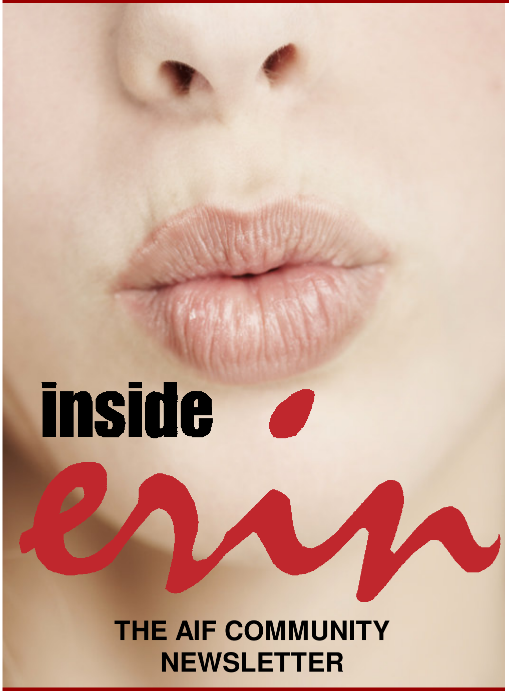
Volume 4 Number 6 June 2008



Contents

A Letter from the Editor	Page 1
This Month in AIF	Page 1
Mini-Comp Games Released	Page 2
SSS Layers	Page 2
How NOT to write AIF	Page 4
Programming Erin	Page 5
Rev: 1st Time	Page 14

Mission Statement

Inside Erin is written and published by people who enjoy AIF. It is done for fun, but we also have some goals that we seek to achieve through the newsletter:

1. To encourage the production of more quality AIF games by providing advice from game developers, and by offering constructive criticism that is specifically relevant to AIF.

2. To encourage activity and growth in the AIF community. We aim to generate a constant level of activity so that there aren't long periods in which people can lose interest in AIF.

3. To help document and organize the AIF community. This is done through reporting on games and events, as well as by helping to organize community-wide activities such as competitions and the yearly Erin Awards.

On May 13th I received an e-mail from A. Ninny informing me that we had received a total of three entries for the 2008 mini-comp. I remember that day well and up until the time I read that e-mail, it had been a very bad day for me. Real life was not treating me well and when I checked in with my favorite hobby I was hoping for some good news. Obviously I didn't get it. My reply to that e-mail is probably something best forgotten but let's just say that it showed my discouragement and disillusionment all too well.

The next day was a much better day for me and I found myself already regretting my hasty words and thoughts of the day before. Then I started reading posts on the AIF Archive asking for an extension and promising more games. I was encouraged by the comments and effort made by members of this community. So much so that I decided to get off my butt, pull out an old game idea that I had abandoned, and get writing myself. You all know the end of the story. In addition to my little game, five other authors came out with one in addition to the original three. Nine games may not be a record, but it gives me hope. To come up with six games in three weeks shows me that there is life left in this community yet, and I'm proud to be a part of it. ◆

After a disastrous turnout for the 12th of may deadline for the mini-comp (only three entries) the deadline was extended to May 30th. In previous years the mini-comp has had much better participation – 12 entries last year – so having only three entries was a serious disappointment. There seemed to have been some confusion about the submission date, and there were a few people that seemed like they might be able to get an entry ready if they had a bit more time, so A. Ninny decided to allow an extension to the 30th, in the hope of getting more games.

Well, fortunately it worked, and we go into the mini-comp with nine games. Surprisingly, we also got another game release (listed below) that fitted the rules of the mini-comp but was not a mini-comp submission. This caused a little confusion, mostly on my part, but of course it's perfectly acceptable to release any sized game at any time. There's no need to enter the mini-comp if you don't want to, and it's definitely nice to see new authors posting games (an all too rare occurrence these days).

As I don't have the info on the mini-comp games yet, I will list those in next month's "this month" column, but the games and voting information will be outlined elsewhere in this issue of *Inside Erin* anyway. Make sure to vote in the mini-comp, and to give the authors some feedback!

New games

1st Time, released 1st Jun 2008, by DaveDKW for ADRIFT 4.0. You are desperate for your frigid girlfriend to put out, and though she may be reluctant, a web-cam chat might loosen her inhibitions.◆

2008 Mini-Comp Games

It looked like this year's mini-comp would be a bust. When the official deadline passed and I had only received three games, I was disappointed, and worried about the health of the community. The mini-comp has traditionally been the most popular event of the year for authors, and if they didn't turn out, I felt that signaled trouble. Fortunately, the three authors who did complete their games on time (A. Bomire, Knight Errant and Dudeman) agreed to give an extension to see if that would shake out any more entries.

I think the knowledge that only three people had entered, along with the three week, extension did the trick. It shook the authors who had begun but not finished, and even apparently got some people to jump in and start from scratch.

So I'm very pleased to announce that the mini-comp is being released now with nine entries. Please play all the games, fill in and mail your vote form by the deadline, which is Monday, June 16. Authors are encouraged to vote. During the voting period, community members should refrain from publicly commenting on the games or requesting hints on the message boards. I will respond to e-mailed requests for hints as best as I can. Questions, comments and vote forms should be e-mailed to NinnyAIF AT gmail DOT com.

Here are the nine games:

Title	System	Author
Office Fantasy: The Boss' Wife	TADS 2	A. Bomire
Lusty Lovers	Inform	Negative Slippy Slide
Office Fantasy: Working Late	TADS 2	A. Bomire
Sexual Awakening	ADRIFT 3.9	Tanner V. Chorus
A Night With Kes	Inform 7	Purple Dragon
Winter Break	Inform 7	Dudeman
A Lady in Waiting	TADS 3	Knight Errant
Bad Day to be a Princess	TADS 3	Evil Bob
Riding Home	ADRIFT 4	Raul

One thing that I've found to be true time and again, is that one of the most important elements to completing an AIF project is planning your interactive sex scene(s) (referred to as "steamy sex scenes" or "SSS"). The first few times you write an SSS you will probably not have too much trouble finding inspiration; the novelty of simply describing the sexual acts will probably carry you through, unless you're already very used to writing about sex. However, once you've written a few you will begin to find that there is a serious snag; the more times you write an SSS, the less easy it will become to do the next one.

The problem is basically this: how many times can you write a description for a "rub tits" command without it being the same thing you've written before? You'll find it increasingly harder and harder to find inspiration for sex scenes, simply because you are doing something you've done before. There is nothing that will stop boredom with the format eventually taking over, but I can, however, suggest a strategy to make writing the SSS easier for both new writers and repeat offenders: layers.

Okay, that needs elaboration. What I mean is you need as many layers of character, story and setting in an SSS as possible to make writing all the responses manageable. There's actually quite a lot of sex text in an SSS, so it should probably go without saying that the scene will have to be quite interesting on multiple levels to make it work for all that time. Now I've said this all before in other articles, really, but I was recently thinking about just how important it is.

The layers that I can think of are as follows:

- **Characters:** Sex scenes are the centrepieces of AIF; they should be more than hum-drum rutting fests, they should actually develop your characters and reveal things about them that the reader didn't already know. What is hidden about your characters that can be drawn out of them in the SSS? For example, maybe the prudish girl is really a bi-sexual nympho? Has the guy loved her for years, but never said anything?

- **Physical features:** This is kind of a subheading of "characters", but is an important one; what physical features do the guy and girl (or girl and girl, or guy and girl and girl, or whatever you want) have? Are her breasts unusually large? Or is his… okay, let's face it – this is mostly about things being big, but it doesn't *have* to be. Maybe she has really nice hair or something. This can have a big impact on some descriptions – for instance, if she does have those big breasts, it's going to make your work clearer for that "rub tits" task. Also, think about how characters *react* physically – are her nipples very sensitive, or can you go ahead and bite them?

- **Dialogue:** How do your characters talk to each other? For instance, is one deferential to the other, or do they argue? Is there anything distinctive about their voices or patterns of speech, like an accent? (Just remember not to overburden your SSS with dialogue at the expense of descriptive text – actions always speak louder than words.)

- **Setting:** What is the location of the SSS? Trust me, a bedroom is the hardest place to set a sex scene – try putting it somewhere more interesting, where the physical surrounds can make a difference to the sexual descriptions. Including interesting objects is an element of good writing. For instance, if you must set an SSS in a conventional home, what about moving it to the laundry? The vibrating washing machine could be a nice prop…

- **Circumstances:** What are the immediate circumstances that have caused the sex scene? Is the girl grateful, or drunk? Is the guy overcome with sudden, animal lusts? Think about it; these things will affect their interactions. This can even be the central element of an SSS; after all, the sex might be something that would never happen except under very specific circumstances (forbidden fruit is always the sweetest).

- **Costuming and props:** This has some cross-over with "setting", but is worth mentioning; your characters may not just be naked. Maybe the girl is still wearing stockings? Or they might have been in such a hurry they didn't get undressed at all. What about uniforms, or, erm, leather goods? What about strap-ons? Whips? Whipped cream? A magic wand? Use your imagination.

- **Kink:** This is not just a subheading of "characters" – it deserves its own heading. Are you putting a special 'kink' of any kind (like bondage, or exhibitionism) into the scene? If so, try to think about how it could affect all sexual acts. This can be a big one – it's often largely what the SSS is all about, but don't forget the other categories too! (And make sure it's a kink you like. Don't bother trying to write a kink you don't like – if you ever finish it, it will still suck.)

Now, what you really want to do is put in something interesting from as many of these categories as you can; preferably all of them (although you should not feel compelled to include an element if it really doesn't suit the scene, or is unnecessary). Then, when you come to that "rub tits" scene you have seven layers to draw upon (or maybe more, if you have several elements from the same layer, like the guy secretly loves the girl *and* she's a secret nympho). It can be hard to get something from every layer into an SSS, but just remember, the more you have the more you can draw upon when you come up to a sex response and you really don't know what to write. Let's see… this "rub tits" task is tricky, but she has big breasts with very sensitive nipples, she's a secret nympho with a bondage fetish, she's drunk, has a French accent and she's wearing a dominatrix costume in this laundry… yeah, I think I can manage to write this (though I probably should have come up with a scene that was a little more consistent…).

You could use this like a check list, though I tend not to be so dogmatic about it. Just remember that even if you have some inspiration for an SSS now, you may find that it runs out half-way through writing the scene. The best time to think about all these layers is *before* you start writing the SSS, so remember this when you are designing your games. ◆

Hey, gang! Bitterfrost here—wisecracker, bushwhacker and all-around slothful slacker.

Add drama queen to that list. Last month, succumbing to the pressures of daily life and the accumulated frustration of a stalled AIF project, I pitched an embarrassing fit in front of the staff. I bitched and whined and announced that I was giving up on my project and moving my junk out of the staff warren.

That was during the traditional newsletter welcome ceremony for Fellatrix, so it was doubly embarrassing. I'm deeply sorry for my drunken tirade and for afterwards when I made a clumsy pass at Miss Trix, spilled my vodka tonic on A. Bomire, set fire to Purple Dragon's sofa, yakked on Knight Errant's shoes, used BBBen as an ashtray and drove A. Ninny's car into the pool. All in all, a good night for me.

However, as the wise Ninny said, "the long rope of AIF" is hard to escape. No matter how far you run away, it always reels you back in. Moments after I declared my intent to abandon all things AIF, I turned around and committed to working on some art for the newsletter and some scene writing for BBBen. And while I can't claim that I've done a lot, I can say that this is more than I've accomplished in months. Damn, I'm an idiot. I can't even leave properly.

So where does this leave my alleged game, "How I Got Syphilix?" At the height of last month's fit of frustration, I wanted to dump it with the millions of "E.T." Atari cartridges buried in the Alamogordo desert. However, I relented. HIGS represents years of tinkering. There's just too much of me in it to abandon it completely. So I've scribbled some "to do" notes for my future-self and accepted that I'll get back to it later this year.

I was tempted to split the hefty file up and distribute it in bite-sized chapter chunks, but that would blow the game's mechanics and the novelty of its absurd, bloated nature—not to mention that it's a beast to split things up in ADRIFT. And there might be vast gulfs of time between releases. And I might change my mind about something from an earlier chapter...

Nah, I'm better off hoarding the thing until it's decided it's done. It's the idiot's way, but it's my way. This is "How NOT to Write AIF" after all. Hopefully ADRIFT Runner will still be around in 2017 so someone will be able to play the game.

Needless to say, my mini-comp entry stalled out on me. I just didn't have time. If I were smart, I'd finish it if for no other reason than to claim I actually finished something I started. And it might provide the momentum I need to overcome HIGS's monumental inertia. That's what any sane AIF author would do. But sanity is overrated. I will, however, help BBBen out as I can, and that should give me a nudge.

Perhaps then I'll graduate up to my mini-comp and then to HIGS—the AIF apprenticeship I should've done from the start... instead of brilliantly deciding to do my last game first. It's just a little brain damage, folks: I'm sure it will pass. Smart? No. Organized? Hardly. Stubborn? Most definitely. The truth is that I can't give up. I can, however, throw embarrassing fits at staff parties.

As for this little column I've been scribbling on cocktail napkins for months and months now, I'll keep at it... but only when I've got some genuine progress to report. My last few entries have been little more than tales of frustration and laundry lists of AIF-avoidances: porn, poetry and pop culture. So it's likely to be more of a seasonal contribution until I hit some sort of lasting stride with my game.

So no matter how my mood oscillates, I'm not giving up. Too stubborn to surrender. Too dumb to quit. Damn. I just remembered I'd mistakenly offered Fellatrix my room when I said I was leaving the staff hive. I can't go back on that. Hopefully she won't mind if I still crash there every now and then. Just leave a cot in the corner for me. Not to worry, I'm on honorable man. Really.

In the meantime, I'll let the truly committed members of the staff carry on the charge. I'll join in as soon as I figure out how to get my sword out of its scabbard and... I... erm. Let's just say, in less Freudian terms, that I'll be back. (Get my therapist on the line). Thanks for reading. If nothing else, I hope this column has shared some of the victories, losses and general madness unique to writing (and aspiring to write) AIF. Take care, all! ◆

Typically, when a new author decides to write a game, the first question he (or she) asks is "which programming language should I use?" All of the major programming languages (TADS 2, TADS 3, Inform 6, Inform 7, and ADRIFT) are perfectly good choices, but they're also each different in their own way. In this series, we will be comparing these systems by taking you through the creation of a simple sample game in each system. By seeing the differences in how each programming language works and how they approach typical AIF tasks, new authors can more easily decide if any given language is right for them.

For the first month we will simply be taking a look at how to get ready to write. What programs you need, where to get them, and how to get them and your first game set up and ready to go.

ADRIFT Segment by BBBen

Okay, the first questions I've been asked to answer on ADRIFT are: how do you get started in the program, and what will you need to download? This is a pretty easy step for budding authors, and part of the appeal of ADRIFT. However, you have a question to answer yourself first; which version of ADRIFT will you use?

Basically there are two usable versions available at the moment: 3.9 or 4.0. Obviously the more recent 4.0 is a better version, but you will have to pay to register it, while 3.9 is free to use. There are a number of small differences between the two versions that would be a bit complicated to go into at this point (plus I can't remember them off the top of my head), but broadly the advantages of version 4.0 can be summed up like this: 4.0 has some extra options that make the engine *slightly* more powerful (i.e. you can do a *little bit* more in it than you can in 3.9); it is easier to organise your projects; it has some better implementation of automated features, so you won't have to override the system quite so much; and finally it is more stable with really large games (3.9 can have some small troubles dealing with particularly big and complex games).

DO NOT bother trying to write your game in the trial version of 4.0. The limitations put in place prevent you from being able to create anything like a real game. The trial version is good if you want to play around and get a feel for the features of 4.0, but really, don't bother trying to complete your game in it unless you plan to pay for the program.

3.9 is still perfectly solid and easy to use. It's more the advanced game designers who will be bothered by what it doesn't do well. I've done the vast majority of my own game design in 3.9, and it's only with my largest projects (which are bigger and more complex by far than the vast majority of ADRIFT AIF games available) that I've felt any strain that made me feel like transferring to 4.0. Still, the registration is not expensive, so don't let me stop you from going to 4.0; the money doesn't go to some callous software company but to Campbell Wild, the creator of the system, and 4.0 is somewhat better for newbies as well as advanced users.

If you're looking for the programs to design ADRIFT games, you can go here: http://www.adrift.org.uk – this is the official site, and should have current downloads for both 3.9 and 4.0. The most up to date version of ADRIFT is never all that hard to find, really, but if you're having trouble getting version 3.9, then you can go here: http://aifcommunity.org/resources/runners/adrift/adrift39_generator_runner_win_install.zip and download it directly from Matrix Mole's archive.

You don't really need anything else to make ADRIFT games. I would suggest typing your text up in a word processor first, but even this isn't really necessary since ADRIFT has a built in spell-checker. Its vocab is a little limited, and it makes the program a little more awkward to use, however, so I prefer to keep it switched off and do my typing and spell-checking in MS Word.

It would also be worth going and checking out the text formatting codes before you start, just for formatting your text properly in the game (so you can have line breaks, bold and italic text, etc.). They can be found here: http://www.thephurroughs.com/projects/atts/atts.html (this is the ADRIFT Topic Tutorial System – it's a pretty helpful resource for finding things out about how ADRIFT works, though you may have to dig through it a little to find this information). Don't worry, I'll add a simple guide on how to handle text formatting in a later article.

You should now be ready to go to start writing with ADRIFT! We'll cover some of the basics of actually making games next month, but this is all you will need to make it all work.

TADS 2 Segment by A. Bomire

Welcome to the TADS 2 portion of the Programming Erin feature. In this portion, we will discuss getting your computer ready to create a game in the TADS 2 authoring system. Let me start off by saying this: TADS isn't easy. I don't find it to be particularly hard, but even after having written games for several years in TADS I don't fool myself into thinking that it is an easy system to use. I'm sure that your average new TADS author sees something like this and immediately looks for some other way of creating their game:

```
“You are carrying “;
if (length(parserGetMe().contents) > 0)
{
   for (i:=1;i<=length(parserGetMe().contents);i++)
  {
          if (i > 1 )
                    “, << i = length(parserGetMe().contents) ? “and “ : nil>> “;
          “<<parserGetMe().contents[i].adesc>>”;
  }
}
else “nothing”;
“. “;
```

But, trust me, once you get past the scary part of deciphering what all of those seemingly cryptic symbols mean, TADS is a very powerful language that will allow you to create just about everything you can imagine.

By the way, the above will provide a simplified list of the inventory carried by the player.

What you need to get started

In writing TADS 2 games, there are a minimum set of files/programs which you will need in order to get started. Here are my recommendations, both in which files to download, and how optional/required they are:

- **The TADS Author's Kit – absolutely required**

  This is a complete kit of files and authoring materials to write, compile and test games written in TADS 2. There are versions of the 'Kit available for various operating systems: Windows, Unix/Linux and Mac. You can get the latest version from:

  The TADS website:
  www.tads.org/tads2.htm

  The IF Achive:
  http://www.ifarchive.org/indexes/if-archiveXprogrammingXtads2Xexecutables.html

- **A Text Editor – optional**

  You can use any text editor available, including Notepad or Vi. There is even a text editor built into the TADS Author's Kit. However, using a good text editor can help simplify coding by automatically catching simple errors like unmatched braces. A couple of highly recommended editors for users of Windows are TFE, PFE, and Crimson Editor.

  There are several locations where you can download a good text editor, including the IF Archive:
  http://www.ifarchive.org/indexes/if-archiveXprogrammingXeditors.html

- **The TADS Manual – technically optional, but very, *very* strongly recommended**

  The TADS manual isn't strictly required, as you can get by on examples and documentation found elsewhere; but, eventually you are going to come across something that you can't answer and the manual is the best place to find those solutions. It can be very hard reading, especially if you aren't that familiar with TADS (if this is you, see the Engelberg Tutorial below). If nothing else, you are going to need Chapter 8- Language Reference. My advice: suck it

up, and get it now. You'll thank me later!
The best place to get this is from the IF Archive:
http://www.ifarchive.org/if-archive/programming/tads2/manuals/tadsman255.zip

- **Mark Engelberg's TADS Tutorial – optional, but highly recommended**

  Mark Engelberg wrote this tutorial for his class of 11-12 year olds (around eighth grade for you Americans), and as such is written in a manner that just about everyone should be able to follow. It is much easier to pick up TADS using this than to dive right into the TADS manual! It walks you through creating your first game, discusses many of the more common objects and items, and has the best description of creating verbs with indirect objects that I've ever read.

  As with other TADS resources, one of the best places to get this is from the IF Archive:
  http://www.ifarchive.org/if-archive/programming/tads2/manuals/TADSTutorial.zip

- **An AIF library – optional**

  If you don't feel like coding up all of the various ins-and-outs of sexual interactions between characters, then a good place to start is to include one of the pre-written modules that handles all of the heavy lifting for you. My recommendation is to use either Sir Gareth's SEX.T library, or NewKid's CHICK.T library, but there are other modules available, such as Choice's Rogue Redux.

  Sir Gareth's SEX.T library:
  http://www.geocities.com/sirgareth99/files/libs104.zip

  NewKid's CHICK.T library:
  http://www.geocities.com/abomire/files/chick.zip

  Choice's Rogue Redux library:
  http://www.ifarchive.org/if-archive/programming/tads2/examples/rredux.zip

- **Other TADS modules and examples – optional**

  There have been many, many TADS authors who have developed some really nifty code and examples for the TADS 2 authoring system, and made this available to the TADS author. Before banging your head against the wall trying to re-invent the wheel (or, for some ideas on how others have done a concept you are considering), you may want to check out these samples and examples.

  TADS Examples at the IF Archive:
  http://www.ifarchive.org/indexes/if-archiveXprogrammingXtads2Xexamples.html

Okay, I've got the files – what do I do with them?

Once you've downloaded the files, you will need to copy or install them onto your computer. There are a few things to keep in mind when doing this, such as where will you store your game files? Where will you put your modules and add-ons? These methods work for me, but I'm not going to assume that they will work for everyone.

First of all, open the TADS Author's Kit that you downloaded. For Windows, this is an executable that will automatically install itself on your computer, into the C:\PROGRAM FILES\TADS TOOLS directory. It will automatically create groups and icons for what it installs. I don't have any particular reason to change the defaults, but if you have a different system of installing programs on your computer that conflicts with how TADS will default, then go ahead and feel free to change it where you see fit.

If you downloaded a separate text editor program, it may have an install program associated with it, or it may not. It depends upon which program you downloaded. Or, perhaps you will be using a program already installed on your computer such as Notepad or Vi.

I recommend creating a sub-directory off of your TADS installation directory for the TADS manual. (You *did* download it, right?) If you downloaded the Engelberg Tutorial, then I would do the same for it. Lastly, I would also recommend creating a separate folder/directory for each game you create. It can be another folder under your TADS installation directories, or somewhere else on your hard drive. But, do keep each game in a separate folder. If you downloaded any of the TADS AIF libraries, then I would recommend extracting them to separate folders as well. A good place for them is under your TADS installation directory, just like the TADS manual (but keep them in a separate location). This is also true for any other libraries or modules that you downloaded from the TADS examples section of the IF Archive.

Okay – now what?

Well, now you are ready to begin writing your game. If you are completely new to TADS, then I would recommend going through the Engelberg Tutorial. It is an excellent introduction to writing TADS games, and will familiarize you with concepts such as creating rooms, objects and characters, as well as "compiling" games and running them.

After that, you are really ready to begin your game. At the very least, you will need to copy a couple of files into your folder you created for your new game. One is ADV.T, the main TADS module used in every game. The other file you will need is STD.T, a subsidiary file also used in every game. You can place a reference with your TADS game to pull these files from the main TADS installation directory, or you can copy them into your TADS game folder. I recommend the latter, because I believe that all of the files necessary for a game should be in one location – not scattered over the disk. This makes it easier to back-up your files. (Hint, hint, hint!) Also, the STD.T file is designed to be modified for every game, so if you are going to do so you will need to have a separate copy for each game.

Note: The ADV.T file is designed to NEVER be modified, but some authors still do so. I don't recommend this at all, and especially not for new TADS programmers. The ADV.T file is THE file that defines all of the objects and verbs in TADS, and messing with it can really play havoc in your game. If you need to change something that is defined within ADV.T (and you probably will), then read up on the TADS manual on using *replace* and *modify*. (I told you you'd need the manual eventually!)

After this, if you are using any additional modules or add-ons, such as an AIF library, then copy the necessary files for those modules or libraries into your game directory as well. Consult the documentation for each module/library to determine exactly which files are necessary. Some, such as Sir Gareth's SEX.T library, require more or less files depending upon which features you are going to use. Others, such as NewKid's CHICK.T require all of the files to be copied to your directory.

With all of the prerequisite files in place, you are finally ready to begin creating your game. I'd suggest starting a new file with a .T extension for your game. If you are going to be creating a really big game, with lots of locations, you might consider breaking your game up into smaller sections. For example, in *Tomorrow Never Comes* I broke down each location in the game into its own modular section: the Casino, the MI6 offices, the UCLA campus, the cargo ship, etc. Similarly, I also create separate files for my non-player characters, especially the sexually active ones. For minor characters, you may consider putting them all into one file. Or, break them out as well. Use whatever system works best for you. You can even put everything, locations, characters, etc., into one big file if you wish. This file will get REALLY big (even broken down into sections, some of my files are easily 2000 or more lines long), so you will spend a lot of time paging up and down and finding the sections you are working upon.

Once you have everything organized, you are ready to begin coding up your game. But, that will have to wait until the next edition of "Programming Erin".

TADS 3 Segment by Knight Errant

Welcome to the TADS 3 corner of this feature. TADS 3 is a recent addition to the IF scene, having only emerged from beta a couple years ago. It's an object-oriented language, which means that everything you code will be part of an object (which may not be the kind of object that will appear in the game). All properties and methods are part of an object. This tutorial will give you some idea of how this all works in practice. For alternate or more detailed explanations, you can also refer to the official TADS 3 documentation, here: http://www.tads.org/t3doc/doc/index.htm I will also be available for questions on the AIF Archive or through my email: sigmundvondanzig@gmail.com

In order to begin a new game, you need some game files. The method for setting up a new game varies a bit depending on if you're using Windows or not. The detailed instructions can be found here: http://www.tads.org/t3doc/doc/gsg/creatingyourfirsttads3project.htm Now that you have an empty file, it's time to set up the game. There are some bits of code that are required to make a TADS 3 game work. If you're using Workbench, then it'll set these up for you automatically, but it's still good to know what it all means. Here's the code:

```
#include <adv3.h>
#include <en_us.h>

gameMain: GameMainDef
        initialPlayerChar = me
;

versionInfo: GameID
        name =  'Programming Erin'
        byline = 'by Some Author'
        authorEmail = 'Some Author <author@myisp.com>'
        desc = 'This is a simple game to show how to write a TADS 3 AIF game.'
        version = '1'
        IFID = 'B9E51A13-935C-42AE-B7F7-6F5DC3C17D92'
;

firstRoom : Room
        'Starting Room'
        "This is a room.  "
;

+ me: Actor
;
```

Ok, I realize this all looks intimidating at first glance, but it's not as bad as it seems. The first line connects your game with the TADS 3 Adventure Library, which contains lots of predefined code to make writing a game easier. This includes all the classes that we will be using from here on. The second line connects your game with the American English language file, which contains all the English-language verbs and grammar, as well as all the default responses. The Adventure Library is constructed in a language-less manner, to make translations easier.

Now we're up to gameMain, our first object. All objects need to follow this syntax. GameMain is the programming name of the object. GameMainDef is the “class” of the object. The class of an object provides a large number of predefined behaviors for your objects to cover many common situations. Most of the default GameMainDef properties will be just fine for most games, the only one you'll need to pay attention to right now is initialPlayerCharacter. This is exactly what it sounds like, this points the program to the character that the player will be controlling at the beginning of the game. We're setting the initialPlayerCharacter to the Actor named “me”. The name doesn't really matter, it could be named “PC” or “r312”, just so long as it matches the name of the object you want to use for the PC. This is the only property we need to set for gameMain, so we end this object with a semicolon. All objects and method statements have to be terminated with a semicolon, to mark where it ends. The next object is versionInfo, which sets some basic descriptions for your game. They're mostly self-explanatory, except for IFID. The IFID is simply a random number used to index games. You can generate your own random IFID here: http://www.tads.org/ifidgen/ifidgen

firstRoom is the first in-game object we're defining. It's a room, just like it sounds like. It's class is “Room”, which defines lots of important properties common to all rooms, such as they can't be picked up, they can contain actors, they have four walls, a floor and a ceiling, and they have descriptions. The single-quoted line 'Starting Room' is the name of the room. This is the name of the room that will appear on the status bar in-game. When the player types >LOOK, it will print the name of the room followed by the description, in this case, “This is a room.” The room definition here follows a standard template.

Templates are predefined shortcuts designed to make programming your game faster. In this case, it's using the Room template, which helps define rooms. The template is defined as such:

```
Room template 'roomName' 'destName'? 'name'? “desc”?;
```

The question marks mark which properties are optional in the template. The template automatically assigns the descriptions we used to the appropriate property name. If we didn't use the template, we would have to define each property explicitly, like this:

```
firstRoom : Room
        roomName = 'Starting Room'
        desc = "This is a room.  "
;
```

It doesn't look like much of a difference here, but as you begin more complicated coding, templates will end up saving you a lot of time and coding. Note that the definition of roomName uses single quotes and desc uses double-quotes. This is important, you can't use single quotes where double quotes are needed and vice versa. We'll go into the difference in another month.

Finally, we have the PC, “me”. Actor is a class used for generic “things that can act on their own”, which may be people, animals, robots, or whatever. Since this is AIF, it's probably safe to assume that the PC is going to be a Person, a more specific class for “Actors that are too big to be picked up and carried”. For now, we'll leave “me” as an Actor. Notice the + right before “me”. That's not part of the object name, it's actually a shorthand way to define the location in the object. What it means is “the object 'me' is located in the object firstRoom”. There will be more on this later. We now have the bare-bones source code for the game. If you're not using Workbench, you still need to create a project file and insert some code. For instructions on how to do this, look here: http://www.tads.org/t3doc/doc/gsg/creatingyourfirsttads3project.htm Congratulations, you have a simple game that you can compile and run. There's not much to do here, but it's a start.

Inform 6 Segment by ‘trix

What is Inform 6?

Inform was created by Graham Nelson to write create new interactive fiction that could be played on the Z-machine, the virtual machine used by classic Infocom games like Zork. There are Z-machine interpreters on loads of platforms (about ten million, I should think. I haven’t researched it), so Inform produces games more portable than those of any other IF language. The Z-machine has some limitations (which are the price for such portability, though they are more of an issue for Inform 7 than Inform 6), so a kindly gentleman known as Zarf designed the Glulx VM, which removes the annoying limitations and added some multimedia capabilities. If you write a game in Inform you can compile to Z-code or to Glulx as the fancy takes you, as long as your game does not rely on the particular aspects of one VM.

Inform 6 is the last version of traditional, object-oriented Inform, before Graham decided to amaze and confound everyone with Inform 7. I6 and I7 are as unalike as any two languages you might imagine. (Just try and imagine two languages more unalike; you will not succeed, even if it seems like you have managed it easily.) I6 is much more similar to TADS than to I7. But if you are an I7 programmer, it is worth knowing a little bit of I6 that you can drop into your I7 games where necessary.

What you need

- ♦ You will need a computer, but the odds are pretty good that you have one of those lying around somewhere already. Look around the room and see if you can see one.
- ♦ You need a text editor (notepad will do if it’s all you can cope with) to run on that computer. There is an inform-mode for emacs (http://www.rupert-lane.org/inform-mode/) if you’re an emacsy person.
- ♦ The best place to start to get the rest of the stuff you need is the Inform 6 website http://www.inform-fiction.org/software/current.html, which will also direct you to some installation instructions: http://www.inform-fiction.org/software/install.html.

Before you start that, one thing to check. Are you using a Mac? If so, congratulations! You can use the Inform 7 IDE to code in Inform 6. That will let you write, compile and run your game all in one application. http://www.inform-fiction.org/I7/Download.html

For all you poor, unredeemed non-Mac users, read on:

- You need the Inform compiler. This is the program that turns your source code into a game.
- You need the Inform library. This is a load of Inform code that has already been written for you, which does all the stuff common to most adventure games.
- You need an interpreter: either a Z-code interpreter or a Glulx interpreter, depending on which you want to compile to. The IF archive (http://mirror.ifarchive.org/indexes/if-archive.html) is a good place to get interpreters.
- The other essential thing to have is the Inform Designer's Manual (the DM4). If you are prepared to sit down and read the whole thing, trying things out as you go, that will give you a very thorough and complete understanding of how to write in Inform 6. It's very readable, but the length does put most people off. It is still the best place to look things up or browse particular topics. The DM4 and other manuals are available from http://www.inform-fiction.org/manual/index.html.

Command-line compilation basically involves going to a command-prompt or terminal and typing "inform mygame" to compile to Z-code, or "inform -G mygame" to compile to Glulx. The complications with this are (1) making sure that the computer knows where the inform compiler is, (2) making sure the compiler knows where your game is, (3) making sure the compiler knows where the library is, and (4) giving the compiler extra configuration instructions. I will try and summarise compilation on Windows after I have gone through laboriously describing how to write some source code to compile.

Source code

This is the stuff you, the game author, write. It is written in a plain text file, usually with a .inf extension (for game source) or a .h extension (for library files and extensions). In the Mac Inform IDE, you type the source straight into the IDE and press *Go!*

Here is a very simple Inform 6 source file:

```
Constant Story "Programming Erin";
Constant Headline "^A simple Inform 6 AIF game by Some Author^^";

Include "Parser";
Include "VerbLib";

Object FirstRoom "Starting Room"
 with  description "This is a room."
 has   light;

[ Initialise;
    location = FirstRoom;
];

Include "Grammar";
```

Here is what it all means:

The first line specifies the title of the game (declared as a constant because it never changes during the game). The second line specifies the headline, which is the text displayed immediately following the title when the game opens. The caret ^ characters in the string are displayed as line breaks.

After that, we include two library files: the parser (which contains the code to interpret the player's commands, and some other stuff), and the verb library (which contains the code to carry out the player's commands, and some more other stuff).

Next we declare an object called FirstRoom (which will be a room), and specify a string "Starting Room", which is the name of the room as it will appear to the player.

Inform objects have *properties* and *attributes*. Properties are the routines, text, and data that specify all the interesting and unique aspects of an object. Attributes are just on/off flags, and having any particular attribute is simply an indication that the object has a certain characteristic, including such diverse things as being static or turned on or female. (Maybe all three, like a statue of a whore.)
The "with" clause defines the object's properties. In this case, we set the description, which is what is displayed in response

to a "look" command.
The "has" clause defines the object's attributes. In this case, since we want the player to be able to see in this room, we give the room the attribute light, which means that the room is lit.

The Initialise code is a routine. Routines in I6 are enclosed in square brackets, and contain a sequence of instructions, each ending in a semi-colon. The only instruction in this case is `location = FirstRoom;` which sets the initial location of the player.

After that we include another library file. "Grammar" mainly contains the code for all the different verbs and how they are used.

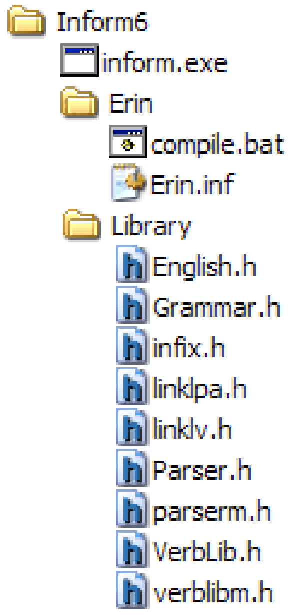
Any game using the standard I6 library needs to include "Parser", "VerbLib" and "Grammar"; it needs to define a room for the player to start in; and it needs to define an Initialise routine, which is run at the start of the game and needs (minimally) to set the initial location. Other library files (such as "English", "VerbLibM" and "ParserM") are included inside the ones we explicitly included.

Compiling

Now comes the fun bit. Here's how you might do it in Windows:

- Create a folder called "Inform6". Inside this folder, create two more folders, called "Library" and "Erin".
- Put the library files (the ones ending in .h) inside your Library folder.
- Put the compiler (probably called "inform-631.exe") in the Inform6 folder, and (for convenience) rename it "inform.exe".
- Copy the source-code above into a text file called "Erin.inf", which should go in the Erin folder.
- Create a new text file, containing the following:
  ```
  ..\inform +..\Library -s Erin
  pause
  ```
  and save it as "compile.bat" in your Erin folder. If you want to compile to Glulx instead of Z-code, put
  ```
  ..\inform +..\Library -Gs Erin
  pause
  ```
  instead. Feel free to compile to both, and then compare the two. That is not compulsory.

You should end up with a structure more or less like this:



```
Inform6
    inform.exe
    Erin
        compile.bat
        Erin.inf
    Library
        English.h
        Grammar.h
        infix.h
        linklpa.h
        linklv.h
        Parser.h
        parserm.h
        VerbLib.h
        verblibm.h
```

To compile, double-click the compile.bat file.

If you see a load of text with the word "Completed" somewhere near the bottom, then it has worked. There should now be an "Erin.z5" or "Erin.ulx" file in your Erin directory, which is your compiled game in Z-code or Glulx format.

If it hasn't worked … well there's not much I can do, because I can't see what's happened from here. I'm not made of eyes. You'll have to go to your nearest convenient IF forum and tell them all about it.

If it worked, you should be able to run your game now (in the interpreter of your choice) and see the exciting room you made.

The next step is to write a more interesting room with some stuff in it, but that will have to wait until next month.

Inform 7 Segment by Purple Dragon

In the previous section, 'trix mentioned the fact that I6 and I7 are very different languages. I do not think she overstated things at all by saying that they are about as different as two languages can be. I7 uses a 'Natural Language' approach to writing IF. That is to say that you write in normal English sentences, which makes it both easier to write, and much easier to read. It might not be quite as powerful a system as I6 or the TADS systems but I haven't tried anything yet (either in an actual game or just playing around) that I haven't been able to do. This month we will take a look at what you need to get started and how to set things up.

What You Need

This is going to be a very short list. Actually, I don't think you can even properly call it a list since there is really only one thing. When you download the I7 program, which you can do from the Inform 7 website at http://www.inform-fiction.org/I7/Inform%207.html, you pretty much get everything you need to get started. The program, text editor, game tester, documentation, and compiler are all combined into a single program, which is convenient to say the least. There are many other resources on writing IF in general and working with I7 in particular. There are also many different Extension (also available on the Inform 7 website) that you may well want to make use of. However, these are all optional, just the main program should be plenty for now.

Setting Up Your Game

Once you have the program, setting up the game is simplicity itself. First, you obviously need to install the program on your computer. Put it wherever you like, as you will probably never need to get directly into any of the files in it. About the closest you are likely to come is examining included extensions and that can be done through the program interface.

Now that you have it installed, go ahead and load it up. You will be presented with a screen offering you three options.

- Start a new project
- Reopen last project
- Open an existing project

The last two should be pretty self-explanatory, and since you don't yet have any existing projects, also pretty useless so let's click on that first one. It now wants three pieces of information.

- The directory where you want to place the new project
- The name of the project
- The author of the project

The directory where you want to put it is probably a bit more important than where you put the actual program. Personally, I have a directory set aside for all my games with a sub-directory for each individual one. You don't have to do it this way of course but I find it helpful to keep everything in one place. You don't actually need to create a directory for the game since Inform will handle that automatically. So let's say you want to put a new project entitled "Programming Erin" in a directory called "My Games." Just browse through the program, find the directory, and once you have entered the title and author, click start. The program will create a directory within "My Games" call "Programming Erin.Inform." Now you have someplace to put anything relating to that project and keep it all together. By the way, don't feel that you have to come up with the perfect title at this point, you can always change it later if you need to.

And that's about it. The program creates the directory, starts up the program, and away you go.

Organizing the game

I just want to end this month by mentioning a good way to organize your project within the program. When creating your game you have the option (and it is technically optional) to use headings for the different sections. I won't explain here exactly how to do that since the documentation explains it well. However, I will say that I highly, highly recommend that you make use of them. Why? One of the few things that I don't like about I7 is that your entire source code (or source text as they like to call it) is kept in one continuous block of text. My recent mini-comp game was very, very short as games go and it still managed to clock in at 16,058 words. To give you an idea of what that means, this Newsletter issue is approximately 9000 words long (a little more than half as many). Can you imagine reading through it if there were no article titles, no subheadings, no breaks in the text, just one paragraph after another? Could you still do it? Of course you could but why make things more difficult on yourself.

What the headings do in I7 is similar to what the titles and headings do in this Newsletter. They break the text down into sections so you know what the text following them relates to. You are allowed to organize the text in any way you wish, letting you have separate sections for different rooms, objects, people, or whatever. Organize it any way you want to, but organize it or when you hit about 5000 words the hair pulling will begin. In addition to listing the headings in the text itself, once you compile the game (either for testing or release) you can bring up an index and move to a particular section with the click of a mouse. The longer your game, the more important headings become but even for short games, they are very helpful.

Well, that should get you up and running. Next month we'll actually start to take a look at how to make things happen. All of us at Inside Erin hope you enjoy this feature and find it helpful. ◆

1st Time

A Review by Knight Errant

Game Info:	1st Time
Released:	5/31/08
Author:	DaveDKW
Platform:	ADRIFT 4
Size:	29KB
Content:	m/f, masturbation, voyeurism (webcams)
Game Type:	ONS
Length:	Short
Reviewed:	June 2008
Extras:	None

DaveDKW's first game is an interesting and original idea. He takes the conventional idea of a horndog teenage guy with a shy virginal girlfriend but actually writes it out in a halfway realistic manner with a long, slow buildup. The combination of the long buildup and his good writing have the consequence of frustrating the player as much as the PC with the eventual interruption. This is not a negative comment in the slightest, as that is the intention of this game. It's an effective teaser (pun intended) for the eventual sequel, hopefully longer.

All in all, 1st time shows good promise. It could have been made better with a more complete implementation and allowing greater flexibility in the order of the player's actions. As it stands, it can be a bit frustrating being forced to perform each sexual action in the order in which the author intended. This seems to be a common problem with new authors, particularly in Adrift. Luckily, it's never too hard to figure out what the player needs to do in order to progress.

Rating: B-

If you can write game reviews, articles, opinion pieces, humorous essays, or endless blather, we want you. Contact the Editor for suggested content or just write what you want and send it to us.

Submitting your work to Inside Erin:

Please direct all comments, articles, reviews, discussion and art to the Editor at aifsubmissions@gmail.com.

Editor:
Purple Dragon has written five AIF games including *Archie's Birthday - Chapter 1: Reggie's Gift, A Dream Come True*, and *Time in the Dark.* He has received one Erin award and been nominated for several others.

Webmaster:
Darc Nite is an avid gamer who heard the call for help with the AIF Newsletter.

Staff:
A Bomire is the author of several TADS AIF games, including *Dexter Dixon: In Search of the Prussian Pussy*, *Tomorrow Never Comes* and *The Backlot*. His games have won numerous awards and Erin nominations. He was the co-recipient of the Badman Memorial Lifetime Achievement Award in 2006.

A Ninny is an AIF player, author of four AIF games and frequent beta-tester. His *Parlour* received an Erin for Best "One Night Stand" game in 2004 and his most recent game, *HORSE,* walked away with three Erins at the 2007 awards show.

BBBen is an author of a number of Adrift AIF games. His games have received numerous Erin awards and nominations and first place in A. Bomire's 2004 mini-comp. He was also the recipient of the 2007 Badman Memorial Lifetime Achievement Award.

Bitterfrost is a longtime IF/AIF player working on his first (and last) game, *How I Got Syphilix*.

Knight Errant is an AIF player who has released one game and is currently working on a couple of others.

'trix has released one game, *Casting*, which was written in Inform 6, and is sporadically working on another in TADS 3.