SR6 COMBAT TOOL
INSTRUCTIONS & CODE GUIDE

OVERVIEW

This document describes the SR6 Combat Tool, a tool intended for use by GMs to manage combats in Shadowrun Sixth Edition (SR6), serving as both instructions and a guide to the code to aid anyone that wants to modify/use the tool.

LICENSING & LEGAL

BACKGROUND

This application was something I put together for use when running my Shadowrun games. Its purpose was to be able to track lots of actors in a combat, and take care of the basic record keeping and dice rolling, so that I could worry about the more complicated stuff. The original version was for SR4, which was updated for SR6, including tracking Statuses, as well as tracking Edge and handling of the various types of Edge rolls.

One of the big shortcomings with this tool is that, while it took care of tracking initiative and stats, there are several other situations where every actor in a combat needs to roll checks - surprise, perception, and stealth being the big three. In order to add that capability, the tool would need to track the stats for each actor in the combat, and that was a significant update. So I was hesitant to do it, save for an additional consideration: 3x5 cards.

When running SR games, my habit is to use 3x5 cards, with the various stats listed on the front, with summaries of the rules for things like augmentations, qualities, and gear on the back. This works, but in really big combats, it's easy to lose track of cards. In addition, the preparation of the cards ends up taking up a huge chunk of my prep time.

With the need to roll group stealths/surprsise/perception checks, and growing irritation with 3x5 cards, I decided to bite the bullet and incorporate the info on the 3x5 cards into the tool. This has turned into a major update.

INSTALLATION

MacOS can be bit twitchy about downloading apps you find online. Generally speaking, you can right click and select open on the app icon, and it will prompt you to all the app to be run. You can also download Xcode for free, and compile the app from source yourself.

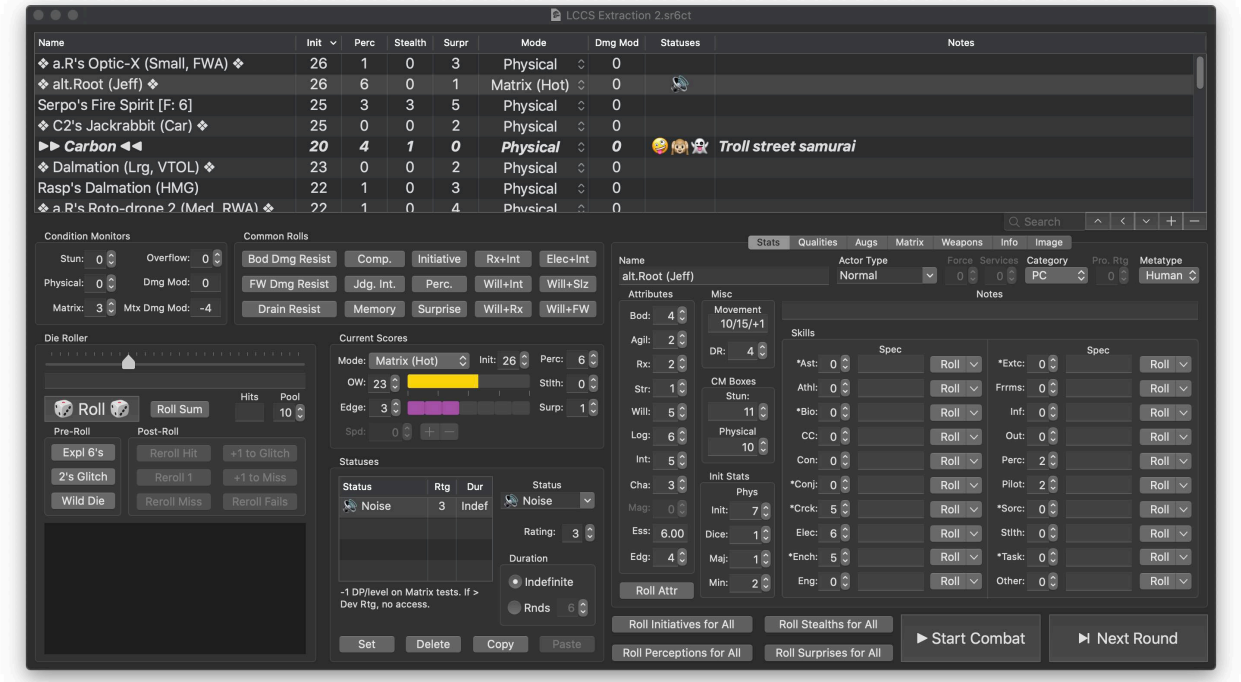
OVERVIEW

The general approach of the tool is to take care of the basic mechanics, and tracking of what is going on - but with minimal enforcement of the rules. As such, you can do things with it that the rules don't cover, or don't really make much sense (e.g. spirits can be designated as Grunts). What the rules are is generally given in tooltips, but the GM is free to ignore or disregards. The only rules that the system enforces are the ones related to initiative, statuses, and damage modifiers - but the GM is generally able to ignore or override them as desired.

One of the primary goals is speed - all information and rolls should be, at most 2 or 3 clicks away. Since there is a lot of information associated with a character, the UI ends up being a bit busy, but is still useful.

ACTORS

Actors is the term used to describe anything that may be involved in a combat - - PCs, NPCs, grunts, critters, spirits, sprites, etc. The tool is essentially a front-end on a database of actors. Actors have a type, which is the type of actors they are (spirit, sprite, normal, etc.), and a category (PC, Grunt, etc.). The Active Actor is the actor currently acting in a combat. The Selected Actor is the actor whose information is currently being displayed.



MAIN SCREEN

The primary user interface is shown in the above image. It consists of several main areas:

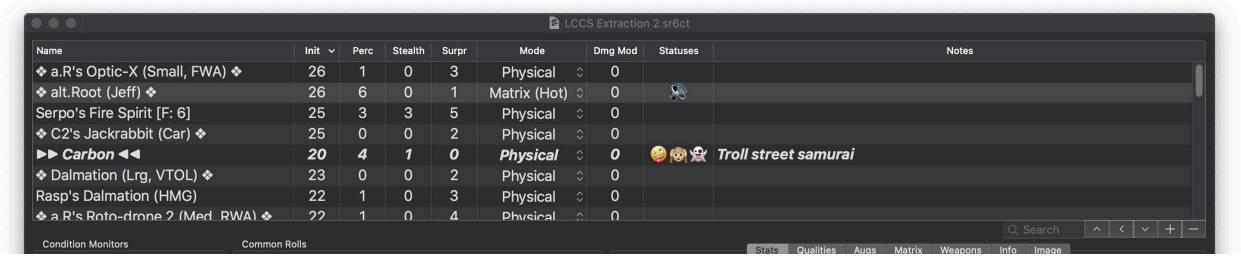
- Actor Table – This is the table at the top of the screen, listing all the Actors in the combat. Buttons at the bottom right are for adding and removing Actors, setting and changing the Active Actor.
- Condition Monitors – Between the Actor Table and the Dice Roller on the left hand side of the screen, this area focuses on the current condition monitor status of the selected Actor.
- Dice Roller – The bottom left of the screen is the interface for the built-in dice rolling element.

- Actor Stats – The tabbed section on the bottom right contains information about the Actor selected in the Actor Table.
- Common Rolls – In the middle-left, below the Actor Table, is an array of buttons providing single-click access to a number of common dice rolls for the selected Actor.
- Current Scores – Below the Common Roll section, the Current Scores section covers transient information for the selected Actor that will change frequently during the course of an encounter (other than condition monitors). Things like Overwatch Score, current mode/ plane (Physical, Matrix, Astral), vehicle speed, and Edge.
- Statuses – Below the Current Scores, along the bottom of the window, the Statuses section is for viewing and editing the status conditions affecting the selected actor.
- Actor Stats – This section, on the right hand side of the screen, beneath the Actor Table, is for viewing and setting the various statistics associated with the selected Actor.
- General Action Buttons - The bottom right portion of the window contains buttons for dealing with all Actors involved in a combat. In addition to having all actors make specific tests, this is where a new combat can be started, and a new round of combat.

The general expected flow for the GM is to prepare for a given combat in advance of a session, adding the actors, and spending most of their time in the Actor Stats section. To speed things along, the GM can copy and paste Actors from other files, and make minor adjustments as needed.

During a session, the Actor Stats section can be modified as needed, but most of the time the GM will just use that section for reference and for rolling skill checks, but otherwise will be working mostly in the sections on the top and bottom right section of the screen.

Actor Table



The actor table provides a summary view of all the actors, showing critical summary information. Information can be edited in the table by double clicking (except for the Damage Modifier and Status information), or edited elsewhere in the window for the Selected Actor.

The Selected Actor is the actor that the use has most recently been selected (clicked on) by the user. This is indicated by being highlighted in the table (lighter grey in the above image).

The Active Actor is the actor that is currently acting in the initiative order. They are indicated by text being in ***bold italic***, and by "▶▶" and "◀◀" tags around the name field in the left column.

Other categories of actor (PC, Grunt, or Lieutenant) are also indicated by tags around their name in the Actor Table. PCS are marked with "❖", Grunts with "⊙", and Lieutenants with "⊙⊙".

The table can be sorted by any of the given values. The tool defaults to sorting by Initiative, and doing things like starting a new combat, or a new round, will cause the table to automatically resort by Initiative.

Some values can be edited directly in the table - the name, notes, mode, and the Perception, Stealth and Surprise hits. Other values, such as the initiative, can be edited elsewhere.

At the bottom right of the table are several buttons for manipulating the Actors on the table.

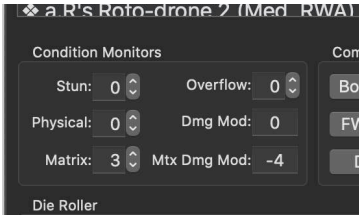
The "+" and "–" buttons are for adding a new Actor, and removing the selected Actors, respectively. The other buttons are for manipulating the Active Actor – moving to the previous or next Actor in the table, and for making the currently Selected Actor the Active Actor.

The Active Actor feature is designed to help the GM keep track of who is going when, but isn't critical. You are can ignore the Active Actor, or not, as desired. You can also set the Active Actor manually, if you need to skip around. For small combats, it's generally not that big a deal. For larger ones, I find it useful.

Note: If the table isn't sorted by Initiative, moving to the Next or Previous Actor will not select the next actor in the initiative order - it selects the actor based on how the Actor Table is currently sorted.

Lastly, there is a Filter button, allowing you to quickly filter down the displayed list to find a specific Actor.

Condition Monitor Section



This section is for viewing and setting damage to the Selected Actor. Which condition monitors are available depend on the type of the actor (Spirits don't have a Matrix condition monitor), and the category of the Actor (Grunts have a combined Stun/Physical monitor).

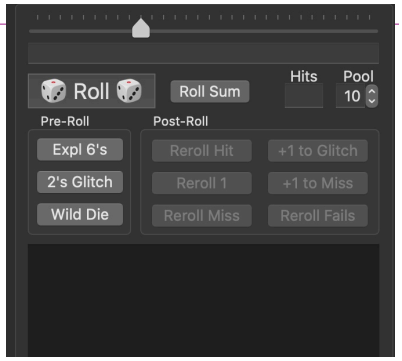
Enforced Rule: The damage modifiers are calculated automatically, including the modifier based on being a Grunt or Lieutenant (-1 per 2 boxes, rather than -1 per 3 boxes for normal Actors) and can't be overridden. You can choose to apply (or not) the modifiers on a given roll using the roll popovers (see below).

Enforced Rule: Exceed a given condition monitor automatically rolls damage over to the proper monitor if you use the stepper control. For example, if the Selected Actor only has 8 Stun Condition Monitor Boxes, has 7 boxes of stun, if you add 4 boxes of stun damage by clicking on the stepper 4 times, it will atomically roll the excess stun over to the physical condition monitor. Excess physical damage goes to Overflow.

Enforced Rule: Actors marked as Grunts or Lieutenants have a single combined condition monitor, rolling excess damage directly to Overflow.

Not Enforced Rule: The number of condition monitor boxes is NOT modified for Lieutenants to include their Professional Rating. The GM should set that directly in the Actor Stats Condition Monitor boxes.

Dice Roller Section



This section is for rolling dice. You use the slider to set the number of dice in the pool, and click Roll to roll that size a dice pool. A summary is provided below the slider, and the details are added to the log window at the bottom of this section.

The various options for the rolls, including Wild Die, Pre-Roll Edge actions, and Post-Roll Edge actions. For the Post-Roll Edge Actions, a given button is only enabled if the roll option is applicable (e.g. if there are no 1's, the "+1 to Glitch" is not available).

There is also an option to just roll and sum the pool of dice,

like in an initiative roll.

When rolling skill and attribute checks elsewhere in the tool, the rolls are generally shown and logged in the Die Roller section. Group rolls (where each Actor rolls a dice pool) are *not* logged, to avoid spamming the log.

Not Enforced Rule: Performing an Edge action does *not* adjust the currently Selected Actor's Edge. The edge cost of a given action is shown in a tool-tip, but is not subtracted from the Selected Actor's current Edge.

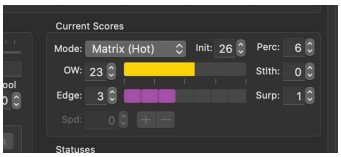
Common Rolls Section

This section allows the GM to roll some of the most common dice pools with a single click. The tool automatically adjusts the size of dice pool as appropriate for the test (no damage modifiers on damage resistance and drain tests, etc.). The specific options used for a given test are shown in the Dice Roller Section, and the pool can be adjusted and re-rolled as needed.

Not Enforced Rule: The GM can use these buttons to make rolls that make no sense (e.g. a Spirit can roll Willpower + Sleaze, a Sprite could try and make a Body Damage Resistance test). The tool *could* enable/disable to prevent this, but I elected not to, on the grounds of allowing the GM maximum flexibility.

Current Scores Section

The Current Scores section is for dealing with those attributes that tend to change a lot during a combat - other than damage, which is handled in the Condition Monitor section. This includes their Initiative, Overwatch Score, Current Edge, Speed (for vehicles), and the hits on tests that all Actors can be called on to make (Perception, Stealth, and Surprise).



Users can also set the current Mode or Plane of the Selected Actor - Physical, Matrix, or Astral. This can also be done on the Actor Table.

For Vehicles and Drones, the Speed section is enabled. In addition to the usual controls to set the speed, you can use the + and – buttons to increment the speed by the Acceleration of the vehicle. The speed in mph and kph is calculated, as is the penalty from the Speed Intervals.

Not Enforced Rule: The Speed Interval penalty is *not* applied to tests for that vehicle.

Not Enforced Rule: When the Overwatch score pegs at 40 (or 50), the tool does *not* automatically Dumpshock the selected actor. Note also that the graphical control maxes out at 40, but you can set the Overwatch score higher in the text box.

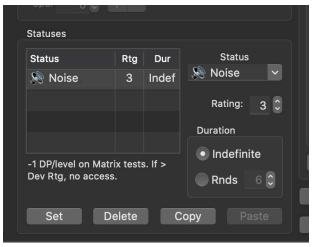
Statuses Section

This section is for dealing with the various statuses associated with the selected Actor.

To add a Status, select from the combo box, set the rating and the duration (the default is Indefinite), and then click on the the Set button. You can modify an existing status by clicking on it in the Status table, modifying the settings, and clicking on set. The status symbol/icon is also shown in the Actor Table for quick reference.

You can also copy the set of statuses on the selected Actor, and paste them onto another Actor. This allows you to apply the effects of flash-bangs and the like a bit more quickly.

Matrix Noise is registered as a Status in the tool, rather than



tracking as a separate element. When the rules for Background Count are finalized, I'll add it to the tool in a similar fashion if appropriate. I may also do a generic Dumpshock one at some point, if that makes sense.

The rules associated with a given status can be seen in the tooltip for that status in the Status Table, and in the text below the table.

Rule Not Enforced: There is no limit on which status can be applied to which actor type, which means you can set statuses that don't make much sense - like Dazed on IC.

Rule Enforced: The duration and rating of statuses is managed by the tool. When you click on the "Next Round" button, it will reduce the remaining duration of all statuses, reduce their rating (if appropriate), and remove them entirely if the status has expired.

Rule Enforced: The dice pool modifiers for statuses are applied to rolls where appropriate by default. In the popovers you can include or not include the status modifiers as desired. Where there is some question if the status modifier, the tool goes with the worst-case option (e.g. it always applies the penalty for Deafened, even if the check does not rely on sound).

Actor Stats – Stats Tab

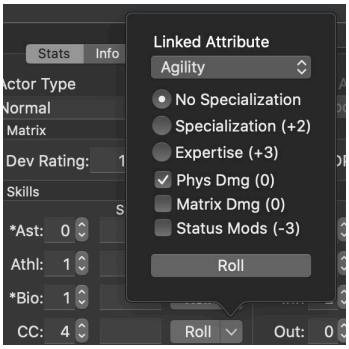
This is the first tab of the large section on the bottom right of the screen, and is for tracking the individual stats of the selected Actor.

The Name field is the name of the Actor, and is also seen (and is editable in) the Actor Table.

The Actor Type is for the type of actor - spirit, sprite, IC, vehicle, etc. The tool enables/disables various elements that do not apply to the selected Actor Type (e.g. vehicles and drones don't have Stun Condition monitors). Likewise for the Category - Normal, Grunt, or Lieutenant.

Awakened characters have an Astral Initiative section, and no Matrix Initiative section, as such characters generally don't spend much time in VR.

Some Actor Types have their stats expressed as variation from a given rating - Force for Spirits, Level for Sprites, Rating for Agents (assuming we get them in SR6 at some point). For those characters, the attributes are set in terms of F+/-X, while skills are either F or –. When making rolls, the total automatically sets the pool based on the given rating for the Actor.



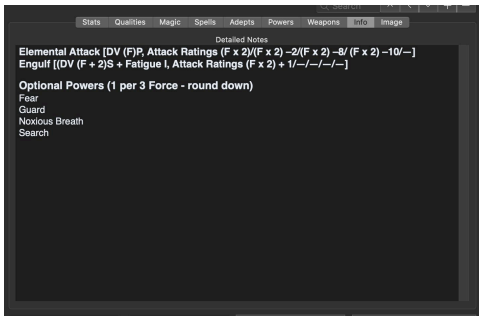
Rules Not Enforced - The tool does not automatically derive stats based on input values (e.g. Initiative is *not* set to Reaction + Intuition). This is partly because there are a *lot* of things that can change those scores, and trying to code them all in is not practical. In addition, making the UI track what has been changed from the default and how turns into a real mess pretty fast. The tooltips provide guidance on what the values should be.

This section also has a mechanism for making attribute-only tests, and skill tests. For skills, the buttons next to the skill make a "default" roll if you click on the "Roll" section - the tool gets the

default attribute, adds in the skill pool, and the default set of damage modifiers. If you click on the down arrow next to the "Roll" button, it brings up a popover allowing the GM to customize the roll - which attribute to use, which modifiers to apply to the test, and if a specialty or expertise applies.

The "Roll Attr" button brings up a similar popover, focused on making Attribute tests using one or two attributes.

Actors Stats – Info Tab



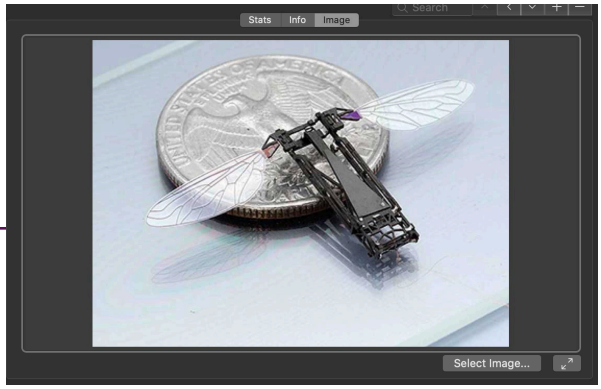
The Info Tab in the Actor Stats section is for more free-form information.

The "Notes" field is shown in the Actor Table, and is for providing quick reference info. I typically use it to identify which miniature it is, and the stats for their primary weapon.

The Detailed Notes field is a rich-text field where you can put whatever additional information you want. You can apply basic formatting using the Format menu, or the usual key commands (⌘B for Bold, etc.). It's not a full-featured rich text editor by any means, but it gives you the basics. If you want to do nice formatting, use Text Edit, and copy-and-paste. The tool parses pasted text to fix text that is too small or is too low-contrast to read. There is no effective limit to what you can put in here - I put in the text for optional critter powers, special rules, etc.

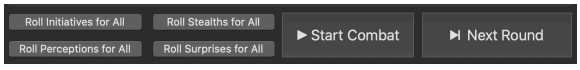
Actor Stats – Image Tab

The tool also provides a place for the user to provide a picture for a given actor. You can either drag-and-drop a picture into the image well, or browse for the image file using the select image button. The "Expansion" button displays the image in a separate, large window, so the GM can show it to the players.



Note: Performance can be an issue with large pictures (or particularly extensive Detailed Notes. Apple's CoreData is pretty good about such things, but it can cause problems.

General Action Buttons Section



This final section is for doing things that affect every actor - having every actor make a specific test, starting new combats, etc.

The "Roll X for All" buttons allow you to have every Actor make the given test - handy when rolling Surprise, or asking for general Stealth and Perception tests. The tests aren't logged in the Dice Roller, but the results of the tests are shown in the "Current Scores" sections.

The Start Combat button does a couple of things - it clears all statuses, removes all damage, sets the Overwatch Score to 0, sets the current Edge to the Edge Attribute, and sets the speed of all vehicles/drones to 0. It then rolls initiative for all Actors, and makes the first Actor to go in the initiative order the Active Actor.

The Next Round button is a bit simpler - it processes the statuses (decrements the duration and rating of any statuses as appropriate, and deletes any expired ones), and sets the Active Actor to the first Actor in the initiative order.

FEEDBACK

If you have any comments, questions, concerns, suggestions, bugs, etc. feel free to drop me an email at ed.pichon@gmail.com. I'm obviously not a UI designer - I am a grunge-coder at best, so I make no guarantees about the quality of the tool. I find it useful, maybe you will too.

FAQ

WHY DOESN'T THE TOOL DO X?

Generally because I didn't think to do it, couldn't figure out a good way to do it, or didn't think it was worth doing. But I'll happily take suggestions!

WHAT IS THE DIFFERENCE BETWEEN THE ACTIVE ACTOR AND THE SELECTED ACTOR?

The Selected Actor is the actor that was last selected – clicked on – in the Actor Table. The info for the Selected Actor is what is shown in the various other parts of the window. The Active Actor is the Actor that is currently acting in the initiative order - it's indicated by the "▶▶" and "◀◀" tags in the Actor Table, and by being in ***bold italic*** in the Actor Table.

Whenever you add/paste an actor, or change the Active Actor, the tool automatically makes it the Selected Actor.

THERE'S AN ITEM MISSING FROM THE LIST OF WEAPONS/POWERS/SPELLS/ETC. THAT I'D LIKE TO ADD. HOW DO I DO THAT?

The data for this is in a "core" database file in the app bundle. The "DBBuilder" app, included in the source, is used to edit/add/remove items from the core database. The trick is to make sure you edit the core database file used by the combat tool. It's named "Core.sr6ctdb" and it's in the app bundle. You can either crack open the bundle and edit the file, or edit the Core.sr6ctdb file in the Xcode project files, and re-compile the app.

WHY CAN'T I MAKE AN ACTOR THAT CAN BOTH ASTRALLY PROJECT AND GO ON THE MATRIX?

Honest answer - because I ran out of room for the UI for tracking those stats. I needed to remove some controls, and it made sense (to me at least) to overload the Matrix/Astral initiative stats bits.

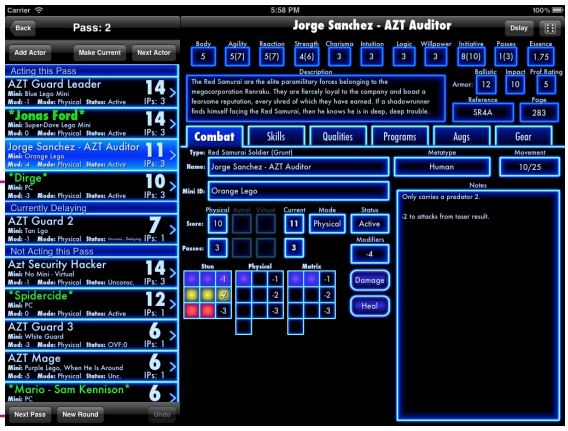
I HAVE AN ACTOR THAT NEEDS TO USE STUFF THAT THE TOOL DISABLES. HELP!

If you want to make something crazy, like a Spirit that can use the Matrix, the tool makes it a bit tricky, because it assumes that Spirits don't need matrix attributes. As a general rule, you can always fall back to setting the Actor Type to Normal or Awakened, and it'll give you access to

most everything you need. You won't be able to have the stats derived from the Force, but it'll work.

WHY DID YOU WRITE THIS IN OBJECTIVE-C? SWIFT IS WHERE IT'S AT!

When I first built this tool (for SR4 - with no Stat tracking), Swift didn't exist. And I don't have time (or interest) to learn a new language.



COULD YOU MAKE AN IPAD VERSION OF THIS?

No. Or rather, I did, for 4th edition. And it was much more full-featured - full character builder, combat tracking, campaign database, the works.

But I ran into a problem - I couldn't get permission from Catalyst to deploy the thing. I had to use a bunch of stuff from the rule books (text, images, etc.) to make it useful, and asked for their permission to use their logo and assets for a split of the sales price. But they had already granted all the rights to someone else - I presume the Herolab folks. And without their permission, I wasn't going to be able to publish it on the App Store without serious legal issues.

The situation may have changed, but doing this on the iPad is a non-trivial challenge, and I don't have the time to do it. Plus I'd probably have to learn Swift…

CODE GUIDE

If you don't care about programming, or have no interest in diving under the hood, you can stop reading now.

If you want, you can download the source code, build It in Xcode, and you should be off to the races. It *should* be easy, even if the compiled app bundle can't be run from Github. If you want to modify the code and tinker with things, please do. Per the CC license, you are can - but you need to share your new version, attribute my work, and not sell it. Ideally, make a pull request on the GitHub site so I can incorporate it into the "baseline".

Anyway, this section is intended to give you a guide to the code, in addition to the hopefully helpful comments I've put in the source code.

To be clear, I do not claim to be a professional programmer. I'm a grunge-coder: I can make simple tools and utilities for my own use, in a single-developer environment. I'm mostly self-taught from a book or two, and (if it wasn't completely obvious from the code) I rely on Stack Overflow to figure out how to make things work. For people that know what they are doing, I'm sure that you can see dozens of easier/better ways to do things.

GENERAL ARCHITECTURE

The tool is built as a document-based application in Objective-C, using CoreData for the back-end. It uses the View-Model-Controller pattern (sort-of), and isn't doing anything particularly exotic. I rely on the standard bindings for most of the interface, with a handful of value transformers.

SOME TERMS

I use a couple of terms in here that should probably be defined:

- Actor – the core managed object. Each Actor shows up in the Actor Table.

- Actor Type – The type of actor, as in a critter, sprite, normal, vehicle, etc. Lots of stuff depends on the Actor Type.
- Actor Category – The category of the actor, as in "normal", a PC, a Grunt or a Lieutenant. Not a whole lot of stuff depends on the Actor Category.
- Level-Based Actors – An actor where the ratings of skills/attributes is based on baseline stat, such Spirits where the Body attribute is the Force plus/minus some value. This covers Spirit, Sprite, and Agent Actor Types.

DATA MODEL

The data model is a flat table, with a *lot* of fields. I could have split the statuses, skills and/or attributes into separate tables, but I'm not entirely certain how. In the original version of the tool (for 4E), there were no statuses, and it didn't track the various stats - I ended up with too many organically. Xcode tends to complain about it, but I haven't noticed any problems. Yet.

In general the allowed ranges for the stats are a bit larger than what the rules define. This is to let GMs cheat the rules in special cases if needed.

There are a couple of cheats/hacks in the data model to be aware of:

- For Level-Based Actor, the value stores for an Attribute is the number added/subtracted from the base Level. As in, a Spirit's Body is stored as "-2" for "Force - 2".
- For Level-Based Actors, the skills are stored as 0 (for no skill) or 1 (for the Level).
- For Status Durations, a duration of -1 indicates "indefinite".

MAJOR CLASSES

SR6Actor

This is the primary object for handling the managed object for an Actor. The interface into CoreData is in SR6ActorProperties, which was auto-generated by CoreData. The SR6Actor class also includes most all of the "rules" logic that applies to a single Actor - calculating damage modifiers, creating dice pools, managing statuses, etc.

ActorArrayController

This object is an NSArrayController subclass that handles the array of Actors from the database. It includes the rules logic for things that apply to all of the actors - the group dice roll methods, round mechanics, etc. It also includes the elements for dealing with the Selected Actor, and adding/removing Actors from the database.

SRDice

This is the class that handles dice rolling, and Edge-actions involving dice pools. Fairly straightforward - call the rollDice method with the appropriate parameters, and it configures a bunch of properties that the UI can pull from to show the results.

Note: SRDice uses the Marsenne Twister random number generator, from Dick van Oudheusden, under GPL 2.0. A very nice piece of code that I use for the dice rolling routines.

Document

This is a subclass of the NSPersistentDocument class, which provides all the functionality for dealing with documents - saving/loading, tabs, etc. This is where most of the user interface code lies, and is the controller for the primary (only) window view.

Broadly speaking, there are a couple of major sections in the code for this:

- Dice roller code. This links the dice roller UI to the SRDice object. I could have pulled this into a separate class, or used bindings more intelligently, but this is pretty old code, and I

didn't want to mess with a scheme that worked, even when I had to crack it open to add the Edge handling bits. So I left a working bit alone as best I could.

- Number formatting code. This code is responsible for shifting from "normal" mode where attributes and skills have numeric values, and Level-Based Character mode. This is done by manipulating the number formatters and value transformers used by the controls.
- Delegate code for the combo boxes, table views, and detail text view. The most complicated bit here is the delegate an format actions for the Detail Text View, which try and keep the text from being too small, or too dark, to read.
- Menu handling code, for setting the availability of menu items, and for executing commands where needed.
- Copy paste UI code for handling copy/paste. Most of this is taken care of in the ActorArrayController, with the code are mostly passing messages along.
- Popover Control Code, for handling the Attribute and Skill rolling popovers.

There is one tricky bit in here that bears further mention - for the multi-line text views (the dice roller log, and the detail text view) I use a couple of performSelector calls to force processing after the current method runs. This is to address two problems - the delegate callback indicating that text is being pasted only passes an unattributed string, so you can't tinker with the formatting. And you can't get the attribute string from the text view because it hasn't pasted yet. So, I use the performSelector to check the formatting after the paste finishes. The other problem is with the dice roller log - trying to scroll to the bottom of the log when it gets long takes a moment, so we do the scroll down after a short delay. Both are hacks - there is probably a better what to do it, but I couldn't figure it out.

The Document.xib is the interface builder file for the interface. It's exactly what you would suspect. It is also where the views for the two custom popover interfaces reside.

FUNCTIONAL BITS

There are some aspects of how the tool works that probably bear with some additional explanation.

Status Handling

Handling the statuses is a bit of a bear, since (among other things) having two tables tied back to the same managed object makes things tricky. There is probably a really elegant way to do this, but what I've done is have the Actor Array Controller serve as the table view data source for the Status table. It in turn looks to SR6Actor to provide an array of data about the statuses associated with that actor. So much of the status handling code within the SR6Actor is about managing the array representation of the statuses, and keeping that array in sync with the managed object data. It works, but it does mean break the MVC model as the Document class has to know a bit about how the statuses work in order to make the display work correctly.

Copy/Paste Handling

Copy/paste is usually pretty easy in the Mac environment, but it doesn't work so well with CoreData. The usual NSCoder methods don't work as NSCoder requires actions at init time, and with a CoreData managed object, you *really* don't want to mess with the managed object at init time. Instead, I code the object to a dictionary, and put the dictionary onto the pasteboard. However, there is a bug when decoding the object from the pasteboard - the Keyed Unarchiver behaves really erratically, particularly when there is an Attributed String in the data object. It will throw an error saying the object wasn't an NSDictionary, even though it is, and even though if you try and unarchive the same data immediately after, it does it fine.

As a workaround, the Detail View string (the NSAttributedString) is placed on the pasteboard separately from the rest of the object. This seems to make the unarchive bug less frequent, but it

still happens. So I've instructed the unarchiver to accept a Mutable String as a valid object type, as well as the expected NSDictionary. If the decode fails, we try again just looking for an NSObject. This works, but Xcode will log a warning saying that using NSObject should be avoided. Oh well.

Status copy and pasting is done very crudely - after banging my head against the above bug, it just keeps a pointer to the SR6Actor that was copied from, and copies the values over on the paste - stupid, but it works. Yes, if the source actor's statuses change between the copy and the paste, the paste will use the new values, but that's a rare occurrence, and behavior I can live with.

StringToSkill Value Transformer

This value transformer is used for the skill rating boxes, and is how those UI elements support Level Based Actors. The transformer has a levelLabel property that can be set externally. For regular characters, this is set to nil, and the transformer works like normal. If it's set to anything ("F", "L", or "R"), the number value is replaced with the levelLabel value if it's not zero, and to "–" if it is zero. The Document class switches the mode of the transformer as needed when the Actor Type changes.

BoolToDisabled/EnabledColor Value Transformer

Apple defaults to disabled controls having a lower-contrast color, which is great. But text labels don't change their appearance when they are disabled. This drives me nuts, so I bind the textColor of the text labels to whatever boolean value is used for the enabled flag, and run it through the transformer. The "Disabled" version sets it to the low contrast mode if false, while the "Enabled" version sets it to low contrast mode if true.