset()     ' New ball routine
' Target3Bank2fpxMemoryLoad()     ' Load player memory
' Target3Bank2fpxMemorySave()     ' Saves player memory
' * Unique subroutines to this vault
' TimerTarget3Bank2fpx_Expired()    ' Timer to reset after set delay.
' * Lights
' LightTarget3Bank2fpx1
' LightTarget3Bank2fpx2
' LightTarget3Bank2fpx3
' LightTarget3Bank2fpx4
' * Bulbs
' BulbTarget3Bank2fpx1
' PlasticTarget3Bank2fpx1
' * Objects
' Target3Bank2fpx1
' Target3Bank2fpx2
' Target3Bank2fpx3
' TimerTarget3Bank2fpx
' t_Target3Bank2fpx1

' * fpx Vault Target3Bank2 *
' --------------------------
' For the fpxEngine.
' 1. Make sure all layers in the FP editor are "visible".
' Copy all the table elements in the editor from this fpt and then paste these
elements into your fpxEngine table.
' 2. Set User adjustments below to suit, then copy this entire script from this fpt
and paste into the script  for your fpx table.
'  I recommend  pasting  in the HIT SECTION.
' 3. Add Lights,plastics and bulbs to your LightList Manager in your fpx table.
' - Look for the names listed on the left side in the LightList Manager to have
"light", "bulb" or "plastic" as fpx will use these words
' at the beginning of the object name for all Vault Items
'  - If you do not know how to transfere the lights, both the fpx manual (in the
vault pages) and the Future Pinball manual explains how to do this.
' 4. Press "play"!
' * User adjustments  *
' Initial light to be lit in this vault item for start of each ball, set to 0 for no lights at
start or "hard".
' (You need to complete the routine once before the first light will be lit)
' Set to 1 to have the first light on at the start of a game
 v_fpxTarget3Bank2CaseStart = 1                              ' Variable to hold
DropTarget Bank starting value
' Handles the Memory feature for Target Score (common pin setting)
' Set to one if you want the player(s) Target Bank made total per game in play
carried over to his next ball in play.
' Set to 0 to have the Target Bank Made total per game in play reset back to
beginning with each new ball,
' as set by v_fpxTarget3Bank2CaseStart  above.
 v_fpxTarget3Bank2Memory=1                                ' Variable to hold Target
Bank Score Value from ball to Ball (for each player)
```

```
' Each of the main companies in the 1980's producing pinball tables had a
unique style to their games. Williams with their targets had the "chaser" lights
' in front of their targets blinking rapidly that turned solid (or "on") when a ball
struck a target (like in Firepower), while Bally had their lights turned completely
' Off and then turned on the lights with a target hit. This pinsetting simulates
both styles, by default this is set to Williams style
  v_fpxTarget3Bank2UseWilliamsLights=1                    ' 0=Bally(no light on
before hit) 1=Williams (light is blinking before hit)
' *** Scoring ***
' Allows you to set the scoring. Everything else is done for you within the engine.
You can add/change anything you want in here
' like adding a multiplier or turning on Inlane lights etc. See Beginners Guide in
manual
Sub AddVaultScoreTarget3Bank2fpx()
 If v_fpxTarget3Bank2On=1 THEN
   Select Case v_fpxTarget3Bank2Case
    Case 0                          ' This is used as the starting TargetBank score
if v_fpxTarget3Bank2CaseStart = 0.
   AddScore(5000)                        '  Adds the value within the brackets to
your player(s) score
   AddJackpot(1000)                       '  Adds the value within the brackets
to the Jackpot value which can be collected later on in the game
   AddBonus(1)                        '  Adds one bonus within the brackets to
the End-Of-Ball Bonus Countdown routine
    Case 1                          ' This is used as the starting TargetBank score
if v_fpxTarget3Bank2CaseStart = 1.
   AddScore(10000)
   AddJackpot(10000)
   AddBonus(1)
    Case 2
   AddScoringEvent "Jackpot"                       ' Collects the Jackpot value

    Case Else                          ' Case else will score if case is higher than
5, and will keep repeating
   v_fpxTarget3Bank2Case=1 : AddVaultScoreTarget3Bank2fpx()                 '
Error catcher, this loops back to case 1
   End Select
 End If
End Sub
' Target Hit . You need to set a scoring value here for a single target hit that
doesn't complete a bank
Sub Target3Bank2fpx1_Hit()
 If v_fpxTarget3Bank2On=1 THEN
  IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub : END IF
      ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
  IF LightTarget3Bank2fpx1.State=BulbOff OR
LightTarget3Bank2fpx1.State=BulbBlink Then              ' Note we check for light
state for both Bally and wms styles
   LightTarget3Bank2fpx1.State=BulbOn                   ' Need to turn on the
light first so AddVaultTarget3Bank2fpx() will work if all 3 lights are set to BulbOn
```

```
  AddScore(1000)                          ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
  AddJackpot(1000)                        ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
  AddMusicSet "drop"
 Else                          ' If the target light is already lit
  AddScore(5000)                          ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
  AddJackpot(5000)                        ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
  AddBonus(1)                         ' Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
  AddMusicSet "drop"
 END IF
 v_fpxTarget3Bank2Lights(1,1).FlashForMs (MusicIntervalTime),
(v_fpxTarget3Bank2BlinkIntervals/4),BulbOn           ' Fancy blinking when hit
then bulb is lit
 AddVaultTarget3Bank2fpx()                        ' Routine to check if Bank is
made.
 set LastSwitchHit = Target3Bank2fpx1                          ' FP code we can use
if needed. Just good programing
 END IF
End Sub
Sub Target3Bank2fpx2_Hit()
 If v_fpxTarget3Bank2On=1 THEN
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub : END IF
     ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
 IF LightTarget3Bank2fpx2.State=BulbOff OR
LightTarget3Bank2fpx2.State=BulbBlink Then              ' Note we check for light
state for both Bally and wms styles
  LightTarget3Bank2fpx2.State=BulbOn                       ' Need to turn on the
light first so AddVaultTarget3Bank2fpx() will work if all 3 lights are set to BulbOn
  AddScore(1000)                          ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
  AddJackpot(1000)                        ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
  AddMusicSet "drop"
 Else                          ' If the target light is already lit
  AddScore(5000)                          ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
  AddJackpot(5000)                        ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
  AddBonus(1)                         ' Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
  AddMusicSet "drop"
 END IF
 v_fpxTarget3Bank2Lights(1,2).FlashForMs (MusicIntervalTime),
(v_fpxTarget3Bank2BlinkIntervals/4),BulbOn           ' Fancy blinking when hit
then bulb is lit
 AddVaultTarget3Bank2fpx()                        ' Routine to check if Bank is
```

```
made.
  set LastSwitchHit = Target3Bank2fpx2                          ' FP code we can use
if needed. Just good programing
 END IF
End Sub
Sub Target3Bank2fpx3_Hit()
 If v_fpxTarget3Bank2On=1 THEN
  IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub : END IF
       ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
  IF LightTarget3Bank2fpx3.State=BulbOff OR
LightTarget3Bank2fpx3.State=BulbBlink Then                 ' Note we check for light
state for both Bally and wms styles
   LightTarget3Bank2fpx3.State=BulbOn                         ' Need to turn on the
light first so AddVaultTarget3Bank2fpx() will work if all 3 lights are set to BulbOn
   AddScore(1000)                                 ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
   AddJackpot(1000)                                 ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
   AddMusicSet "drop"
  Else                              ' If the target light is already lit
   AddScore(5000)                                 ' Adds the value within the brackets to
your player(s) score for Single Target Hit (not entire bank)
   AddJackpot(5000)                                 ' Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
   AddBonus(1)                               ' Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
   AddMusicSet "drop"
  END IF
  v_fpxTarget3Bank2Lights(1,3).FlashForMs (MusicIntervalTime),
(v_fpxTarget3Bank2BlinkIntervals/4),BulbOn             ' Fancy blinking when hit
then bulb is lit
  AddVaultTarget3Bank2fpx()                              ' Routine to check if Bank is
made.
  set LastSwitchHit = Target3Bank2fpx3                          ' FP code we can use
if needed. Just good programing
 END IF
End Sub
' Engine code, do not change or modify. This is needed by the engine to
reconize this vault item. Changing this will cause errors.
 v_fpxTarget3Bank2On=1                             ' KEEP THIS SET TO 1.
' ----------------------------
'  END fpx Vault Target3Bank2
' ----------------------------
' -----------------------------------------------------------
' *** fpx Vault Target3Bank2 ***
' -----------------------------------------------------------
Const VaultTarget3Bank2fpxDebug=0
Dim
v_fpxTarget3Bank2On
,v_fpxTarget3Bank2CaseStart
```

```
,v_fpxTarget3Bank2Memory
,v_fpxTarget3Bank2Case,v_fpxTarget3Bank2UseWilliamsLights     ' Variables
Dim
v_fpxTarget3Bank2BlinkIntervals
,v_fpxTarget3Bank2CaseEnd,v_fpxTarget3Bank2CaseRepeat
Dim
v_fpxTarget3Bank2Lights
(
9
,
9),v_fpxTarget3Bank2Bulbs
(9,9),v_fpxTarget3Bank2LightCount,v_fpxTarget3Bank2BulbCount          '
Variables used in For/Next loops
Dim
m_fpxTarget3Bank2p1
,m_fpxTarget3Bank2p2,m_fpxTarget3Bank2p3,m_fpxTarget3Bank2p4
' Variables for Player(s) memory
If v_fpxTarget3Bank2On=1 THEN                              ' Checks if vault item is
being used
 Set v_fpxTarget3Bank2Lights(1,1)=LightTarget3Bank2fpx1:Set
v_fpxTarget3Bank2Lights(1,2)=LightTarget3Bank2fpx2:Set
v_fpxTarget3Bank2Lights(1,3)=LightTarget3Bank2fpx3 ' Define lights used in
FOR/NEXT loops
 Set v_fpxTarget3Bank2Lights(1,4)=LightTarget3Bank2fpx4
 Set v_fpxTarget3Bank2Bulbs(1,1)=BulbTarget3Bank2fpx1:Set
v_fpxTarget3Bank2Bulbs(1,2)=PlasticTarget3Bank2fpx1          ' Define Bulbs
used in FOR/NEXT loops
 v_fpxTarget3Bank2LightCount = 4:v_fpxTarget3Bank2BulbCount = 2
   ' Number of lights and number of bulbs used in FOR NEXT loops
 v_fpxTarget3Bank2CaseEnd =2                              ' Amount of scoring cases
before wraps to CaseStart
 v_fpxTarget3Bank2CaseRepeat=1
' BAM bulb
 BulbTarget3Bank2fpx1EXT
.Brightness=(fpxBulbBrightness):BulbTarget3Bank2fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbTarget3Bank2fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticTarget3Bank2fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticTarget3Bank2fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticTarget3Bank2fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
' light lens
 LightTarget3Bank2fpx1EXT
.Brightness=(fpxLensBrightness):LightTarget3Bank2fpx1EXT
.GlowRadius=(fpxLensGlowRadius):LightTarget3Bank2fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightTarget3Bank2fpx2EXT
.Brightness=(fpxLensBrightness):LightTarget3Bank2fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightTarget3Bank2fpx2EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightTarget3Bank2fpx3EXT
.Brightness=(fpxLensBrightness):LightTarget3Bank2fpx3EXT
```

```
.GlowRadius=(fpxLensGlowRadius):LightTarget3Bank2fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightTarget3Bank2fpx4EXT
.Brightness=(fpxLensBrightness):LightTarget3Bank2fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightTarget3Bank2fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
End If
' Main scoring routine. On target bank, v_fpxTarget3Bank2Case increases by
one, then runs that number in the matching case
Sub AddVaultTarget3Bank2fpx()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
      ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
 If v_fpxTarget3Bank2On=1 THEN                                    ' Check if vault
item is being used
  IF VaultTarget3Bank2fpxDebug=1 THEN  AddDebugText
"AddVaultTarget3Bank2fpx()"                 ' Dev Debug Code
  If LightTarget3Bank2fpx1.State=BulbOn AND
LightTarget3Bank2fpx2.State=BulbOn AND
LightTarget3Bank2fpx3.State=BulbOn THEN
   FOR x = 1 To
(v_fpxTarget3Bank2LightCount):v_fpxTarget3Bank2Lights
(1,x).State=BulbOff:NEXT               ' All lens lights off first
   LockDisplay=0                                ' Clears any music priorty code
   FOR x = 1 TO
v_fpxTarget3Bank2BulbCount:v_fpxTarget3Bank2Bulbs(1,x).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval), BulbOn:NEXT   ' Bulbs behind object
will Blink rapidly for time set
  ' Note. Default music is overwritten by any AddScoringEvent code, so you
need this default in case there is no special scoring feature
   AddMusicSet "dropbank"
   Target3Bank2fpxLightControl()                          ' Reset lights
   Select Case v_fpxTarget3Bank2Case
    Case 0 :                              ' v_fpx CaseStart = 0. no light lens till next
case
    Case 1 :  v_fpxTarget3Bank2Lights(1,4).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn           ' First bank made, Turn on Jackpot light
    Case 2 :                              ' Second Bank made
    Case Else : v_fpxTarget3Bank2Case=(v_fpxTarget3Bank2CaseStart)
      ' error catcher
   End Select
   AddVaultScoreTarget3Bank2fpx()                                   ' Goto scoring
code as set by user. Must be before increase case code! (geez shiva)
   v_fpxTarget3Bank2Case = v_fpxTarget3Bank2Case+1                                   '
Increases value by 1 for select case routines
   IF v_fpxTarget3Bank2Case> v_fpxTarget3Bank2CaseEnd THEN
v_fpxTarget3Bank2Case=(v_fpxTarget3Bank2CaseStart)         ' Wraps back to
CaseStart if over max CaseEnd
   Target3Bank2fpxMemorySave()                                 ' Save Case Value
to that players memory
  END IF
 END IF
End Sub
' Timer to reset target bank after a delay.
Sub TimerTarget3Bank2fpx_Expired()
 TimerTarget3Bank2fpx.Enabled = False
 If v_fpxTarget3Bank2On=1 THEN
  IF VaultTarget3Bank2fpxDebug=1 THEN  AddDebugText
"TimerTarget3Bank2fpx_Expired()"              ' Dev Debug Code
  FOR x = 1 To (v_fpxTarget3Bank2LightCount):
v_fpxTarget3Bank2Lights(1,x).State = BulbOff:NEXT             ' We need to
restore the light memory for each player, so all lights off
  IF v_fpxTarget3Bank2Case=2 THEN
v_fpxTarget3Bank2Lights(1,4).State=BulbOn                  ' Jackpot Light blinks
rapidly for time set
  Target3Bank2fpxLightControl()                          ' Restore the target lights to
start again
 END IF
End sub
' runs the correct light routine and restores proper light if
v_fpxTarget3Bank2Memory=1
Sub CloseVaultTarget3Bank2fpx()
 If v_fpxTarget3Bank2On=1 THEN                                   ' Only if this
feature is set to "1" and nothing else
  IF VaultTarget3Bank2fpxDebug=1 THEN  AddDebugText
"CloseVaultTarget3Bank2fpx()"               ' Dev Debug Code
  Target3Bank2fpxLightControl()                          ' Restore the target lights to
start again
  Select Case v_fpxTarget3Bank2Case                                ' Sets next
light to display based on Case
   Case 0:
   Case 1:
   Case 2: v_fpxTarget3Bank2Lights(1,4).State = BulbOn                 ' Turn on
Jackpot Light
   Case Else : v_fpxTarget3Bank2Case=1                           ' error catcher
  END SELECT
  END IF
End Sub
' Main bank scoring routine
Sub Target3Bank2fpxLightControl()
 If v_fpxTarget3Bank2On=1 THEN
  IF VaultTarget3Bank2fpxDebug=1 THEN  AddDebugText
"Target3Bank2fpxLightControl()"              ' Dev Debug Code
                   ' Only if this feature is set to "1" and nothing else
  v_fpxTarget3Bank2BlinkIntervals=150                              ' Blink Intervals
  LightTarget3Bank2fpx1.BlinkInterval =
(v_fpxTarget3Bank2BlinkIntervals):LightTarget3Bank2fpx2.BlinkInterval =
(v_fpxTarget3Bank2BlinkIntervals)
  LightTarget3Bank2fpx3.BlinkInterval =
(v_fpxTarget3Bank2BlinkIntervals):LightTarget3Bank2fpx4.BlinkInterval =
(v_fpxTarget3Bank2BlinkIntervals)
  LightTarget3Bank2fpx1.BlinkPattern =
"100":LightTarget3Bank2fpx2.BlinkPattern =
"010":LightTarget3Bank2fpx3.BlinkPattern =
"001":LightTarget3Bank2fpx4.BlinkPattern = "010" ' Blink Pattern
  IF v_fpxTarget3Bank2UseWilliamsLights THEN
   FOR x = 1 To (v_fpxTarget3Bank2LightCount-1):
v_fpxTarget3Bank2Lights(1,x).State = BulbBlink:NEXT            ' (WMS) all lights
Blink
  ELSE
   FOR x = 1 To (v_fpxTarget3Bank2LightCount-1):
v_fpxTarget3Bank2Lights(1,x).State = BulbOff:NEXT             ' (Bally) Turns off
lights
  END IF
 END IF
End Sub
' Start Game Routine
Sub Target3Bank2fpxGameReset()
 IF v_fpxTarget3Bank2On=1 THEN                               ' Check if vault item is
being used
  IF VaultTarget3Bank2fpxDebug=1 THEN  AddDebugText
"Target3Bank2fpxGameReset()"                 ' Dev Debug Code
  FOR x = 1 To (v_fpxTarget3Bank2LightCount):
v_fpxTarget3Bank2Lights(1,x).State = BulbOff:NEXT              ' We turn off all the
Bulb lights first before we update the scoring
  v_fpxTarget3Bank2Case=(v_fpxTarget3Bank2CaseStart)                        '
Resets scoring case setting back to starting default (set in vault user options)
  m_fpxTarget3Bank2p1=(v_fpxTarget3Bank2CaseStart):m_fpxTarget3Bank2p2=
(v_fpxTarget3Bank2CaseStart)          ' Resets variables back to initial starting
point
  m_fpxTarget3Bank2p3=(v_fpxTarget3Bank2CaseStart):m_fpxTarget3Bank2p4=
(v_fpxTarget3Bank2CaseStart)
 END IF
End Sub

' Run from NewBall(). Also used as a blanket reset called from tilt, startup and
game over subroutines.
Sub Target3Bank2fpxBallReset()
 IF v_fpxTarget3Bank2On=1 THEN                               ' Check if vault item is
being used
  IF VaultTarget3Bank2fpxDebug=1 THEN  AddDebugText
"Target3Bank2fpxBallReset()"                 ' Dev Debug Code
  TimerTarget3Bank2fpx.Set True, 20                          ' Resets targets (after
set delay time) Note we do this here before the memory check
  If v_fpxTarget3Bank2Memory=1 Then                                       ' Look to see if
player memory feature is on if it is then...
   Target3Bank2fpxMemoryLoad()                                   ' Load in last
v_fpxTarget3Bank2CaseStart made from loss of previous ball
  ELSE                              ' or if player memory feature is off
   FOR x = 1 To (v_fpxTarget3Bank2LightCount):
v_fpxTarget3Bank2Lights(1,x).State = BulbOff:NEXT              ' We turn off all the
lights first before we update the scoring
   FOR x = 1 TO
```

```
v_fpxTarget3Bank2BulbCount:v_fpxTarget3Bank2Bulbs(1,x).State =
BulbOn:NEXT             ' We turn on all the Bulb lights first before we update the
scoring
   IF v_fpxTarget3Bank2CaseStart > 1 THEN v_fpxTarget3Bank2CaseStart = 1
                ' error catcher and prevents people from cheezing
   v_fpxTarget3Bank2Case=(v_fpxTarget3Bank2CaseStart)
  ' Reset v_fpxTarget3Bank2Case back to the beginning(user selectable top of
script)
   CloseVaultTarget3Bank2fpx()                              ' runs the correct light routine
and restores proper light if target bank player memory feature is set to 1
  END IF
 END IF
End Sub
' Only called from Target3Bank2fpxBallReset() but keep here in case future
vaults need this
Sub Target3Bank2fpxMemoryLoad()
 IF v_fpxTarget3Bank2On=1 THEN
  IF VaultTarget3Bank2fpxDebug=1 THEN  AddDebugText
"Target3Bank2fpxMemoryLoad()"                ' Dev Debug Code
  FOR x = 1 TO v_fpxTarget3Bank2BulbCount:v_fpxTarget3Bank2Bulbs(1,x).State
= BulbOn:NEXT             ' PF bulb lights should be on, but lets make sure
  Select Case CurrentPlayer                             ' Now we look at the memory for
the player that is up to see which light should be turned back on
   Case 1:v_fpxTarget3Bank2Case=m_fpxTarget3Bank2p1
   Case 2:v_fpxTarget3Bank2Case=m_fpxTarget3Bank2p2
   Case 3:v_fpxTarget3Bank2Case=m_fpxTarget3Bank2p3
   Case 4:v_fpxTarget3Bank2Case=m_fpxTarget3Bank2p4
  End Select
  IF v_fpxTarget3Bank2Case> v_fpxTarget3Bank2CaseEnd THEN
v_fpxTarget3Bank2Case=(v_fpxTarget3Bank2CaseStart)          ' Wraps back to
caseStart if over max CaseEnd
  Select Case v_fpxTarget3Bank2Case                              ' Restore the proper
light the extra snazzy way
   Case 0 : FOR x = 1 To
(v_fpxTarget3Bank2LightCount):v_fpxTarget3Bank2Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxTarget3Bank2BlinkIntervals/4),BulbOn:NEXT
   Case 1 : FOR x = 1 To
(v_fpxTarget3Bank2LightCount):v_fpxTarget3Bank2Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxTarget3Bank2BlinkIntervals/4),BulbOn:NEXT
   Case 2 : FOR x = 1 To
(v_fpxTarget3Bank2LightCount):v_fpxTarget3Bank2Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxTarget3Bank2BlinkIntervals/4),BulbOn:NEXT
   Case Else:FOR x = 1 To
(v_fpxTarget3Bank2LightCount):v_fpxTarget3Bank2Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxTarget3Bank2BlinkIntervals/4),BulbOn:NEXT
  End Select
  IF VaultTarget3Bank2fpxDebug=1 AND v_fpxTarget3Bank2On=1 THEN
              ' Dev Debug Code
   AddDebugText " - m_fpxTarget3Bank2p1 =   " &
(m_fpxTarget3Bank2p1):AddDebugText " - m_fpxTarget3Bank2p2 =   " &
(m_fpxTarget3Bank2p2):AddDebugText " - m_fpxTarget3Bank2p3 =   " &
(m_fpxTarget3Bank2p3):AddDebugText " - m_fpxTarget3Bank2p4 =   " &
```

```
(m_fpxTarget3Bank2p4)
  END IF
 END IF
End Sub
' Only called from AddVaultTarget3Bank2fpx() but keep here in case future
vaults need this
Sub Target3Bank2fpxMemorySave()
  IF v_fpxTarget3Bank2On=1 THEN                                 ' Check if vault item is
being used
  IF VaultTarget3Bank2fpxDebug=1 THEN  AddDebugText
"Target3Bank2fpxMemorySave()"                  ' Dev Debug Code
  Select Case CurrentPlayer                                  ' we see which player  is
playing.
   Case 1:m_fpxTarget3Bank2p1=v_fpxTarget3Bank2Case
   Case 2:m_fpxTarget3Bank2p2=v_fpxTarget3Bank2Case
   Case 3:m_fpxTarget3Bank2p3=v_fpxTarget3Bank2Case
   Case 4:m_fpxTarget3Bank2p4=v_fpxTarget3Bank2Case
  End Select
  END IF
End Sub
'  /END fpx Vault Target3Bank2
' ***********************************************************************
' **                                   Vault                                            **
' ***********************************************************************
' *** These subroutines are Master subroutines and are used by all Vault items
***
' Run at NewBall. Also used as a blanket reset used for tilt, startup or game
over. This is used by all Vault items and called directly in the main engine code
Sub VaultBallReset()
 ' Standup Targets
 IF v_fpxTarget3Bank2On=1 THEN Target3Bank2fpxBallReset() : END IF
           ' v_fpxTarget3Bank2On
End Sub
' Game Start. This resets the variables. This is used by all Vault items and
called directly in the main engine code
Sub VaultGameReset()
 ' StandupTargets
 IF v_fpxTarget3Bank2On=1 THEN Target3Bank2fpxGameReset():END IF
End Sub
' Closes ScoringEvent code pointed to by background timer for additional
instructions. This is used by all Vault items and called directly in the main
engine code
' TimerCloseScoringEventCase is the control variable
Sub TimerCloseScoringEvent_Expired()                              ' Restore the
scoring display
End Sub
' Saves the v_fpxAVCase settings for each player at the loss of a ball
(Selectable by user). This is used by all Vault items and called directly in the
main engine code
Sub VaultAVMemorySave()
 '  StandupTargets
 IF v_fpxTarget3Bank2On=1 THEN Target3Bank2fpxMemorySave()
End Sub
' Loads the v_fpx "Case" settings for each player at the start of a ball, and
restores that value back on his next ball if Memory=1.
' This is used by all Vault items and called directly in the main engine code
Sub VaultAVMemoryLoad()
 '  StandupTargets
 IF v_fpxTarget3Bank2On=1 THEN Target3Bank2fpxMemoryLoad():END IF
end sub
```

Vault - Triggers

vault_fpxAV1

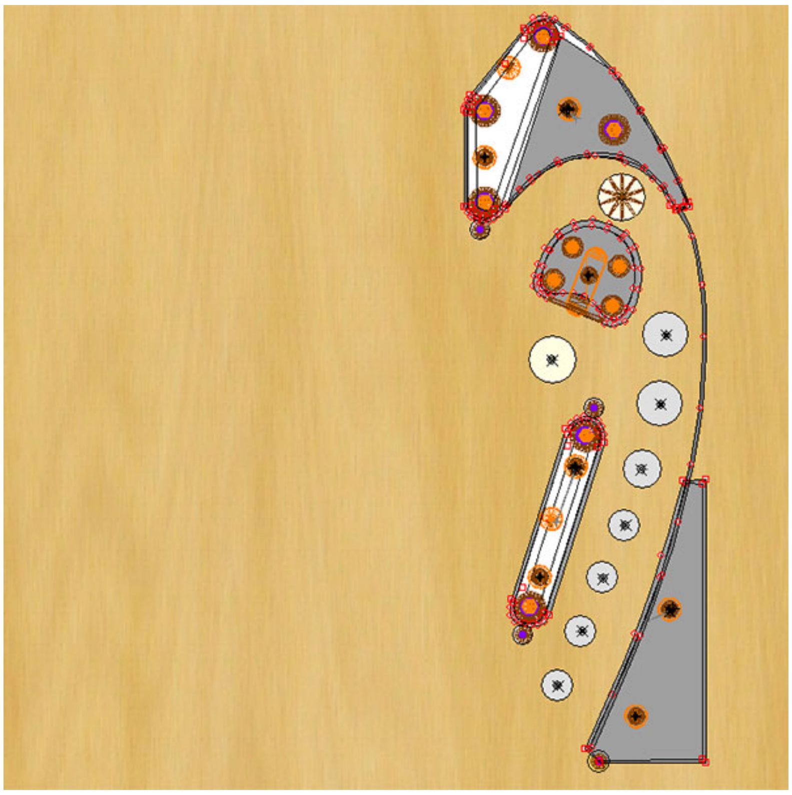
Vault - Triggers Targets - fpxTAV1

Vault - Triggers - fpxAV1

How to use this Vault Item in the fpxEngine

Using this Vault item and including it in your table design is very simple. It's just a couple copy and pastes from one fpt file to another!
Don't know How? Just click here.

vault_fpxTarget3Bank1



How it works

This Vault item has a Advancing Value system, which is set to increase a set value everytime the ball rolls over the top trigger. There are also code for the two rubbers, as well as a Target that when hit will light up that target for a bonus that is added to the bonus countdown at the loss of a ball.

User adjustments (pinsettings)

v_fpxAV1Memory= 1

- Handles the Memory feature for Target Score (common pin setting)
- Set to one if you want the player(s) Target Bank made total per game in play carried over to his next ball in play.
- Set to 0 to have the Target Bank Made total per game in play reset back to beginning with each new ball,

v_fpxAV1CaseStart = 1

- Initial light to be lit in this vault item for start of each ball
- Set to 0 for no lights at start or "hard". (You need to complete the routine once before the first light will be lit)
- Set to 1 to have the first light on at the

start of a game

How to Change Scoring (Bank)

Sub AddVaultScorefpxAV1()

This subroutine handles scoring when a bank is made. you can modify each stage by modifying the code with each case setting by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement fpxEngine AddScoringEvents routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.
This uses a Star trigger, which has a light attached to it. The first time you roll over the Star Trigger, that light will be lit, and score a single bonus added to the bonus countdown at the loss of a players ball.
- Case 0: 500 points,500 added to Jackpot value
- Case 1: 1000 points,1000 added to Jackpot value
- Case 2: 2000 points,2000 added to Jackpot value
- Case 3: 4000 points,4000 added to Jackpot value
- Case 4: 6000 points,6000 added to Jackpot value
- Case 5: 8000 points,8000 added to Jackpot value
- Case 6: 10000 points,10000 added to Jackpot value
- Case 7: Special
- Case 8: 10000 points,10000 added to Jackpot value

How to Change scoring (Target Objects)

Sub TargetAV1fpx1_Hit()

This subroutine handles scoring when the single Target is made. There are two stages, one for when the target is lit, and the second for when the Target is unlit. If you hit the target when it is unlit, the light for the target will lite up. The target is set to be automatically unlit when the trigger is rolled over, so the player will have to hit the target again to light it up. You can modify these 2 stages by modifying the code by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement fpxEngine AddScoringEvents routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.

```
c IF LightAV1fpx8.State=BulbOff THEN
o  AddScore(5000):AddJackpot(5000):AddMusicSet "trigger"
d ELSE
e  AddScore(1000):AddJackpot(1000):AddBonus (1):AddMusicSet
  "inlaneislit"
  END IF
```

How to Change scoring (Rubber Objects)

Sub RubberAV1fpx1_Hit(), Sub RubberAV1fpx3_Hit()

There are 2 Rubber objects which also score. These are set to the standard "sling" music sound, and will score 10 points when the ball strikes the rubber. RubberAV1fpx1 also contains code for the Alternating Lights scoring feature. If you do not wish to use this feature, just delete the light in the script.

```
c ' Rubber hits. You can use AddAlternatingLanes if you want.
o Sub RubberAV1fpx1_Hit()
d  IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
e  If v_fpxAV1On=1 THEN
    AddScore (10)
```

```
  AddScoringEvent "AddAlternatingLanes"          ' Alternate
Inlane/outlanes if enabled
  LightsRoutineAV1fpxGroup1()             ' routine to flash lights/bulbs
  AddMusicSet "sling"
 END IF
 set LastSwitchHit = TargetAV1fpx1
End Sub
' with this vault, everything is divided to 4 groups. There's no rubbers in
group 2
Sub RubberAV1fpx3_Hit()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 If v_fpxAV1On=1 THEN
  AddScore (10)
  LightsRoutineAV1fpxGroup3()             ' routine to flash lights/bulbs
  AddMusicSet "sling"
 END IF
 set LastSwitchHit = RubberAV1fpx3
End Sub
```

List of Objects

These objects are needed for the entire script to work. If you accidentally delete one of these objects and then run the table, the table will throw up a error message telling you a object is missing. If you restore that object back, your table will run fine.

This is a list of all the objects needed:

Lights

LightAV1fpx1
LightAV1fpx2
LightAV1fpx3
LightAV1fpx4
LightAV1fpx5
LightAV1fpx6
LightAV1fpx7
LightAV1fpx8 (used for the single target)
LightAV1fpx9 (used for the light for the trigger)

Bulbs and Plastics

Bulbs and plastics are divided into 4 groups. Group 1 is the top portion of the design. Group 2 is the single target area. Group 3 is the narrow rubber area to the left, and Group 4 is the bottom area at the right. This is used internally within the code, it's main advantage is this allows separate lighting effects for each group that can be run separately or in a combination with the other 3 groups. If you are modifing the design, you can "split" apart the design and move each group to separate locations, but you can not delete any of the objects.

* Bulbs Group 1
BulbAV1fpx1
BulbAV1fpx2
PlasticAV1fpx1

* Bulbs Group 2
BulbAV1fpx3

PlasticAV1fpx2

* Bulbs Group 3
BulbAV1fpx4
BulbAV1fpx5
PlasticAV1fpx3

* Bulbs Group 4
BulbAV1fpx6
BulbAV1fpx7
PlasticAV1fpx4

Objects

TriggerAV1fpx1
TargetAV1fpx1
RubberAV1fpx1
RubberAV1fpx3

Additional Information

- All Stand Up Targets objects are set to Layer 4 in your editor. Each major group of Vault items are set to their own layer for easy moving and modification.
- Layer 9 are reserved for the top (header) and Bottom (slingshots/flippers/aprons)
- Layer 0 is reserved for the fpxEngine objects.
- All objects can be modified or rotated to fit your design. Consult the FP manual for making changes directly in the editor, or visit the Pinball Nirvana website if you get really stuck.

A Note

fpxEngine uses a shaped light as a "plastic", as well as a semi-transparent surface as a edge around it. This is set to a height of 32mm, to form a proper cover over posts/rubbers etc that the ball can roll underneath. The Plastic light is part of the bulb code, it will flash in unison with the bulbs underneath, and can not be deleted without causing a error message when you run the table. Plastics always have the word in front of the file name. Semi-transparent edges (surfaces) always has the "t_ in front of the name, and are not part of the script, so you can safely delete these objects.

Vault Worksheet

The very nature of the fpxEngine is to allow absolute beginners to build and create complex tables with as little actual experience as possible, but you still need to learn a tiny bit of coding sometimes. Most of the Vault items will be based on the actual arcade table it came from, so it's just a quick copy and paste the design elements, the code, and then adding lights in the LightList manager.

More advanced coders though may wish to duplicate or modify this code to run different routines. This Vault item worksheet is the master template I use to create vault items from, so if you decide to duplicate or modify this vault item, you can change it easily using this worksheet. For more information, check the Using the Vault Worksheets page.

```
co  ' * Find and replace code keywords for duplication or new vault items
```

de
:

```
' replacement table vault name keyword: fpx
' replacement object keyword: AV1
' Remark Keyword: AV1
' Coding Names Keywords
' v_ variables keyword
' m_ memory keyword
' VaultAV1fpxDebug      ' Debug
' v_fpxAV1On       ' Turns on or off this vault item. (Prevents errors if no objects
on the table)
' v_fpxAV1CaseStart     ' Initial starting case(position) for Bank made
' v_fpxAV1Memory      ' Handles memory for each player
' v_fpxAV1Case       ' Bank scoring control case
' v_fpxAV1BlinkIntervals    ' Interval of flashing lights
' v_fpxAV1CaseEnd      ' Last case (position) for each bank made
' v_fpxAV1CaseRepeat
' v_fpxAV1Lights(9,9)     ' Use in FOR/NEXT loops. 9 groups with 9 objects in
each group max.
' v_fpxAV1Bulbs(9,9)     ' Use in FOR/NEXT loops. 9 groups with 9 objects in
each group max.
' v_fpxAV1LightCount     ' Total amount of lights used
' v_fpxAV1BulbGroup1Count    ' Group 1 bulb count
' v_fpxAV1BulbGroup2Count    ' Group 2 bulb count
' v_fpxAV1BulbGroup3Count    ' Group 3 bulb count
' v_fpxAV1BulbGroup4Count    ' Group 4 bulb count
' m_fpxAV1p1       ' Variable for Player 1 memory
' m_fpxAV1p2       ' Variable for Player 2 memory
' m_fpxAV1p3       ' Variable for Player 3 memory
' m_fpxAV1p4        ' Variable for Player 4 memory
' * Subroutines
' AddVaultScoreAV1fpx()     ' Main Bank Scoring (user adjustable)
' AV1fpx1_Hit()       ' Hit routine for object (user adjustable)
' AV1fpx2_Hit()       ' Hit routine for object (user adjustable)
' AV1fpx3_Hit()       ' Hit routine for object (user adjustable)
' AddVaultAV1fpx()      ' Main bank scoring routine (Vault Engine)
' CloseVaultAV1fpx()     ' Used in player memory to restore the proper lights
' AV1fpxLightControl()     ' Main lighting routine
' LightsRoutineAV1fpxGroup1    ' Group 1 bulb/plastics routines
' LightsRoutineAV1fpxGroup2    ' Group 2 bulb/plastics routines
' LightsRoutineAV1fpxGroup3    ' Group 3 bulb/plastics routines
' LightsRoutineAV1fpxGroup4    ' Group 4 bulb/plastics routines
' AV1fpxGameReset()      ' Start game routine
' AV1fpxBallReset()      ' New ball routine
' AV1fpxMemoryLoad()     ' Load player memory
' AV1fpxMemorySave()     ' Saves player memory
' * Unique subroutines to this vault
' TimerAV1fpx_Expired()     ' Timer to reset after set delay. (Not used)
' * Lights
' LightAV1fpx1
' LightAV1fpx2
' LightAV1fpx3
' LightAV1fpx4
' LightAV1fpx5
' LightAV1fpx6
' LightAV1fpx7
' LightAV1fpx8
' LightAV1fpx9
' * Bulbs Group 1
' BulbAV1fpx1
' BulbAV1fpx2
' PlasticAV1fpx1
' * Bulbs Group 2
' BulbAV1fpx3
' PlasticAV1fpx2
' * Bulbs Group 3
' BulbAV1fpx4
' BulbAV1fpx5
' PlasticAV1fpx3
' * Bulbs Group 4
' BulbAV1fpx6
' BulbAV1fpx7
' PlasticAV1fpx4
' * Objects
' TriggerAV1fpx1
' TargetAV1fpx1
' RubberAV1fpx1
' RubberAV1fpx3
' * fpx Vault AV1 *
' --------------------------
' For the fpxEngine.
' 1. Make sure all layers in the FP editor are "visible".
' Copy all the table elements in the editor from this fpt and then paste these
elements into your fpxEngine table.
' 2. Set User adjustments below to suit, then copy this entire script from this fpt
and paste into the script  for your fpx table.
'  I recommend  pasting  in the HIT SECTION.
' 3. Add Lights,plastics and bulbs to your LightList Manager in your fpx table.
' - Look for the names listed on the left side in the LightList Manager to have
"light", "bulb" or "plastic" as fpx will use these words
' at the beginning of the object name for all Vault Items
'  - If you do not know how to transfere the lights, both the fpx manual (in the
vault pages) and the Future Pinball manual explains how to do this.
' 4. Press "play"!
' * User adjustments  *
' Initial light to be lit in this vault item for start of each ball, set to 0 for no lights at
start or "hard".
' (You need to complete the routine once before the first light will be lit)
' Set to 1 to have the first light on at the start of a game
 v_fpxAV1CaseStart = 1                          ' Variable to hold DropTarget Bank
starting value
' Handles the Memory feature for Target Score (common pin setting)
' Set to one if you want the player(s) Target Bank made total per game in play
carried over to his next ball in play.
' Set to 0 to have the Target Bank Made total per game in play reset back to
beginning with each new ball,
' as set by v_fpxAV1CaseStart  above.
 v_fpxAV1Memory=1                         ' Variable to hold Target Bank Score
Value from ball to Ball (for each player)                         ' 0=Bally(no light on
before hit) 1=Williams (light is blinking before hit)
' *** Scoring ***
' Allows you to set the scoring. Everything else is done for you within the engine.
You can add/change anything you want
' like a multiplier or turning on Inlane lights etc. See Beginners Guide in manual
Sub AddVaultScorefpxAV1()
 If v_fpxAV1On=1 THEN
  Select Case v_fpxAV1Case
   Case 0
    AddScore(500):AddJackpot(500)
   Case 1
    AddScore(1000):AddJackpot(1000)
   Case 2
    AddScore(2000):AddJackpot(2000)
   Case 3
    AddScore(4000):AddJackpot(4000)
   Case 4
    AddScore(6000):AddJackpot(6000)
   Case 5
    AddScore(8000):AddJackpot(8000)
   Case 6
    AddScore(10000):AddJackpot(10000)
   Case 7
    AddScoringEvent "Special"                         ' Adds a credit (free game)
   Case 8
    AddScore(10000):AddJackpot(10000)
   End Select
 End If
End Sub
' Star Rollover trigger advances lit value
Sub TriggerAV1fpx1_Hit()
 PlaySound "trigger"
  IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 IF v_fpxAV1On=1 THEN AddVaultAV1fpx():END IF                    ' Only if this
feature is set to "1" and nothing else
 ' The star is switched off at every new ball, so we set it to on when ball rolls over,
and then it will score a bonus every time till loss of ball
 IF LightAV1fpx9.State=BulbOn THEN                    ' look to see if light is already
on
  AddBonus (1)                         ' Add Bonus to bonus count
 ELSE                         ' Light not on
  LightAV1fpx9.State=BulbOn                    ' so turn it on
 END IF
 set LastSwitchHit = TriggerAV1fpx1
```

```
End Sub

' Additional Target scores points, lits TriggerAV1fpx1 for bonus
Sub TargetAV1fpx1_Hit()
  IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
  If v_fpxAV1On=1 THEN                                   ' Only if this feature is set to "1" and
nothing else
  IF LightAV1fpx8.State=BulbOff THEN
   AddScore(5000):AddJackpot(5000):AddMusicSet "trigger"
  ELSE
   AddScore(1000):AddJackpot(1000):AddBonus (1):AddMusicSet "inlaneislit"
  END IF
  LightAV1fpx8.State=BulbOn
  END IF
  set LastSwitchHit = TargetAV1fpx1
End Sub
' Rubber hits. You can use AddAlternatingLanes if you want.
Sub RubberAV1fpx1_Hit()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 If v_fpxAV1On=1 THEN
  AddScore (10)
  AddScoringEvent "AddAlternatingLanes"          ' Alternate Inlane/outlanes if
enabled
  LightsRoutineAV1fpxGroup1()              ' routine to flash lights/bulbs
  AddMusicSet "sling"
 END IF
 set LastSwitchHit = TargetAV1fpx1
End Sub
' with this vault, everything is divided to 4 groups. There's no rubbers in group 2
Sub RubberAV1fpx3_Hit()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 If v_fpxAV1On=1 THEN
  AddScore (10)
  LightsRoutineAV1fpxGroup3()              ' routine to flash lights/bulbs
  AddMusicSet "sling"
 END IF
 set LastSwitchHit = RubberAV1fpx3
End Sub
                        ' KEEP THIS SET TO 1.
 ' Turns on or off this vault item.If set to 0 then this code is disabled.
 ' Set to 1 to use this vault item complete with code and objects
  v_fpxAV1On=1
'  /END fpxVaultAV1
' -----------------------------------------------------
' *** fpxAdvanceValue ***
' -----------------------------------------------------
VaultAV1fpxDebug=0
Dim
v_fpxAV1On
,v_fpxAV1CaseStart
,v_fpxAV1Memory
,v_fpxAV1Case
```

```
,v_fpxAV1Lights
(
9
,
9),v_fpxAV1LightCount
,v_fpxAV1CaseEnd,v_fpxAV1CaseRepeat,VaultAV1fpxDebug
Dim
v_fpxAV1Bulbs
(
9
,10),v_fpxAV1BulbGroup1Count
,v_fpxAV1BulbGroup2Count
,v_fpxAV1BulbGroup3Count,v_fpxAV1BulbGroup4Count
Dim m_fpxAV1p1,m_fpxAV1p2,m_fpxAV1p3,m_fpxAV1p4
If v_fpxAV1On=1 THEN
 Set v_fpxAV1Lights(1,1)=LightAV1fpx1:Set v_fpxAV1Lights(1,2)
=LightAV1fpx2:Set v_fpxAV1Lights(1,3)=LightAV1fpx3:Set v_fpxAV1Lights(1,4)
=LightAV1fpx4
 Set v_fpxAV1Lights(1,5)=LightAV1fpx5:Set v_fpxAV1Lights(1,6)
=LightAV1fpx6:Set v_fpxAV1Lights(1,7)=LightAV1fpx7:Set v_fpxAV1Lights(1,8)
=LightAV1fpx8:Set v_fpxAV1Lights(1,9)=LightAV1fpx9
 Set v_fpxAV1Bulbs(1,1)=BulbAV1fpx1:Set v_fpxAV1Bulbs(1,2)
=BulbAV1fpx2::Set v_fpxAV1Bulbs(1,3)=PlasticAV1fpx1
 Set v_fpxAV1Bulbs(2,1)=BulbAV1fpx3:Set v_fpxAV1Bulbs(2,2)=PlasticAV1fpx2
 Set v_fpxAV1Bulbs(3,1)=BulbAV1fpx4:Set v_fpxAV1Bulbs(3,2)
=BulbAV1fpx5:Set v_fpxAV1Bulbs(3,3)=PlasticAV1fpx3
 Set v_fpxAV1Bulbs(4,1)=BulbAV1fpx6:Set v_fpxAV1Bulbs(4,2)
=BulbAV1fpx7:Set v_fpxAV1Bulbs(4,3)=PlasticAV1fpx4
End If
' BAM bulb
 BulbAV1fpx1EXT
.Brightness=(fpxBulbBrightness):BulbAV1fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbAV1fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbAV1fpx2EXT
.Brightness=(fpxBulbBrightness):BulbAV1fpx2EXT
.GlowRadius=(fpxBulbGlowRadius):BulbAV1fpx2EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbAV1fpx3EXT
.Brightness=(fpxBulbBrightness):BulbAV1fpx3EXT
.GlowRadius=(fpxBulbGlowRadius):BulbAV1fpx3EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbAV1fpx4EXT
.Brightness=(fpxBulbBrightness):BulbAV1fpx4EXT
.GlowRadius=(fpxBulbGlowRadius):BulbAV1fpx4EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbAV1fpx5EXT
.Brightness=(fpxBulbBrightness):BulbAV1fpx5EXT
.GlowRadius=(fpxBulbGlowRadius):BulbAV1fpx5EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbAV1fpx6EXT
.Brightness=(fpxBulbBrightness):BulbAV1fpx6EXT
.GlowRadius=(fpxBulbGlowRadius):BulbAV1fpx6EXT
```

```
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbAV1fpx7EXT
.Brightness=(fpxBulbBrightness):BulbAV1fpx7EXT
.GlowRadius=(fpxBulbGlowRadius):BulbAV1fpx7EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticAV1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticAV1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticAV1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
 PlasticAV1fpx2EXT
.Brightness=(fpxPlasticBrightness):PlasticAV1fpx2EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticAV1fpx2EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
 PlasticAV1fpx3EXT
.Brightness=(fpxPlasticBrightness):PlasticAV1fpx3EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticAV1fpx3EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
 PlasticAV1fpx4EXT
.Brightness=(fpxPlasticBrightness):PlasticAV1fpx4EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticAV1fpx4EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
' light lens
 LightAV1fpx1EXT
.Brightness=(fpxLensBrightness):LightAV1fpx1EXT
.GlowRadius=(fpxLensGlowRadius):LightAV1fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightAV1fpx2EXT
.Brightness=(fpxLensBrightness):LightAV1fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightAV1fpx2EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightAV1fpx3EXT
.Brightness=(fpxLensBrightness):LightAV1fpx3EXT
.GlowRadius=(fpxLensGlowRadius):LightAV1fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightAV1fpx4EXT
.Brightness=(fpxLensBrightness):LightAV1fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightAV1fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightAV1fpx5EXT
.Brightness=(fpxLensBrightness):LightAV1fpx5EXT
.GlowRadius=(fpxLensGlowRadius):LightAV1fpx5EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightAV1fpx6EXT
.Brightness=(fpxLensBrightness):LightAV1fpx6EXT
.GlowRadius=(fpxLensGlowRadius):LightAV1fpx6EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightAV1fpx7EXT
.Brightness=(fpxLensBrightness):LightAV1fpx7EXT
.GlowRadius=(fpxLensGlowRadius):LightAV1fpx7EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightAV1fpx8EXT
.Brightness=(fpxLensBrightness):LightAV1fpx8EXT
.GlowRadius=(fpxLensGlowRadius):LightAV1fpx8EXT
```

```
.GlowBrightness=(fpxLensGlowBrightness)
 LightAV1fpx9EXT
.Brightness=(fpxLensBrightness):LightAV1fpx9EXT
.GlowRadius=(fpxLensGlowRadius):LightAV1fpx9EXT
.GlowBrightness=(fpxLensGlowBrightness)
 v_fpxAV1LightCount = 9                                ' Number of Lens Lights used (in
set code) in FOR NEXT loops
 v_fpxAV1BulbGroup1Count = 3                           ' Number of bulbs used (in
set code) in FOR NEXT loops
 v_fpxAV1BulbGroup2Count = 2
 v_fpxAV1BulbGroup3Count = 3
 v_fpxAV1BulbGroup4Count = 3
 v_fpxAV1CaseEnd=8
 v_fpxAV1CaseRepeat=8
' Main scoring routine. On target bank, v_fpxAV1Case increases by one, then
runs that number in the matching case
Sub AddVaultAV1fpx()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
      ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
 If v_fpxAV1On=1 THEN                                  ' Check if vault item is
being used
FOR x = 1 To (v_fpxAV1LightCount):v_fpxAV1Lights(1,x).State=BulbOff:NEXT
                 ' All lens lights off first
   TimerCloseScoringEventCase = 1:LockDisplay=0                          '
Points to the closing routines after the timers are done
   AddMusicSet "guidetrigger"                          ' Music set. Needed for timer
intervals. Overwritten by AddScoringEvent routines
   LightsRoutineAV1fpxGroup1():LightsRoutineAV1fpxGroup2():LightsRoutineAV1
fpxGroup3():LightsRoutineAV1fpxGroup4()         ' Flash bulbs. This uses the
timer interval in AddMusicScore and resets to on afterwards
   'AddVaultScorefpxAV1()                                ' Goto scoring code as
set by developer
   Select Case v_fpxAV1Case
    Case 0                                  ' Start. No lens assigned
    Case 1                                ' 1st light lens
     DisplayBlinkInterval=(FlashForMSBlinkInterval
+40):v_fpxAV1Lights(1,1).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
    Case 2
     DisplayBlinkInterval=(FlashForMSBlinkInterval
+30):v_fpxAV1Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
    Case 3
     DisplayBlinkInterval=(FlashForMSBlinkInterval
+20):v_fpxAV1Lights(1,3).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
    Case 4
     DisplayBlinkInterval=(FlashForMSBlinkInterval
+10):v_fpxAV1Lights(1,4).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
```

```
    Case 5
     DisplayBlinkInterval=(FlashForMSBlinkInterval):v_fpxAV1Lights
(1,5).FlashForMs (MusicIntervalTime), (DisplayBlinkInterval),BulbOn
    Case 6
     DisplayBlinkInterval=(FlashForMSBlinkInterval):v_fpxAV1Lights
(1,6).FlashForMs (MusicIntervalTime), (DisplayBlinkInterval),BulbOn
    Case 7
     DisplayBlinkInterval=(FlashForMSBlinkInterval):v_fpxAV1Lights
(1,7).FlashForMs (MusicIntervalTime), (DisplayBlinkInterval),BulbOn
    Case 8
     DisplayBlinkInterval=(FlashForMSBlinkInterval):v_fpxAV1Lights
(1,6).FlashForMs (MusicIntervalTime), (DisplayBlinkInterval),BulbOn
   End Select
   AddVaultScoreAV1fpx()                              ' Goto scoring code as set
by user. Must be before increase case code! (geez shiva)
   v_fpxAV1Case = v_fpxAV1Case+1                              ' Increases value by
1 for select case routines
   IF v_fpxAV1Case> v_fpxAV1CaseEnd THEN
v_fpxAV1Case=(v_fpxAV1CaseStart)          ' Wraps back to CaseStart if over
max  CaseEnd
   AV1fpxMemorySave()                                ' Save Case Value to that
players memory
 END IF
End Sub
' runs the correct light routine and restores proper light if v_fpxAV1Memory=1
Sub CloseVaultAdvanceValue()
 AddDebugText "CloseVaultAdvanceValue() "
  If v_fpxAV1On=1 THEN                           ' Only if this feature is set to "1" and
nothing else
   FOR x = 1 To (v_fpxAV1LightCount-1): v_fpxAV1Lights(1,x).State =
BulbOff:NEXT
   BulbRoutineAV1fpxOn()
   Select Case v_fpxAV1Case                        ' Sets next light to display
    Case 0: Exit Sub
    Case 1:v_fpxAV1Lights(1,1).State = BulbOn:v_fpxAV1Lights(1,1).FlashForMs
(MusicIntervalTime/2), (FlashForMSBlinkInterval),BulbOn  ' 1k, light 10k light for
next case
    Case 2:v_fpxAV1Lights(1,2).State = BulbOn:v_fpxAV1Lights(1,2).FlashForMs
(MusicIntervalTime/2), (FlashForMSBlinkInterval),BulbOn  ' 10
    Case 3:v_fpxAV1Lights(1,3).State = BulbOn:v_fpxAV1Lights(1,3).FlashForMs
(MusicIntervalTime/2), (FlashForMSBlinkInterval),BulbOn  ' 20
    Case 4:v_fpxAV1Lights(1,4).State = BulbOn:v_fpxAV1Lights(1,4).FlashForMs
(MusicIntervalTime/2), (FlashForMSBlinkInterval),BulbOn  ' 30
    Case 5:v_fpxAV1Lights(1,5).State = BulbOn:v_fpxAV1Lights(1,5).FlashForMs
(MusicIntervalTime/2), (FlashForMSBlinkInterval),BulbOn  ' 40
    Case 6:v_fpxAV1Lights(1,6).State = BulbOn:v_fpxAV1Lights(1,6).FlashForMs
(MusicIntervalTime/2), (FlashForMSBlinkInterval),BulbOn  ' 50
    Case 7:v_fpxAV1Lights(1,7).State = BulbOn:v_fpxAV1Lights(1,7).FlashForMs
(MusicIntervalTime/2), (FlashForMSBlinkInterval),BulbOn  ' E
   END SELECT
  END IF
```

```
End Sub
' Main bank scoring routine
Sub AV1fpxLightControl()
 If v_fpxAV1On=1 THEN
' disabled. Not used
 END IF
End Sub
' Light/Bulb/Plastic routines for a rubber hit
' Group1 top with Star Rollover light
Sub LightsRoutineAV1fpxGroup1()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 If v_fpxAV1On=1 THEN
  FOR x = 1 To (v_fpxAV1BulbGroup1Count):v_fpxAV1Bulbs(1,x).FlashForMs
(MusicIntervalTime), (FlashForMSBlinkInterval), BulbOn:NEXT
  LightAV1fpx9.FlashForMs (MusicIntervalTime), (FlashForMSBlinkInterval),
BulbOn
 END IF
End Sub
' Group2 with target light
Sub LightsRoutineAV1fpxGroup2()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 If v_fpxAV1On=1 THEN
  FOR x = 1 To (v_fpxAV1BulbGroup2Count):v_fpxAV1Bulbs(2,x).FlashForMs
(MusicIntervalTime), (FlashForMSBlinkInterval), BulbOn:NEXT
  LightAV1fpx9.FlashForMs (MusicIntervalTime), (FlashForMSBlinkInterval),
BulbOn
 END IF
End Sub
' Group3 side
Sub LightsRoutineAV1fpxGroup3()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 If v_fpxAV1On=1 THEN
  FOR x = 1 To (v_fpxAV1BulbGroup3Count):v_fpxAV1Bulbs(3,x).FlashForMs
(MusicIntervalTime), (FlashForMSBlinkInterval), BulbOn:NEXT
 END IF
End Sub
' Group4 bottom
Sub LightsRoutineAV1fpxGroup4()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
 If v_fpxAV1On=1 THEN
  FOR x = 1 To (v_fpxAV1BulbGroup4Count):v_fpxAV1Bulbs(4,x).FlashForMs
(MusicIntervalTime), (FlashForMSBlinkInterval), BulbOn:NEXT
 END IF
End Sub
Sub BulbRoutineAV1fpxOn()
 FOR x = 1 To
(v_fpxAV1BulbGroup1Count):v_fpxAV1Bulbs(1,x).State=BulbOn:NEXT
 FOR x = 1 To
(v_fpxAV1BulbGroup2Count):v_fpxAV1Bulbs(2,x).State=BulbOn:NEXT
 FOR x = 1 To
(v_fpxAV1BulbGroup3Count):v_fpxAV1Bulbs(3,x).State=BulbOn:NEXT
 FOR x = 1 To
```

```
(v_fpxAV1BulbGroup4Count):v_fpxAV1Bulbs(4,x).State=BulbOn:NEXT
End Sub
' Start Game Routine
Sub AV1fpxGameReset()
 IF v_fpxAV1On=1 THEN                          ' Check if vault item is being used
  IF VaultAV1fpxDebug=1 THEN  AddDebugText "AV1fpxGameReset()"
  ' Dev Debug Code
  FOR x = 1 To (v_fpxAV1LightCount): v_fpxAV1Lights(1,x).State = BulbOff:NEXT
            ' We turn off all the Bulb lights first before we update the scoring
  v_fpxAV1Case=(v_fpxAV1CaseStart)                        ' Resets scoring case
setting back to starting default (set in vault user options)
  m_fpxAV1p1=(v_fpxAV1CaseStart):m_fpxAV1p2=(v_fpxAV1CaseStart)            '
Resets variables back to initial starting point
  m_fpxAV1p3=(v_fpxAV1CaseStart):m_fpxAV1p4=(v_fpxAV1CaseStart)
 END IF
End Sub

' Run from NewBall(). Also used as a blanket reset called from tilt, startup and
game over subroutines.
Sub AV1fpxBallReset()
 IF v_fpxAV1On=1 THEN                          ' Check if vault item is being used
  IF VaultAV1fpxDebug=1 THEN  AddDebugText "AV1fpxBallReset()"
' Dev Debug Code
  If v_fpxAV1Memory=1 Then                                ' Look to see if player
memory feature is on if it is then...
   AV1fpxMemoryLoad()                                  ' Load in last
v_fpxAV1CaseStart made from loss of previous ball
  ELSE                              ' or if player memory feature is off
   FOR x = 1 To (v_fpxAV1LightCount): v_fpxAV1Lights(1,x).State = BulbOff:NEXT
             ' We turn off all the lights first before we update the scoring
   BulbRoutineAV1fpxOn()           ' We turn on all the Bulb lights first before we
update the scoring
   IF v_fpxAV1CaseStart > 1 THEN v_fpxAV1CaseStart = 1                ' error
catcher and prevents people from cheezing
   v_fpxAV1Case=(v_fpxAV1CaseStart)                              ' Reset
v_fpxAV1Case back to the beginning(user selectable top of script)
   CloseVaultAV1fpx()                         ' runs the correct light routine and
restores proper light if target bank player memory feature is set to 1
  END IF
 END IF
End Sub
' Only called from AV1fpxBallReset() but keep here in case future vaults need
this
Sub AV1fpxMemoryLoad()
 IF v_fpxAV1On=1 THEN
  IF VaultAV1fpxDebug=1 THEN  AddDebugText "AV1fpxMemoryLoad()"
    ' Dev Debug Code
  BulbRoutineAV1fpxOn()            ' PF bulb lights should be on, but lets make
sure
  Select Case CurrentPlayer                          ' Now we look at the memory for
the player that is up to see which light should be turned back on
```

```
   Case 1:v_fpxAV1Case=m_fpxAV1p1
   Case 2:v_fpxAV1Case=m_fpxAV1p2
   Case 3:v_fpxAV1Case=m_fpxAV1p3
   Case 4:v_fpxAV1Case=m_fpxAV1p4
  End Select
  IF v_fpxAV1Case> v_fpxAV1CaseEnd THEN
v_fpxAV1Case=(v_fpxAV1CaseStart)          ' Wraps back to caseStart if over
max  CaseEnd
  Select Case v_fpxAV1Case                          ' Restore the proper light the
extra snazzy way
   Case 0 : FOR x = 1 To (v_fpxAV1LightCount):v_fpxAV1Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxAV1BlinkIntervals/4),BulbOn:NEXT
   Case 1 : FOR x = 1 To (v_fpxAV1LightCount):v_fpxAV1Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxAV1BlinkIntervals/4),BulbOn:NEXT
   Case 2 : FOR x = 1 To (v_fpxAV1LightCount):v_fpxAV1Lights(1,x).FlashForMs
(MusicIntervalTime/2), (v_fpxAV1BlinkIntervals/4),BulbOn:NEXT
   Case Else:FOR x = 1 To
(v_fpxAV1LightCount):v_fpxAV1Lights(1,x).FlashForMs (MusicIntervalTime/2),
(v_fpxAV1BlinkIntervals/4),BulbOn:NEXT
  End Select
  IF VaultAV1fpxDebug=1 AND v_fpxAV1On=1 THEN                        ' Dev
Debug Code
   AddDebugText " - m_fpxAV1p1 =   " & (m_fpxAV1p1):AddDebugText " -
m_fpxAV1p2 =   " & (m_fpxAV1p2):AddDebugText " - m_fpxAV1p3 =   " &
(m_fpxAV1p3):AddDebugText " - m_fpxAV1p4 =   " & (m_fpxAV1p4)
  END IF
 END IF
End Sub
' Only called from AddVaultAV1fpx() but keep here in case future vaults need
this
Sub AV1fpxMemorySave()
  IF v_fpxAV1On=1 THEN                              ' Check if vault item is being used
  IF VaultAV1fpxDebug=1 THEN  AddDebugText "AV1fpxMemorySave()"
    ' Dev Debug Code
  Select Case CurrentPlayer                         ' we see which player  is
playing.
   Case 1:m_fpxAV1p1=v_fpxAV1Case
   Case 2:m_fpxAV1p2=v_fpxAV1Case
   Case 3:m_fpxAV1p3=v_fpxAV1Case
   Case 4:m_fpxAV1p4=v_fpxAV1Case
  End Select
  END IF
End Sub
'  /END fpx Vault Target(3)Bank1
' ***********************************************************************
' **                            Vault                                   **
' ***********************************************************************
' *** These subroutines are Master subroutines and are used by all Vault items
***
' Run at NewBall. Also used as a blanket reset used for tilt, startup or game
over. This is used by all Vault items and called directly in the main engine code
```

```
Sub VaultBallReset()
 ' Standup Targets
 IF v_fpxAV1On=1 THEN AV1fpxBallReset() : END IF                    '
v_fpxAV1On
End Sub
' Game Start. This resets the variables. This is used by all Vault items and
called directly in the main engine code
Sub VaultGameReset()
 ' StandupTargets
 IF v_fpxAV1On=1 THEN AV1fpxGameReset():END IF
End Sub
' Closes ScoringEvent code pointed to by background timer for additional
instructions. This is used by all Vault items and called directly in the main
engine code
' TimerCloseScoringEventCase is the control variable
Sub TimerCloseScoringEvent_Expired()                          ' Restore the
scoring display
End Sub
' Saves the v_fpxAVCase settings for each player at the loss of a ball
(Selectable by user). This is used by all Vault items and called directly in the
main engine code
Sub VaultAVMemorySave()
 '  StandupTargets
 IF v_fpxAV1On=1 THEN AV1fpxMemorySave()
End Sub
' Loads the v_fpx "Case" settings for each player at the start of a ball, and
restores that value back on his next ball if Memory=1.
' This is used by all Vault items and called directly in the main engine code
Sub VaultAVMemoryLoad()
 '  StandupTargets
 IF v_fpxAV1On=1 THEN AV1fpxMemoryLoad():END IF
end sub
```

Vault - Kickers

vault_fpxKicker1

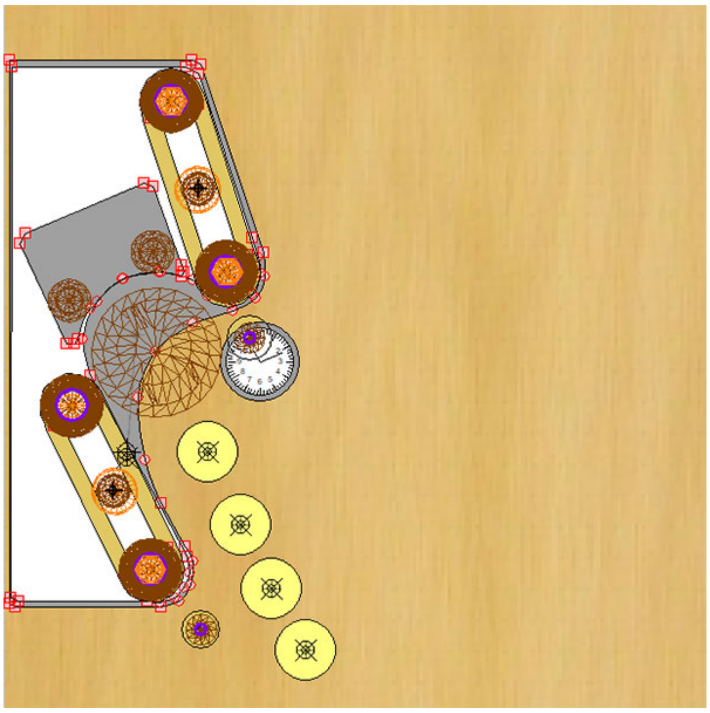
Vault - Triggers Targets - fpxTAV1

Vault - Kickers - fpxKicker1

How to use this Vault Item in the fpxEngine

Using this Vault item and including it in your table design is very simple. It's just a couple copy and pastes from one fpt file to another!
Don't know How? Just click here.

vault_fpxTarget3Bank1



How it works

This Vault item has a Advancing Value system, which is set to increase a set value every time the ball rolls over the top trigger. There are also code for the two rubbers, as well as a Target that when hit will light up that target for a bonus that is added to the bonus countdown at the loss of a ball.

User adjustments (pinsettings)

v_fpxKicker1Memory=1
- Handles the Memory feature for the Kicker scoring (common pin setting)
- Set to one if you want the player(s) made total per game in play carried over to his next ball in play.
- Set to 0 to have the Kicker total per game in play reset back to beginning with each new ball,

v_fpxKicker1CaseStart = 1
- Initial light to be lit in this vault item for start of each ball
- Set to 0 for no lights at start or "hard". (You need to complete the routine

once before the first light will be lit)
- Set to 1 to have the first light on at the start of a game

How to Change Scoring (Bank)

Sub AddVaultScorefpxAV1()

This subroutine handles scoring when the Kicker is made. you can modify each stage by modifying the code with each case setting by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement fpxEngine AddScoringEvents routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.
- Case 0: 5000 points,1000 added to Jackpot value
- Case 1: 10000 points,10000 added to Jackpot value
- Case 2: 25000 points
- Case 3: Extra Ball
- Case 4: Special
- Case 5: 25,000 points

How to Change scoring (Target Objects)

Sub TargetAV1fpx1_Hit()

This subroutine handles scoring when the Kicker is made. There are 5 stages, with the final stage (case 5 in this example) repeating till the player loses the ball. The scoring will then reset back to the beginning if v_fpxKicker1Memory is set to "0". If v_fpxKicker1Memory is set to "1", the player will be able to continue where he last left off, but note that this does not allow him to repeat the previous stages.
 You can modify these stages by modifying the code by adding different values between the brackets for higher scores, or by adding or replacing the code for additional or replacement fpxEngine AddScoringEvents routines. In most cases with AddScoringEvent, these are just 1 or 2 lines of code you can copy and paste.

code

```
' Kicker . You need to set a scoring value here for a single drop target hit
that doesn't complete a bank
Sub Kicker1fpx1_Hit()
 'IF VaultDebug=1 THEN AddDebugText "Kicker1_Hit() "
   ' Play the mechanical sound. this will sound even if game is tilted
 Playsound "kicker"
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN
TimerKicker1fpx1.Set True, 200:Exit Sub : END IF              ' See if in tilt
state or no game in progress to kick ball back out, exit the subroutine
and stop any scoring                                   ' Adds one bonus within the
brackets to the End-Of-Ball Bonus Countdown routine
 If v_fpxKicker1On=1 THEN AddVaultKicker1fpx() : END IF
     ' if all targets in this bank are down, go to bank made scoring
subroutine
 set LastSwitchHit = Kicker1fpx1                              ' FP code we can
use if needed. Just good programing
End Sub
```

List of Objects

These objects are needed for the entire script to work. If you accidentally delete one of

these objects and then run the table, the table will throw up a error message telling you a object is missing. If you restore that object back, your table will run fine.
This is a list of all the objects needed:

Lights

LightKicker1fpx1
LightKicker1fpx2
LightKicker1fpx3
LightKicker1fpx4

Bulbs and Plastics

BulbKicker1fpx1
BulbKicker1fpx2
PlasticKicker1fpx1

Objects

Kicker1fpx1
TimerKicker1fpx1
t_fpxDropTarget3Bank1

Additional Information

- All Kicker vault items objects are set to Layer 3 in your editor. Each major group of Vault items are set to their own layer for easy moving and modification.
- Layer 9 are reserved for the top (header) and Bottom (slingshots/flippers/aprons)
- Layer 0 is reserved for the fpxEngine objects.
- All objects can be modified or rotated to fit your design. Consult the FP manual for making changes directly in the editor, or visit the Pinball Nirvana website if you get really stuck.

A Note

fpxEngine uses a shaped light as a "plastic", as well as a semi-transparent surface as a edge around it. This is set to a height of 32mm, to form a proper cover over posts/rubbers etc that the ball can roll underneath. The Plastic light is part of the bulb code, it will flash in unison with the bulbs underneath, and can not be deleted without causing a error message when you run the table. Plastics always have the word in front of the file name. Semi-transparent edges (surfaces) always has the "t_ in front of the name, and are not part of the script, so you can safely delete these objects.

Vault Worksheet

The very nature of the fpxEngine is to allow absolute beginners to build and create complex tables with as little actual experience as possible, but you still need to learn a tiny bit of coding sometimes. Most of the Vault items will be based on the actual arcade table it came from, so it's just a quick copy and paste the design elements, the code, and then adding lights in the LightList manager.

More advanced coders though may wish to duplicate or modify this code to run different routines. This Vault item worksheet is the master template I use to create vault items from, so if you decide to duplicate or modify this vault item, you can change it easily using this worksheet. For more information, check the Using the Vault Worksheets page.

code:

```
' * Find and replace code keywords for duplication or new vault items
' replacement table vault name keyword: fpx
' replacement object keyword: Kicker1
' Remark Keyword: DropTargetBank1
' Coding Names Keywords
' v_ variables keyword
' m_ memory keyword
' * List of variables
' VaultKicker1fpxDebug
' v_fpxKicker1On
' v_fpxKicker1CaseStart
' v_fpxKicker1CaseEnd
' v_fpxKicker1Memory
' v_fpxKicker1Case
' v_fpxKicker1Lights(9,9)     ' For/Next Loops
' v_fpxKicker1Bulbs(9,9)      ' For/Next Loops
' v_fpxKicker1LightCount      ' Amount Lights in v_fpxKicker1Lights
' v_fpxKicker1BulbCount       ' Amount Bulbs in v_fpxKicker1Bulbs
' m_fpxKicker1p1
' m_fpxKicker1p2
' m_fpxKicker1p3
' m_fpxKicker1p4
' * Subroutines
' AddVaultKicker1fpx()       ' Main vault routine
' CloseVaultKicker1fpx()     ' Light control
' Kicker1fpxGameReset()      ' New game
' Kicker1fpxBallReset()      ' New ball in play
' Kicker1fpxMemorySave()     ' saves players progress
' Kicker1fpxMemoryLoad()     ' Restores players progress
' AddVaultScoreKicker1fpx
' fpxKicker1

' * Unique subroutines to this vault
' TimerKicker1fpx1_Expired() ' Timer
' * Lights
' LightKicker1fpx1
' LightKicker1fpx2
' LightKicker1fpx3
' LightKicker1fpx4
' * Bulbs
' BulbKicker1fpx1
' BulbKicker1fpx2
' PlasticKicker1fpx1
' * Objects
' Kicker1fpx1
' TimerKicker1fpx1
' t_fpxDropTarget3Bank1
' -----------------------------------------------------
' * fpx Vault Kicker1fpx *
' -----------------------------------------------------
' For the fpxEngine.
' 1. Make sure all layers in the FP editor are "visible".
' Copy all the table elements in the editor from this fpt and then paste these
elements into your fpxEngine table.
' 2. Set User adjustments below to suit, then copy this entire script from this fpt
and paste into the script  for your fpx table.
'  I recommend  pasting  in the HIT SECTION.
' 3. Add Lights,plastics and bulbs to your LightList Manager in your fpx table.
' - Look for the names listed on the left side in the LightList Manager to have
"light", "bulb" or "plastic" as fpx will use these words
' at the beginning of the object name for all Vault Items
'  - If you do not know how to transfere the lights, both the fpx manual (in the
vault pages) and the Future Pinball manual explains how to do this.
' 4. Press "play"!
' * User adjustments  *
' Initial light to be lit in this vault item for start of each ball, set to 0 for no lights at
start or "hard".
' (You need to complete the routine once before the first light will be lit)
' Set to 1 to have the first light on at the start of a game
 v_fpxKicker1CaseStart = 1                              ' Variable to hold DropTarget
Bank starting value
' Handles the Memory feature for DropTarget Score (common pin setting)
' Set to one if you want the player(s) DropTarget Bank made total per game in
play carried over to his next ball in play.
' Set to 0 to have the DropTarget Bank Made total per game in play reset back
to beginning with each new ball,
' as set by v_fpxKicker1CaseStart  above.
 v_fpxKicker1Memory=1                                   ' Variable to hold DropTarget
Bank Score Value from ball to Ball (for each player)
' *** Scoring ***
' Allows you to set the scoring. Everything else is done for you within the engine.
You can add/change anything you want in here
' like adding a multiplier or turning on Inlane lights etc. See Beginners Guide in
manual
Sub AddVaultScoreKicker1fpx1()
 'IF VaultDebug=1 THEN AddDebugText "AddVaultScoreKicker1fpx1() "
           '  Dev Debug Code
 If v_fpxKicker1On=1 THEN
  Select Case v_fpxKicker1Case
   Case 0                              ' This is used as the starting TargetBank score
if v_fpxKicker1CaseStart = 0.
  AddScore(5000)                          '  Adds the value within the brackets to
your player(s) score
  AddJackpot(1000)                          '  Adds the value within the brackets to
the Jackpot value which can be collected later on in the game
  AddBonus(1)                             '  Adds one bonus within the brackets to the
End-Of-Ball Bonus Countdown routine
   Case 1                              ' This is used as the starting TargetBank score
if v_fpxKicker1CaseStart = 1.
  AddScore(10000)
  AddJackpot(10000)
  AddBonus(1)
   Case 2
  AddScoringEvent "25kAward"
   Case 3
  AddScoringEvent "ExtraBall"                          ' Adds a Extra Ball
   Case 4
  AddScoringEvent "Special"                          ' Adds a credit (free game)
   Case Else                          ' Case else will score if case is higher than 5,
and will keep repeating
  AddScoringEvent "25kAward"
  End Select
 End If
End Sub
' Kicker . You need to set a scoring value here for a single drop target hit that
doesn't complete a bank
Sub Kicker1fpx1_Hit()
 'IF VaultDebug=1 THEN AddDebugText "Kicker1_Hit() "                         ' Play
the mechanical sound. this will sound even if game is tilted
 Playsound "kicker"
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN TimerKicker1fpx1.Set True,
200:Exit Sub : END IF               ' See if in tilt state or no game in progress to
kick ball back out, exit the subroutine and stop any scoring
   ' Adds one bonus within the brackets to the End-Of-Ball Bonus Countdown
routine
 If v_fpxKicker1On=1 THEN AddVaultKicker1fpx() : END IF                         ' if
all targets in this bank are down, go to bank made scoring subroutine
 set LastSwitchHit = Kicker1fpx1                              ' FP code we can use if
needed. Just good programing
End Sub
' Engine code, do not change or modify. This is needed by the engine to
reconize this vault item. Changing this will cause errors.
 v_fpxKicker1On=1                              '  KEEP THIS SET TO 1.
'  /END fpx Vault Kicker1fpx
' Placed in main fpx Vault Routines ******
' -------------------------------------------------------
' *** fpx Vault Kicker1fpx ***
' -------------------------------------------------------
Const VaultKicker1fpxDebug=1                              ' Dev Debug. Set to 1 to
generate debug text for this vault item
Dim
v_fpxKicker1On
,v_fpxKicker1Memory
,v_fpxKicker1Lights
(9,9),v_fpxKicker1Bulbs(9,9),v_fpxKicker1LightCount,v_fpxKicker1BulbCount
     ' Variables
Dim v_fpxKicker1Case,v_fpxKicker1CaseEnd,v_fpxKicker1CaseStart
          ' Scoring variables
Dim m_fpxKicker1p1,m_fpxKicker1p2,m_fpxKicker1p3,m_fpxKicker1p4
               ' Player(s) memory
If v_fpxKicker1On=1 THEN                              ' Checks if vault item is being
```

```
used
 Set v_fpxKicker1Lights(1,1)=LightKicker1fpx1:Set v_fpxKicker1Lights(1,2)
=LightKicker1fpx2:Set v_fpxKicker1Lights(1,3)=LightKicker1fpx3:Set
v_fpxKicker1Lights(1,4)=LightKicker1fpx4
 Set v_fpxKicker1Bulbs(1,1)=BulbKicker1fpx1:Set v_fpxKicker1Bulbs(1,2)
=BulbKicker1fpx2:Set v_fpxKicker1Bulbs(1,3)=PlasticKicker1fpx1          ' Bulb
and Plastics
' BAM bulb
 BulbKicker1fpx1EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx1EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx1EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 BulbKicker1fpx2EXT
.Brightness=(fpxBulbBrightness):BulbKicker1fpx2EXT
.GlowRadius=(fpxBulbGlowRadius):BulbKicker1fpx2EXT
.GlowBrightness=(fpxBulbGlowBrightness)
 PlasticKicker1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticKicker1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticKicker1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
' light lens
 LightKicker1fpx1EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx1EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx1EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx2EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx2EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx2EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx3EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx3EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx3EXT
.GlowBrightness=(fpxLensGlowBrightness)
 LightKicker1fpx4EXT
.Brightness=(fpxLensBrightness):LightKicker1fpx4EXT
.GlowRadius=(fpxLensGlowRadius):LightKicker1fpx4EXT
.GlowBrightness=(fpxLensGlowBrightness)
 PlasticKicker1fpx1EXT
.Brightness=(fpxPlasticBrightness):PlasticKicker1fpx1EXT
.GlowRadius=(fpxPlasticGlowRadius):PlasticKicker1fpx1EXT
.GlowBrightness=(fpxPlasticGlowBrightness)
 v_fpxKicker1LightCount = 4                    ' Number of Lens Lights used
(in set code) in FOR NEXT loops
 v_fpxKicker1BulbCount = 3                     ' Number of bulbs used (in
set code) in FOR NEXT loops
 v_fpxKicker1CaseEnd=5                        ' Last case number for scoring,
used to reset case to beginning if over at next ball
End If
' Main scoring routine. On target bank, v_fpxKicker1Case scores then
increases by one
Sub AddVaultKicker1fpx()
 IF (fpTilted = TRUE) OR (GameInProgress=0) THEN Exit Sub:END IF
```

```
      ' See if in tilt state or no game in progress to exit the subroutine and stop
any scoring
 If v_fpxKicker1On=1 THEN                                    ' Check if vault item
is being used
  IF VaultKicker1fpxDebug=1 THEN  AddDebugText "AddVaultKicker1fpx()"
            ' Dev Debug Code
  FOR x = 1 To
(v_fpxKicker1LightCount):v_fpxKicker1Lights(1,x).State=BulbOff:NEXT
      ' All lens lights off first
  LockDisplay=0                                ' Clears any music priorty code
  DisplayBlinkInterval=(FlashForMSBlinkInterval+40)                  ' Sets new
interval for DT lights
  FOR x = 1 TO v_fpxKicker1BulbCount:v_fpxKicker1Bulbs(1,x).FlashForMs
(MusicIntervalTime), (FlashForMSBlinkInterval), BulbOn:NEXT        ' Bulbs
behind object will Blink rapidly for time set
  AddMusicSet "kicker"                         ' Note. Default music is overwritten
by any AddScoringEvent code, so you need this default in case there is no
special scoring feature
  Select Case v_fpxKicker1Case
   Case 0 :                            ' v_fpx CaseStart = 0. no light lens till next
case
   Case 1 :                            ' v_fpx CaseStart = 1. first light lens used
   Case 2 : v_fpxKicker1Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn                ' Second Light lens used
   Case 3 : FlushDisplay()                            ' Have to add flushdisplay here
before extra ball/special/jackpot routines
   Case 4 : FlushDisplay() : v_fpxKicker1Lights(1,2).FlashForMs
(MusicIntervalTime), (DisplayBlinkInterval),BulbOn            ' Last light lens, so turn
back on LightLens 2 to keep repeating till loss of ball
   Case 5 : v_fpxKicker1Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn                ' After last case, this just keeps
repeating for the ball in play till ball lost
  End Select
  AddVaultScoreKicker1fpx1()                                      ' Goto scoring
code as set by developer. Must be before increase case code! (geez shiva)
  v_fpxKicker1Case = v_fpxKicker1Case+1                                   '
Increases value by 1 for select case routines
  IF v_fpxKicker1Case> v_fpxKicker1CaseEnd THEN
v_fpxKicker1Case=(v_fpxKicker1CaseEnd)                   ' Wraps back to
caseStart if over max CaseEnd
  TimerKicker1fpx1.Set True, 1500                          ' kicks ball (after set
delay time)
  Kicker1fpxMemorySave()                                    ' Save to that players
memory
  CloseVaultKicker1fpx()                            ' Run closing light routine

 END IF
End Sub
' Timer to reset target bank after a delay.
Sub TimerKicker1fpx1_Expired()
 TimerKicker1fpx1.Enabled = False
 If v_fpxKicker1On=1 THEN                                          ' Check if vault item
is being used
  IF VaultKicker1fpxDebug=1 THEN  AddDebugText
"TimerKicker1fpx1_Expired() "                        ' Dev Debug Code
  Kicker1fpx1.SolenoidPulse : PlaySound "kicker"                       ' Resets drop
target bank
 END IF
End sub
' runs the correct light routine and restores proper light if
v_fpxKicker1Memory=1
Sub CloseVaultKicker1fpx()
  If v_fpxKicker1On=1 THEN                                         ' Only if this feature
is set to "1" and nothing else
   IF VaultKicker1fpxDebug=1 THEN  AddDebugText "CloseVaultKicker1fpx()
":AddDebugText  " ":END IF                ' Dev Debug Code
   FOR x = 1 To (v_fpxKicker1LightCount): v_fpxKicker1Lights(1,x).State =
BulbOff:NEXT
   Select Case v_fpxKicker1Case                                        ' Sets next light
to display based on Case
    Case 0: Exit Sub
    Case 1: v_fpxKicker1Lights(1,1).State = BulbOn:
v_fpxKicker1Lights(1,1).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn       ' 1k, light 10k light for next case
    Case 2: v_fpxKicker1Lights(1,2).State = BulbOn:
v_fpxKicker1Lights(1,2).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn       ' 20k
    Case 3: v_fpxKicker1Lights(1,3).State = BulbOn:
v_fpxKicker1Lights(1,3).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn       ' Extra Ball
    Case 4: v_fpxKicker1Lights(1,4).State = BulbOn:
v_fpxKicker1Lights(1,4).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn       ' Special
    Case 5:v_fpxKicker1Lights(1,2).State = BulbOn:
v_fpxKicker1Lights(1,2).FlashForMs (MusicIntervalTime/2),
(FlashForMSBlinkInterval),BulbOn      ' Any thing over
   END SELECT
  END IF
End Sub
' Game Start. This resets the variables. This is used by all Vault items and
called directly in the main engine code
Sub Kicker1fpxGameReset()
 IF v_fpxKicker1On=1 THEN                          ' Check if vault item is being
used
  IF VaultKicker1fpxDebug=1 THEN  AddDebugText
"Kicker1fpxGameReset()":AddDebugText  " ":END IF              ' Dev Debug
Code
  TimerKicker1fpx1.Set True, 20                          ' Resets kicker (after set
delay time)
  FOR x = 1 To (v_fpxKicker1LightCount): v_fpxKicker1Lights(1,x).State =
BulbOff:NEXT                   ' We turn off all the lights first before we update the
scoring
```

```
  FOR x = 1 TO v_fpxKicker1BulbCount:v_fpxKicker1Bulbs(1,x).State =
BulbOn:NEXT                    ' We turn on all the Bulb lights first before we update
the scoring
  v_fpxKicker1Case=(v_fpxKicker1CaseStart)                          ' Resets
scoring case setting back to starting default (set in vault user options)
  m_fpxKicker1p1=(v_fpxKicker1CaseStart):m_fpxKicker1p2=(v_fpxKicker1Cas
eStart):m_fpxKicker1p3=(v_fpxKicker1CaseStart):m_fpxKicker1p4=(v_fpxKicker
1CaseStart)     ' Resets variables back to initial starting point
 END IF
End Sub
' Run at NewBall. Also used as a blanket reset used for tilt, startup or game
over. This is used by all Vault items and called directly in the main engine code
Sub Kicker1fpxBallReset()
 IF v_fpxKicker1On=1 THEN                              ' Check if vault item is being
used
  IF VaultKicker1fpxDebug=1 THEN  AddDebugText "CloseVaultKicker1fpx()
":AddDebugText  " ":END IF               ' Dev Debug Code
  If v_fpxKicker1Memory=1 Then                                     ' Look to see if
player memory feature is on if it is then...
   Kicker1fpxMemoryLoad()                                   ' Load in last
v_fpxKicker1CaseStart made from loss of previous ball
  ELSE                                ' or if player memory feature is off
   FOR x = 1 To (v_fpxKicker1LightCount): v_fpxKicker1Lights(1,x).State =
BulbOff:NEXT                   ' We turn off all the lights first before we update the
scoring
   FOR x = 1 TO v_fpxKicker1BulbCount:v_fpxKicker1Bulbs(1,x).State =
BulbOn:NEXT                    ' We turn on all the Bulb lights first before we update
the scoring
   IF v_fpxKicker1CaseStart > 1 THEN v_fpxKicker1CaseStart = 1
' error catcher and prevents people from cheezing
   v_fpxKicker1Case=(v_fpxKicker1CaseStart)                              ' Reset
v_fpxKicker1Case back to the beginning(user selectable top of script)
   CloseVaultKicker1fpx()                          ' runs the correct light routine and
restores proper light if target bank player memory feature is set to 1
  END IF
 END IF
End Sub
' Saves the Case settings for each player at the loss of a ball (Selectable by
user). This is used by all Vault items and called directly in the main engine
code
Sub Kicker1fpxMemorySave()
  IF v_fpxKicker1On=1 THEN
  Select Case CurrentPlayer                              ' we see which player  is
playing.
   Case 1:m_fpxKicker1p1=v_fpxKicker1Case
   Case 2:m_fpxKicker1p2=v_fpxKicker1Case
   Case 3:m_fpxKicker1p3=v_fpxKicker1Case
   Case 4:m_fpxKicker1p4=v_fpxKicker1Case
  End Select
  IF VaultKicker1fpxDebug=1 THEN                                   ' Dev Debug Code
   AddDebugText "Kicker1fpxMemorySave()":AddDebugText " - fpxKicker1Case
```

```
=   " & (v_fpxKicker1Case):AddDebugText " - fpxKicker1Case =   " &
(v_fpxKicker1Case)
   AddDebugText " - fpxKicker1Case =   " & (v_fpxKicker1Case):AddDebugText "
- fpxKicker1Case =   " & (v_fpxKicker1Case)
  END IF
  END IF
End Sub
' Loads the "Case" settings for each player at the start of a ball, and restores
that value back on his next ball if Memory=1.
' This is used by all Vault items and called directly in the main engine code
Sub Kicker1fpxMemoryLoad()
 IF v_fpxKicker1On=1 THEN
  FOR x = 1 To (v_fpxKicker1LightCount): v_fpxKicker1Lights(1,x).State =
BulbOff:NEXT                  ' We need to restore the light memory for each player,
so all lights off
  FOR x = 1 TO v_fpxKicker1BulbCount:v_fpxKicker1Bulbs(1,x).State =
BulbOn:NEXT                   ' PF bulb lights should be on, but lets make sure
  Select Case CurrentPlayer                              ' Now we look at the memory for
the player that is up to see which light should be turned back on
   Case 1:v_fpxKicker1Case=m_fpxKicker1p1
   Case 2:v_fpxKicker1Case=m_fpxKicker1p2
   Case 3:v_fpxKicker1Case=m_fpxKicker1p3
   Case 4:v_fpxKicker1Case=m_fpxKicker1p4
  End Select
  IF v_fpxKicker1Case=> v_fpxKicker1CaseEnd THEN
v_fpxKicker1Case=v_fpxKicker1CaseStart                   ' Wraps back to caseStart
if over max CaseEnd
  Select Case v_fpxKicker1Case                          ' Restore the proper light
the extra snazzy way
   Case 0
   Case 1:v_fpxKicker1Lights(1,1).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 2:v_fpxKicker1Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 3:v_fpxKicker1Lights(1,3).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 4:v_fpxKicker1Lights(1,4).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
   Case 5:v_fpxKicker1Lights(1,2).FlashForMs (MusicIntervalTime),
(DisplayBlinkInterval),BulbOn
  End Select
  IF VaultKicker1fpxDebug=1 THEN                          ' Dev Debug Code
   AddDebugText " - m_fpxKicker1p1 =   " & (m_fpxKicker1p1):AddDebugText " -
m_fpxKicker1p2 =   " & (m_fpxKicker1p2)
   AddDebugText " - m_fpxKicker1p3 =   " & (m_fpxKicker1p3):AddDebugText " -
m_fpxKicker1p4 =   " & (m_fpxKicker1p4)
  END IF
 END IF
End Sub
'  /END fpx Vault Kicker1fpx
' **************************************************************************
' **                              Vault                                   **
' ***********************************************************************
' *** These subroutines are Master subroutines and are used by all Vault items
***
' These are for use by the fpxEngine. Any other templates these subroutines
must be linked to within that templates code
' Run at NewBall. Also used as a blanket reset used for tilt, startup or game
over. This is used by all Vault items and called directly in the main engine
code.
' ResetTiltedState(): ResetForNewPlayerBall()
Sub VaultBallReset()
 ' Kickers
 IF v_fpxKicker1On=1 THEN Kicker1fpxBallReset()

End Sub
' This resets the variables for the start of a game. This is used by all Vault items
and called directly in the main engine code
' ResetForNewGame() : EndOfGame()
Sub VaultGameReset()
 ' Kickers
 IF v_fpxKicker1On=1 THEN Kicker1fpxGameReset():END IF

End Sub
' Closes ScoringEvent code pointed to by background timer for additional
instructions.
' TimerCloseScoringEventCase is the control variable
Sub TimerCloseScoringEvent_Expired()
End Sub
' Saves the v_fpxAVCase settings for each player at the loss of a ball
(Selectable by user).
Sub VaultAVMemorySave()
 ' Kickers
 IF v_fpxKicker1On=1 THEN Kicker1fpxMemorySave()

End Sub
Sub VaultAVMemoryLoad()
 ' Kickers
 IF v_fpxKicker1On=1 THEN Kicker1fpxMemoryLoad():END IF

end sub
```

Plastics and Spare Parts

vault_fpxPlastics1

Vault - Plastics and Spare Parts - Plastic 1

Vault - Plastics and Spare Parts - Plastic 1

How to use this Vault Item in the fpxEngine

Using this Vault item and including it in your table design is very simple. It's just a couple copy and pastes from one fpt file to another!
Don't know How? Just click here.

Plastics

How it works

This Vault item is a non-scoring table part. Some parts of a pinball table design require a shaped design to guide the players ball to certain areas of the table. Because fpx uses a lighting system for the bulbs and the plastics, several premade parts are included for the designer to use. These can be modified and reshaped according to the developers needs, and to "fit" his design better. In most cases, all that is needed is to copy and paste one line of code from the Vault item template into the main fpxEngine script. There are no settings needed for adjustments.

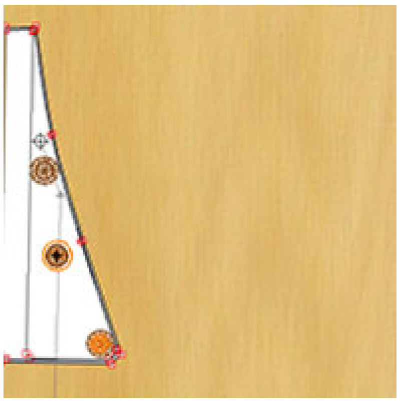
How to Modify

Unlike other Vault items, these are generally used to "fill in" parts of your design, and can be modified anyway you wish. Most of these parts have very simple scripting, and usually contain bulb and lights used to shape the plastic in the code. Everything else within the vault item can be safely deleted if you wish, but it is recommended you just reshape the other parts, as they are used to set the "heights" of the plastics, and if you delete the transparent edge surface, the plastic will drop down to the playfield level and will need to be assigned a new surface height.
You can also add other parts such as posts or rubbers as you see fit.
The fpxEngine Vault item templates all use the same system for the layout design. A bulb underneath a light shaped as the plastic. We also use 2 "surfaces" with each vault item, which is detailed below. Note that every Vault item also has these 2 surfaces.

- The transparent edging around the lighted plastic. This is preset for you and matches all the other vault items. All plastic trim surfaces have a " ***t_*** " prefix in front of the surface name (as a example: *t_fpxHeadConnect1*)
- The plastic ball guide. This is set to 16 height, and is used to guide the ball. All ball guide surfaces have a " ***m_*** " prefix in front of that surface name (for example: *m_fpxHeadConnect1*)

A Note

These vaults are very simple to use, there is just 1 line of code to copy into your script to use.



vault_fpxPlastics1
A simple corner piece suitable for connecting to the top header portion of a table.