The Well-Versed Informer

Writing Interactive Fiction with Inform 7

1/14/2010
Jeff Nyman
Version 0.1

Contents

Introduction

About These Guides

What I'm hoping is that "The Well-Versed Informer" can be a series of guides to learning the Inform 7 tool. The key word there is "learning." These are not necessarily guides that *teach* Inform 7. Rather, these are guides that I hope can show people how to *start learning* how to use Inform 7 to create works of textual interactive fiction.

Inform is a system for producing such textual interactive fiction, which I'll refer to as textual IF from this point forward. One issue for large manuals that often comes up is that presenting any system in a strictly linear way is a challenge. I think any "Well-Versed" guides should try to meet this particular challenge by thinking of the learning process as a kind of widening spiral, building on the familiar but always opening out into new territory. At times, with certain guides, you'll be shown enough of a future topic to serve as a placeholder, so that you can learn the current aspect of a topic in depth. Later, with the necessary bootstrapping already done, you can go back to the placeholder topic and dig in with more detail. The series of *The Well-Versed Informer* is designed to expose you to as much material as possible as efficiently as possible, consistent with an overall mission of providing you with a solid foundation in Inform.

As with many such systems, Inform is backed up by a programming language. I've found that there's often a feeling that you almost have to know the whole language in order to make sense of any parts of it. Yet that's not true and it's important to see why. Consider this:

Do you have to know *everything* about a car before you start driving?

Did you have to know anything about fuel injection, combustion, or timing belts to drive? Of course not. It's the same with programming in a new language. By necessity these guides are going to show you lots of Inform programming, many of it consisting of just one-line additions to existing source, and then tell you how and why that logic works – just enough to get you rolling down the road. I believe these guides should take this approach because I believe the evidence is pretty solid that we do most of our learning by observing, imitating, and playing around. You should know up front that these guides are a just-get-in-and-drive sort of thing without always looking under the hood. (After all, you can drive a car even if you don't know whether the car has six or eight cylinders.) The guides also move as quickly as possible, not getting bogged down in the quicksand of every detail beyond their immediate topic. The details will come in time, as they are needed; the main thing each guide should give you is forward movement and momentum.

If you just follow along with what the guides are doing, running the programs and altering them to your taste, you'll learn quickly. The more you run these programs, the more fluency you'll develop, and before long, you'll start *thinking in Inform*. That's important! The focus on learning really means internalizing the concepts and developing the heuristics about how to use Inform.

What's nice is that this style of learning is that it also matches that of learning to write fiction. One of your goals, as a writer, is to raise your knowledge of writing techniques from the subconscious to the conscious. This often has to occur during troublesome times, such as when a scene just won't work for you or a character just isn't fitting in quite right. The point is that the more familiar you become with writing techniques, the easier they will be to use, requiring far less conscious, step-by-step precision on your part.

In this sense, and sticking with my previous analogy, using these techniques is like driving a car. When you first learn to drive, you have to pay attention to every detail: where your hands should be on the wheel, how to watch for other cars, when to use the turn lights, and what rules of the road to apply in a given situation. But after a while, these things become second nature. This doesn't mean you should ever settle back completely, letting all awareness slip away as you drive. Certainly there will be troublesome spots – in snowy weather or really bad rain – where you'll need to return to the conscious effort of driving safely. Still, most of the time driving doesn't require step-by-step concentration on your part. In the same way, as you learn writing concepts, they will become second nature. You'll have to concentrate on their individual steps only when you run into troublesome areas.

I believe building up that base of heuristics and that almost intuitive nature of technique is crucial to being successful with Inform, not only because you are practicing writing but because you are practicing programming as well.

Who Should Read These Guides

The Well-Versed Informer is optimized for the reader who wants to create textual IF and is willing to recognize the challenge of combining writing fiction with creating a game. This book is not a replacement for the two manuals that come with Inform: *Writing with Inform* and the *Recipe Book*.

I don't know whether this should be your first port of call for using Inform. Probably not and there's certainly enough material out there for that purpose, but it would also depend on the nature of the particular guide you are reading. Each guide is for someone who wants to get more into the specifics of how Inform works – not only the specific techniques (although the guides should include plenty of those) but also the design principles that make Inform what it is. I'm a great believer in knowing what you're doing. I also believe that knowing what you're doing doesn't mean you have to compose a treatise in your head every time you write a line of code; it means you know how to make the most out of the language, and understand how to analyze problems when they arise.

In the guides I've written I can say that I've hedged my bets a little, in terms of targeted readership, in that I've included some introductory remarks about a number of topics and techniques that are possibly familiar to experienced programmers or those who have gone through the *Writing with Inform* manual or the *Inform 7 Handbook*. I would ask the indulgence of those readers. The remarks in question go by pretty quickly, and I believe that even a few words of explanation of terms here and there can make a surprisingly big difference in how many people feel at home in, and welcomed by, the guides. If you're a more experienced Informer (or programmer) and see passages where the guide seems to be spoon-feeding a bit, please bear with it. It's for a good cause.

By the same token, if this is your first foray into programming, be prepared to do a little extra self-imposed "homework" to get ramped up into the programming process – but by all means, give *The Well-Versed Informer* guides a go.

What These Guides Don't Include

The Well-Versed Informer is meant to be a broad but extensive look at the Inform language. But it probably will never be a complete language reference. For example, there are core elements that I say little or nothing about, and I discuss only a modest number of standard extensions. That's by design. You don't need me to spell out for you how to use every single aspect of Inform, and I don't. What you do need, in all likelihood, is someone to explain to you exactly what an action means, or why you would use an activity, or the distinction between a phrase, a sentence and an instruction, or what a rulebook is and how it differs from a rule.

You need to know these things. And you need to see them in operation and to start using them. Eventually you'll need to plunge deeply into the standard library in your work with Inform in order to truly understand how things are working. How much or how little of that you do really depends on your goals for what you are trying to produce. For my part, I'm aiming to impart a particular kind and degree of understanding in these guides that will make the process a bit easier.

Am I Writing or Am I Programming?

If you use Inform, you are programming. You're also writing.

At one point in the *Writing for Inform* manual, you'll see this line: "... the pretence that Inform is not really a programming language must be dropped sooner or later." At the time of writing this guide, this statement is actually mentioned in chapter 11, section 5, of the manual. That's pretty far into the manual. To be sure, the pretense had to be dropped much, much earlier than that. The sooner you drop that pretense – assuming it even existed in the first pace – the better off you will be.

Like all such skills, becoming totally comfortable with a given language or technology solution requires two things: a good theoretical foundation and a lot of practice. The latter is often predicated upon how much time you spend with one language or technology rather than many. Likewise, the effectiveness of your theoretical foundation is often predicated upon how much time you spend with many languages and technologies rather than just one.

Many programming languages are rich languages, consisting of hundreds of core classes and literally thousands of operations that can be performed with and by these classes. To become extremely proficient with a language means, to a certain extent, to specialize in that language.

It's like the concept of abstraction. The simpler an abstraction – that is, the fewer features it presents – the more general it is, and the more versatile it is in describing a variety of real-world situations. The more complex an abstraction, the more restrictive it is, and thus the fewer situations it's useful in describing. It's the same with knowledge that's generalized versus specialized. The power often comes from being able to move between various levels of abstraction. It's the balance of knowing when to reduce or increase the number of concepts you consider. A rule set that is too relaxed is as much of a problem as an overly restrictive rule set: but that admonition depends on when the rule set is being considered and how it's being applied. I believe that your goal should be to find just the right set of rules – not too general, not too restrictive, and containing no redundancies.

Once you've determined the essential aspects of a situation, you can prepare a model of that situation. Modeling is the process by which you develop a pattern for something to be made. Modeling and abstraction go hand in hand, because a model is essentially a physical or graphical portrayal of an abstraction; before you can model something effectively, you must have determined the essential details of the subject to be modeled. As you'll see in these guides, this resonates quite well with the idea of a model world that you'll build with tools like Inform.

In developing programmatic solutions, it's also important to understand the underlying principles of how people do the things they do to create solutions. That often speaks to the mechanics of the language you're using. But I would argue that what's even more important is *why* people do the things that they do as they develop the solution. These are the principles as opposed to the techniques.

"Thinking in Inform"

Earlier I mentioned the idea of *thinking in Inform*. Then in the last section I mentioned principles as distinct from techniques. I just want to expand on those two related points slightly because they both speak very much to my approach here in terms of how I believe these guides should be written.

I believe that tools like Inform 7 – and the programming languages behind them – become effective when you start to build up an *intuition* about how the tool and the language works. To do that – I mean to *truly* do that – you have to stumble a bit sometimes and see why you stumbled and what you can do differently. That means that these guides should not be sanitized of all wrong paths or non-optimal ways of doing something. Going down various paths is the way that you start to learn the nuances, subtleties and idiosyncrasies of such a tool. I think it's necessary to capture some of that exploratory element in these guides. I believe that's why the Well-Versed "book" (taken as a whole, anyway) is really more effective as a series guides than a single tutorial, manual or reference.

I should also note that part of the artistry of programming is in being able to turn your ideas into computer programs. Once you become proficient with a programming language, you can turn your ideas directly into code. However, before you can do this, you need to see how the programming language you are using understands real-world concepts itself, and how you can relay your ideas into a form that the language can put to use. Yet, in the case of Inform 7, you're not really just writing a program, even in the conventional sense. You're writing a story (and a game) and that provides a different focus for what it means to "turn your ideas directly into code."

Writing *effective games* in Inform can be as simple as playing with conceptual building blocks. Writing *effective stories* in Inform is a little bit more involved, since it involves a lot more than the mechanics but also what you bring as a storyteller.

The idea is to encourage people to think about their textual IF from the ground up, in the sense of considering what scaffolding they need to build to support the story/game they want to write. A connection between what the game world models, what the output prints, and what the parser understands is a large part of what makes a game feel smooth and solid, and allows the player to understand a new kind of simulation. A connection between the ability to tell a story and the ability to allow a reader to have agency (interactivity) with that story while still maintaining the ability to tell a story, with such things as effective dialogue, narrative pacing, and so forth, is part of what makes the game feel cohesive like an experience, and allows the player to understand a new kind of reading – and a new kind of playing.

Inform is easy to read, being closer to natural language than many other programming languages, so there's a good chance that you will spend less time trying to decode what you've just written, which in turn means more time to think about how you could improve and expand your work. That's in theory. However, that does actually put a bit more onus on you, the programmer and author. I believe part of these guides should be a focus on writing self-explanatory code because then, in several months, even years, you will be able to come back to your programs and see immediately what they were supposed to do and what your original intentions were; this makes maintaining those programs much simpler too.

Conventions

Terminology

Let's talk nomenclature for a second. It's going to help you and me immensely if the terms being used in this guide are understood. Even if you disagree with my choice of terms, at least you'll know that choice and can make the appropriate mental shifts to your own terminology as you read.

The first is how to refer to the "user" of any textual interactive fiction that you write. If you consider textual interactive fiction primarily a game, the user is a player. If you consider textual interactive fiction primarily a story, the user is a reader. I don't think it's going out on a limb too much to say that textual interactive fiction is both game *and* story and thus you have a both reader *and* player. As far as I know, there's been no standard understanding of what to call this sort of user. You could presumably just refer to them as a "user." That's a bit dull. Others have used the term "interactor." I suppose that's not an inaccurate title, but it's a little clinical and academic sounding to me.

Since I view interactive fiction primarily as a medium by which stories are told, I'll stick with reader. (This gets interesting in that Inform has many code constructs that refer to "the player." Inform is largely predicted upon the notion that you are creating games and much of its nomenclature reflects that.)

In terms of the "person" that the reader/player is controlling in the story/game, this is often referred to as the "player character" (PC). Others may refer to this as the "avatar" or the "agent." I prefer to stick to the writing theme and call this character the protagonist. (I can even keep the PC acronym and call it the Protagonist Character.) Other characters may populate your textual interactive fiction and those are usually referred to as "non-player characters" (NPCs). I prefer to just call them characters.

I will sometimes collectively refer to all characters in a work of textual interactive fiction as actors or characters.

A few final points probably bear mention. First, you'll notice I refer to *textual* interactive fiction. This is to distinguish the concept from *graphical* interactive fiction. (It's the same distinction as you'd get between "text adventure" and "graphical adventure.") I'll shorten interactive fiction to IF and often speak of "textual IF." Second, I've noticed in some discussions that people don't like to refer to a "work" of textual IF. I'm not entirely sure why but since I've already stated that you can look at textual IF as either a game or a story and since I don't want to type "game/story" a lot, I may simply refer to a "work of textual IF." That being said, when it comes down to whether to use the word game or story, I'll pretty much always settle for story.

Typography

There are a few visual conventions I'm using throughout this book. First I have a few general sectional elements that are meant to simply draw your eye but are small expansions (and usually opinionated to an extent) to the main text.

Something to Note
These sections will contain a brief note about something just discussed in the main text that it might be helpful to call out.

Something to Be Watchful Of
These sections will contain a brief warning about some "gotcha" or element that might trip you up.

Something to Internalize
These sections will essentially be a handy tip or what I would call an effective heuristic for using some aspect of Inform.

I also have a few sectional elements that are meant to be read along with the main body text. Consider these sections as just another part of the main text, but set aside to get you more actively engaged in the material.

Try It!
These sections will contain things that you should specifically try out because it will call attention to some aspect being discussed.

Look Closely.
These sections are meant to call your attention to things that you might otherwise miss.

	Something To Keep In Mind These sections are meant to distill what might have been a lot of information about a given subject into a relatively concise nugget of knowledge.

Read It!
Make sure to look at the differences in output.
These sections will invite you to notice differences in output.

Original Output	New Output
The text as it was.	The text as it now is.

When I want to present Inform source text to you, it will look like this:

SOURCE TEXT

```
The Living Room is a room.
The description of the Living Room is "This is the Living Room description."
```

Finally, it's possible to play Inform games or stories in many different interpreters on many different operating systems. However, what matters is the output that you see on the screen. As such, when I want to show you your story text (as it's being played or interacted with) I'll use the following:

STORY TEXT

```
Living Room
This is the Living Room description.

> LOOK
Living Room (Modified)
This is the (modified) Living Room description.
```