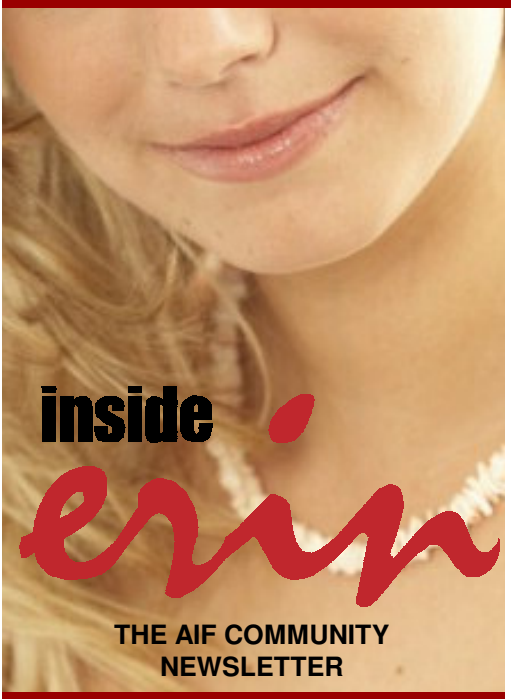
Volume 4 Number 10 October 2008



inside erin

THE AIF COMMUNITY NEWSLETTER

Contents

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.

I hope you all enjoyed last month's Christopher Cole Special. I'm glad that we went ahead and did it, even if it was a lot of work having to play all those games again to be able to write reviews on them. Hey, we make the sacrifice because we love ya so much. Having said that, I wouldn't expect any more issue in the near future to have 18 game reviews in them. Well, I do have a couple for you this month at least. Programming Erin is back as well as another visit from Aphrodite so sit back, relax, and enjoy.

Darc Nite has been the web master for the newsletter website since its beginning. Unfortunately, he has been MIA for several months now. Since I am unable to contact him, I feel have no choice but to drop him from the roster for now. Wherever he is, we at Inside Erin wish him well and thank him for all of his hard work on the site in the past.

Finally, I wanted to discuss the Erin Awards for this year. As BBBen mentions in his "This Month" article, there is some real doubt as to whether or not they will be held this year. This is not a decision to be made lightly, but I simply do not feel that there have been enough games released this year to justify holding the awards as normal.

To give you an idea of what I mean, last year we had 20 games, 12 of which were from the 2007 mini-comp. The year before we had 28 games with 8 of them from the mini-comp. So far this year we have 11 games and of those 11, nine of them were from the mini-comp. Holding the Erins at this point would accomplish little more than redoing the mini-comp that we have already voted on. The games released so far would not be out of luck, just pushed off for consideration in the 2009 awards.

Continued on page 2

Holet's been very busy this month with his live AIF "The Test". It's now quite long, and can still be read and played on AIF-Games.com. There is also a new AIF games list, which can be found at http://www.delron.org.uk/d_aif_games.htm. It can be arranged by various categories, to make searching for games easier, and looks like a very good list.

The deadline for this year's Erin awards is at the end of October, but we've had such a small number of game releases this year (particularly when factoring out the mini-comp) that we are most likely going to postpone the awards until next year, making the 2009 awards a bi-annual event. This isn't a mopey decision – there just aren't the games available to justify the awards this year, so unless there's a huge glut of games this month, all games from this year will have to wait until next year to be considered for the Erins.

It wasn't a very eventful month, so I don't have much to say here. I will remind everyone, though, that the "Threesome comp" deadline is in four months time (the end of January), which is heaps and heaps of time to get an entry ready. I'll state the rules in brief here again: **1 PC, 2 sexual NPCs, 1 non-sexual NPC, 5 rooms, 300kb of multimedia, and only one room may have a threesome.** ◆

Letter from the Editor Continued from page 1

I won't say that the decision is final yet, but I don't see a good reason to go ahead with it as things stand at the moment. The Erin Awards are a community event, and even though *Inside Erin* has traditionally been responsible for hosting and moderating the awards I don't feel comfortable summarily canceling them without giving the community the opportunity to voice its opinion. So if you have any comments or concerns about this decision then please e-mail me with them and your voice will be taken into consideration in the final decision. ◆

With the holidays quickly approaching I thought it might be nice to try to get into the spirit with couple of little competitions. I'm just going to lay out the whole thing here so that you all know what is coming up and are able to plan ahead if you want to participate. All of the following represent a relatively small time commitment. If you've been looking for a way to participate in the community but just don't have the time to write a game then this is perfect for you. The more the merrier, so get those pens and keyboards warmed up and start writing.

Short Story Comp 1

Theme: Halloween
Due Date: Monday, October 27, 2008
Description: We are going to keep these things as wide open as possible so the only criteria here is that it has to have something to do with Halloween in some way. You could go the real monster route and write a story about a haunted house, a pack of zombies, or Mrs. Dracula getting down and dirty and sucking some -- er -- blood, yeah that's it. Or maybe you want to write about something that happened at the office costume party, it's all up to you. I'm going to try to get the newsletter out by the 29th so that you all have a couple of days before Halloween to read them.

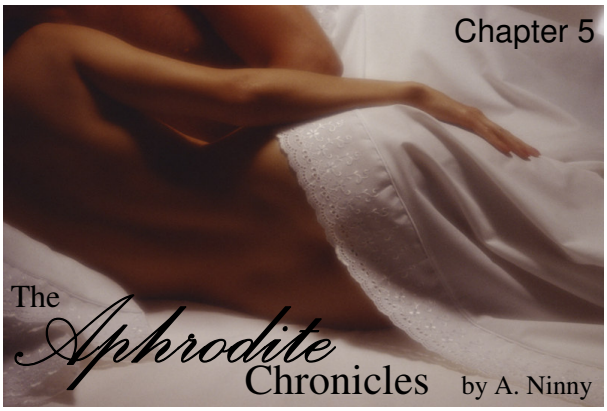
Short Story Comp 2

Theme: Christmas
Due Date: Thursday, November 27, 2008
Description: Different theme but the same basic idea. Write about Santa and elves or about a group of carolers being invited in for something more than just milk and cookies, whatever you want. Just make sure the Christmas theme is in there somewhere.

Poetry Comp

Theme: Christmas
Due Date: Thursday, November 27, 2008
Description: Yeah, I know. I don't expect we'll get the same number of entries for this one as the other two but I still thought it would be a cool thing to try. Even if you think you're no good at writing poetry why don't you go ahead and give it a try? What do you have to loose? Who knows, you might even like it. As above, pick any Christmas type theme that you want.

Feel free to enter any or all of the above. All entries should be sent to me at purpledragon.aif@gmail.com by the due date listed. I will set up a poll for voting on the AIF Archive when the newsletter is released. You may have noticed that I didn't say anything about a length limit, because there isn't one. However, please keep in mind that this is a newsletter and not a book so try to keep it within the bounds of reality. ◆



Dear mortal men and women,

Several months ago I began relating my experience helping a woman named Chica explore and amplify her sexuality. She thought I could help her learn how to make her body into a fully erotic whole. I asked her to make love to herself in my presence, but without my direct participation, but that she was to use my presence as an amplifier of her erotic experience. I will now continue my telling of that story.

Without taking her eyes from mine, Chica pulled her t-shirt up to her neck, but didn't take it off. She wasn't wearing a bra and her nipples looked piercingly rigid; they were pointing right at me, making my fingers curl. She didn't touch her breasts, though. Instead she said, "I love the feeling when the cool of the air first touches my nipples. The transition from warm and contained to cool and free is exhilarating. And I love the feeling when my nipples harden on their own. It's like crossing into a wave of horniness. And I absolutely love to be naked!"

She deftly tucked her shirt into itself so that it wouldn't fall back down and then began lightly tickling her ribs and her belly, her muscles seeming to contract in tiny waves as she moved her fingers over her skin. Her face showed how she was concentrating on every spot she was touching, and she cooed lightly in pleasure.

Her efforts at making me horny as a way to amplify her own experience were definitely having their desired effect, and to keep myself from reaching over and grabbing her, I placed my hands on my thighs and dug my fingernails in, locking them in place and creating a pain/pleasure mix to help distract myself.

Chica hugged herself, squeezing her breasts together, closed her eyes, then smiled a comforted, happy, very private smile. She was clearly getting the idea about whole-self loving, and then she decided that it was time to be naked. She pulled off the t-shirt and set it aside. She wriggled out of her shorts, pulling her panties off with them, revealing a really splendid, healthy, feminine body.

Still hugging herself tightly, she pivoted on her hips so that her back was to me, then pitched forward, curling herself into a fetal position. She had turned so that she would have me full-on facing her ass, and I found myself staring at it longingly. She slyly opened her eyes and watched me staring at her fabulous cheeks and at her pussy lips peeking eagerly out of her thick, natural pubes.

Chica humped the air for a moment, and I could see that she was imagining someone's fingers or cock thrust inside her, causing her to moan, her breath escaping in ragged semi-controlled gasps.

One thing that astonished me in my experience with her was that she would stay almost still in any given pose for several minutes, perhaps fantasizing, perhaps cataloging her different pleasure nodes. Only her face gave away that she was doing anything other than resting, different expressions crossed it – heavy-lidded bedroom eyes, small, private smiles, bottom lip-biting, tongue wetting lips. Even when she started actually masturbating with her fingers on her vagina, it was with an astonishing patience, an almost imperceptible build-up. She'd spend a few minutes stroking herself, then stop, change positions, and go back to hugging herself, or stretching her limbs, or curling up into herself, creating an orgasmic trajectory like a stock market chart, with big ups and downs along the way.

When she finally came, she was on her hands and knees facing me so that I could watch her facial expressions. She was reaching between her legs, and her ass was vibrating, quaking and bouncing; her body was doing the work that most women do just with their hands. Her orgasm was a force to be reckoned with, an astonishing gush of sexual energy released literally over several minutes. It was loud. It went on and on. It was a beautiful thing. I have to admit to being powerfully envious of her. Even her post-orgasmic cool-down was gorgeous. She just lay there, recovering her breath, remembering the sensations, smiling to herself, rubbing her arms.

"I really didn't know I had it in me," she said at last. "Thank you."

"No, thank you," I responded. "And I knew you had it – you just needed to unleash it. You're a force of nature. You need to share it with as many people as you can, this is something important. Find a way to use this to promote female pleasure."

"Can you suggest some things I should do?" Chica asked.

"Sure," I replied. "Start by making a blog. Make videos of yourself. Post you pictures, thoughts, relate your experiences. And, of course, share the goods! Have lots of sex."

She laughed, and agreed, and I took my leave of her, knowing that this rare woman would contribute to the cause. And so she has. Videos of her masturbating can be found on the web site www.ifeelmyself.com. Her blog entitled "Chica and Her Van" can be found at http://chicaandhervan.blogspot.com.

Just recalling my time with her is making me want to masturbate. I'll leave you now and wish you adieu. And until next month, I wish you all wonderful love.

Aphrodite

Salvation

Review by Purple Dragon

Game Info:	Salvation
Author:	LyingBastard
Release Date:	March 28, 2003
Platform:	ADRIFT 3.9
Size:	51KB
Content:	mf
Type:	ANW
Length:	Short
Reviewed:	October 2008
Extras:	None

Basic Story

You are a mercenary, currently working a little war in Africa. There is a raid scheduled for tonight, but you have a bit of free time to kill this afternoon.

Overall Thoughts

The thing that immediately struck me about this game was the amount and depth of background information on both the situation and the PC. The PC seems much more like a real person than do most characters, which is even more impressive when you consider that this was a mini-comp entry and thus, a very short game. I felt like I was given the opportunity to get to know him a bit before being thrown into the sex and it was a nice feeling.

The author tried to do the same thing with the NPC in the game, but I didn't feel that he was as successful here. If you take the time to talk to her you find out some about her background. Although she is currently a nun, she hasn't always been and her life before the church was not a happy one. There is a story there but it is not quite developed enough to show her motivation for what happens in this game. Listen to me talking about motivation for sex in an AIF game. What the hell was I thinking? It is actually a backhanded compliment since other parts of the game were so good that it got me expecting more here.

Puzzles/Game Play

There was not a lot in the way of puzzles. You need to find and use a couple of items but they are not hard to find. One of the things you need to do is to gather your weapons and ammo for the coming raid. Even though you never actually get to that raid in the game the act of gathering the equipment in preparation of it again gives the feeling that the character actually has a life that goes on after the game is done.

Sex

OK, the sex was good, well written, and engaging. However, I don't think enough time was spent dealing with the fact that this is a nun of ten years breaking her vow of celibacy. It is talked about at a couple of points, but not enough. Of course, since Maja had a life before becoming a nun, she does not go into this relationship a virgin, but it still seems like she is just a bit too eager. I can read between the lines and figure out a reason why, but it would have been nice to have a bit more information. It is clear that this is more than just sex for both people, that they have feelings for each other, but where these feeling came from is not made clear. Again, I'm being too hard here considering this was a mini-comp game but those are the things I was thinking about as I read through the otherwise extremely well written sex scene.

Technical

There weren't a lot of technical issues with the game. There were a few typos, but nothing very distracting. I did find a couple of odd things that might be worth mentioning. First, before you get to the actual sex scene, trying the sexual commands gives the response "I don't understand what you mean" which makes is seem like you're typing the wrong thing rather than just not yet to the point where you can do that. Also, when asking Maja to minister to your member the commands are "Maja rub cock" and "Maja suck dick." Dick and cock are evidently not synonyms so those are the exact commands that you must enter. In all fairness, the readme file does list these commands and it's a good thing because if I hadn't read it, I would have missed the blowjob altogether.

Final Thoughts

This whole review feels like I've been harping on characterization and plot points rather than discussing the nuts and bolts of the sex that these games are almost always all about. Of course, it feels like this because that is exactly what I've been doing. If you think I'm being too hard on the game then you are probably right, but you are also missing the point. The fact that I spent so much time thinking about the plot should tell you that there was a good one here to begin with. I feel that the author was hampered a bit by the restrictions of the mini-comp and that he had more to say if he had had the time. This game was written five years ago and to my knowledge, it is still his only game to date. However, I know that he is still around and hope that he takes my comments as encouragement to write another, longer game. I'm sure it will be well worth playing.

Rating: B

Amy and The Raging Hormones

Review by Purple Dragon

Game Info:	Amy and The Raging Hormones v1.1
Author:	Snarfbert
Release Date:	April 23, 2003
Platform:	ADRIFT 4
Size:	24KB
Content:	mf
Type:	ANW
Length:	Short
Reviewed:	October 2008
Extras:	None

Basic Story

You have tickets to the Raging Hormones concert tonight. They may not be your absolute favorite band but they are Amy's. Amy is a girl from work that you really want to get to know better and there is no way that she will not be at the concert tonight.

Overall Thoughts

The game has a decent pace and good writing. It is very short but it was a mini-comp entry so that's not too surprising.

Puzzles/Game Play

There are no puzzles in the game at all. The author warns us about this in his readme file and he is a man of his word. There are just a few preliminaries to get through before getting to the sex.

Sex

Most of the sexual responses in the game tend to be a bit short, but what's there is well written and pretty hot. Actually, I think I enjoyed the initial foreplay with Amy at the concert more that the actual sex scene itself. It's probably the exhibitionist in me, but something about the idea doing things like that with a girl in a public place gets to me.

The main sex scene is pretty straightforward and while you don't have to go in an exact order, you do have to do a bit before jumping into the main course. That isn't a complaint, just a comment. I actually think that doing it this way is a good compromise between having all commands available from the beginning and having a full arousal system where you have to sit through the same block of repeating text over and over again. If you were playing the game like normal not trying to break it (like I tend to do when I'm testing or reviewing one) you probably wouldn't even notice.

One command that I felt was conspicuous in its absence was spanking. A couple of the responses more than suggest that Amy likes a bit of pain mixed in with her pleasure so spanking seemed like a natural, but the game doesn't even recognize the word. Oh well.

By the way, there is no anal sex in this game and trust me, Amy is a girl that you really, really don't want to press the issue with. Of course, with that comment I have just about guaranteed that anyone playing this game WILL press the matter, but don't say you weren't warned.

Technical

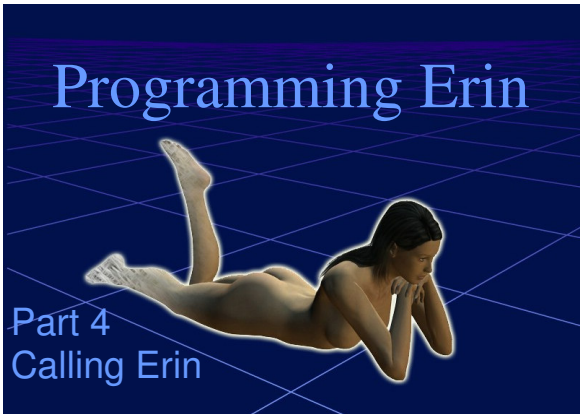
I didn't find any major bugs in the game although, because of the short size and simple format, I wouldn't really expect to find many. One bug I did find can be mentioned in the same breath with something I liked. When it comes time for Amy to loose her clothing you can either do it for her or have her do it herself and the game will respond accordingly for removing her shirt and jeans. This is a little thing, but it is a nice touch nonetheless. The problem comes when it's time to get rid of her underwear. Here you have to do it yourself because if you ask her to do it the game tells you that she is still wearing her shirt and jeans (which she isn't). An unfortunate bug, but not really that big of a deal.

Final Thoughts

A solid little game for a mini-comp entry. Nothing really bad about it but nothing exceptional either means that it gets a pretty average grade. Still, I feel it is well worth the few minutes that it will take you to play it.

Rating: C+

This month we are taking a look at how to manipulate our environment a bit. Everything up until this point has pretty much just been setting the stage. We create rooms, objects, and people and all the different systems have a mechanism set up for the basic handling of them, but for the most part, it's pretty boring. If we want anything really interesting (or even mildly interesting) to happen we have to do it ourselves. This month we program a couple of little things for the player to do. They are pretty simple, but the basic ideas show how the systems can be used for a wide variety of activities and actions.



TADS 3 Segment by Knight Errant

By now we have the basic skeleton of our scenario, but there's nothing that the player can really do yet. TADS 3 provides some basic action handling for the player, but it's pretty boring, suitable only for unimportant objects and illogical actions. In order to actually make this setting into an actual game, we need to override the default handling. This month we'll look at how to do that. Note, I swear I came up with this game idea before I saw A. Bomire's "Working Late" games for this year's mini-comp.

Before we begin on new stuff, though, I'm ashamed to admit some bugs in the code I wrote last month. First, although I defined both sides of the door to Erin's office, I neglected to connect them together. Therefore, the code for officeDoor2 should be:

```
+ officeDoor2: Lockable, Door -> officeDoor 'erin\'s office door' 'Erin\'s
door'
    isQualifiedName = true
;
```

The other bugs are fairly trivial. Immovables like the computers are by default not listed in the room description, so the player wouldn't know they're there unless they're described in the room description. Therefore, we either have to mention them in the room or add isListed = true to the object.

Moving onward. Erin is locked in her office, and right now there's no way for the player to encounter her. In fact, there's no way for the player to know she's in there at all. We need to let the player know where the object of his desire is, and give him some hint at how to encounter her. We'll do this by letting the player overhear a phone conversation she's having. To make sure the player won't miss it, we'll make this scene triggered when he enters the hallway. In TADS, there are often several different ways to accomplish a specific task. This is one way to do it, but it's not the only way.

```
hallway: Room 'hallway' 'hallway'
    "To the south is your office and to the north is Erin's office.  The hall-
way also leads further east to other offices.  <<erinCall.doScript>>"
    erinCall: EventList
    {
        [
        new function
            {
      "As you enter the hall, you hear Erin's voice through the door.  Some-
thing about the tone of her voice catches your interest, so you step closer to
the door.<.p>
<q>Yeah, I'm looking at the pictures right now, I can't believe you talked me
into letting you take them!  That was so hot.</q>  There's a brief pause be-
fore she continues, she must be talking on the phone.  <q>Oh, you're naughty!
I can't now, though, there's a coworker just across the hall, and I can't go
somewhere more private without some kind of excuse ... </q>  The rest of the
conversation is too low to hear.";
      addToScore(1, 'hearing about the pictures');
            }
        ]
    }
    south = myOffice
    north = officeDoor
    east: FakeConnector {"Everyone else has gone home, all their offices are
locked.  ";}
;
```

There are two new concepts we're introducing here. First, look at the property "erinCall". This is a nested object, much like the FakeConnector we're also using in the hallway. Nested objects are exactly what they sound like, objects that are defined inside other objects. The FakeConnector and EventList are separate classes that can be used in their own objects, but in this case we'll only be referring to them in the context of the hallway, so it makes our code more compact and easier to understand by presenting it like this. Now, look at the description of the hallway. At the end of the description, I've added <<erinCall.doScript>>. The << >> are tags that you can insert into a double-quoted string to let you mix programming statements in with the text to output, so that you can vary what's presented based on game conditions. In this case, when the player enters the hallway, it triggers the doScript method of erinCall. As I said, erinCall is an EventList, which means it's a list of events to be

triggered in order, when the doScript property is called. In this case, the EventList has only one event, which means it's called once and then the list does nothing. The event is the "new function", which presents the text and then awards the player a point.

There are a few things to notice about the double-quoted string itself. Since TADS uses quotation marks to specify the beginning and ending of the string, we can't use quote marks in the string to let the player know someone's talking! TADS takes care of that with the <q> and </q> tags, they insert quotation marks for us. Similarly, the <.p> tag is code for a new paragraph. After the double-quoted string is called, we use the addToScore method to add one point to the player's score for hearing about the pictures. Be careful using the addToScore method, because it doesn't have any internal check for if the player has already received a point for that reason yet. In this case, the action will only be triggered once, so it's safe to use it here.

The text hints that there might be a way for the player to entice Erin out of her office so that he can snatch her photos. It also hints that it should be pretty easy, given that Erin is looking for an excuse to have phone sex with her caller. Since she's locked in her office, the most logical way to get in contact with her would be through a phone or intercom. An intercom would cause fewer complications, but a phone would be more likely in an office context. This is a bit more complicated, as TADS 3 doesn't have any default handling for phones. To make things more difficult, phones completely ignore the way TADS 3 handles actions. Normally when a player types >VERB NOUN, TADS 3 only allows the action if the noun is in scope (if the PC can see/hear/smell it). Of course, >CALL ERIN only makes sense when Erin is *not* in scope but the phone is. This will affect how we have to implement the verb CALL.

CALL is a verb with one object (players would be more likely to CALL ERIN rather than CALL HER ON PHONE or CALL HER WITH PHONE), so we need to define CALL as a transitive action.

```
DefineTAction(Call);
```

Having a Call action isn't enough, we also need the verb rules to handle all the possible player inputs. There aren't too many logical synonyms for Call, so this part is pretty easy.

```
VerbRule(Call)
  ('call' | 'phone') singleDobj
  : CallAction
  verbPhrase = 'call/calling (who)'
;
```

Finally, we need to define some default handling for CALL, to handle when the player tries to call things that we don't expect.

```
modify Thing
  dobjFor(Call)
  {
    verify()
    {
      illogical('That isn\'t something you can call. ');
    }
  }
;
```

Now the game responds when the player types call, but TADS will still disallow calling Erin unless the player is in the same room with her. If we wanted to allow the player to have an extended conversation with Erin, then we would make the phone into a SenseConnector. However, in this case all we need to do is parse a "call Erin" command, so the easiest way to do this is with a simple kludge: define an intangible object with the name Erin and put it in the phone. We can put the handling for the phone call in there.

```
+ yourPhone: CustomImmovable 'phone' 'phone'
    "It's just like any standard office phone, except this one is yours.  "
    cannotTakeMsg = 'It doesn\'t do much good if you unplug it and take it
somewhere else. '
    isListed = true
;

++ Intangible 'erin' 'erin'
    soundPresence = true
    dobjFor(Call)
```

```
    {
        verify(){}
        action()
        {
            if (pictures.isKnown==nil)
                "You call Erin and chat for a few minutes, but there's really
nothing you need from her right now.";
            else if (!erin.isIn(erinOffice))
                "The phone rings and rings, but there's no answer.  She must
have left her office.";
            else
            {
                "You call Erin's office, and she picks up the phone.
<q>Hello?</q></p>
                <q>Hi Erin, it's me.  I know you're working on the Anderson
file, but I need to see the expense report.  Could you make a copy for me?</q>
you ask.  It's just an excuse to get her out of her office so you can look for
the pictures, but you hope it works.
                <q>Oh, sure,</q> she says.  <q>Just a moment.</q>";
                officeDoor.makeLocked(nil);
             officeDoor.makeOpen(true);
                erin.moveIntoForTravel(nil);
            }
        }
    }
    dobjFor(Default)
    {
        verify(){illogical('Erin isn\'t here.');}
    }
;
```

This phone call is the most complicated thing we've done so far, so there's a lot of new stuff in here. First is the method dobjFor(Verb). This is the method that handles the actual action when the object in question is the direct object (dobj) of the verb. The handling is broken down into several phases, preconditions, verify, check, and action. Preconditions are what must be in place before the action can happen. For example, in order to take something, you have to be able to touch it. Verify is the phase where we exclude actions that would make no sense to the player (not the PC!), such as eating a computer. This will exclude these items from disambiguation. Check is where we stop an action if we want it to fail but it's still logical. For example, trying to turn on a light when the power is off. Finally, action is where the action actually occurs. The other new thing is the if-else if checks in the action. If is needed to prevent players from getting into illogical situations. In this case, without checking if pictures.isKnown, the player could call Erin and get her to leave the room without ever knowing why he needs to do so. Of course, to make this check work, we need to add a pictures object. This is a simple Thing, so I'll leave that as an exercise for the reader. We also need to add a statement at the end of erinCall, setting pictures.isKnown = true. Finally, at the end of the phone call, three more actions happen. We manually unlock Erin's door and open it (to reflect her passage through it), but we also move Erin into nil. Nil is a convenient place outside of the game world where we can put things temporarily. We could've just as easily made a copy room and moved her into there, but that's not the direction I want to take this game.

Now that the player can actually interact with the world around him in some interesting ways, it's starting to seem like a real game. Next month, Erin will return to her office and we'll finally begin interacting with her directly.

TADS 2 Segment by A. Bomire

Before I begin, I would like to also say that the idea to create a comparative game in an office environment was indeed discussed before anyone knew that I was writing my "Office Fantasy" series for this year's mini-comp. Well, not "anyone" - I knew I was writing the "Office Fantasy" series, and in fact had already begun, but no one else was aware of it. Okay, that's out of the way. Let's continue in creating our own sample game, which also takes place within an office.

In this month's entry, we shall be adding a door to Erin's office, and setting up some automatic responses. Finally, we shall be addressing a common problem - using verbs on objects which are not in sight. Let's begin by adding a door.

In TADS 2, a door is usually made up of two objects - one in the room from which the player will be exiting, and one within the room into which the player will be entering. They represent the two sides of the door. We'll place one "side" in Erin's office, and the other in the hallway. Here is a typical door:

```
Halldoor: doorway
location = Hallway
sdesc = "Erin's door"
ldesc = "This is the door to Erin's office. It is currently
    <<self.isopen ? "open" : "closed">>. "
noun = 'door'
adjective = 'Erin\'s'
isopen = nil
doordest = ErinsOffice
;
```

You'll notice a couple of properties here: *isopen* and *doordest*. The *isopen* property tracks whether the door is open or not, and we can use it in the door's description to describe the door as being open or closed (using a TADS expression). The *doordest* property lists the destination of someone going through the door. In this particular instance, this property isn't very useful. The proper use of a doorway object is to use it in place of an exit in the room in which it resides. So, we need to place the door into the Hallway room in place of the exit to Erin's office, and slightly modify the hallway's description to reflect the change:

```
Hallway: room
     sdesc = "Hallway"
     ldesc = "To the south is your office and to the north is the
  door to Erin's office. The hallway also leads further east
  to other offices. "
     north = Halldoor
     south = startroom
     east = "Everyone else has gone home; all of their offices are locked. "
;
```

All well and good, but now Erin's office also needs a door. The definition isn't very different, other than swapping the *location* and *doordest* properties:

```
Erinsdoor: doorway
location = ErinsOffice
sdesc = "Erin's door"
ldesc = "This is the door to the hallway. It is currently
    <<self.isopen ? "open" : "closed">>. "
noun = 'door'
adjective = 'Erin\'s'
isopen = nil
doordest = Hallway
;
```

Placing this within Erin's office and changing the exit to point to the door will give the same effect as the door we just put into the hallway. Now, one last thing to do and our door will be complete. We need to join these two objects together, so that they act as one. Opening one will automatically open the other, and closing will have the reverse effect. We do this using a special property belonging to just doorway objects: *otherside*. The *otherside* property does exactly what it says: designates the other side of the two-sided door. For Erin's door, it looks like this:

```
Erinsdoor: doorway
location = ErinsOffice
sdesc = "Erin's door"
ldesc = "This is the door to the hallway. It is currently
    <<self.isopen ? "open" : "closed">>. "
noun = 'door'
adjective = 'Erin\'s'
isopen = nil
doordest = Hallway
otherside = Halldoor
;
```

There is a much similar change to be made to the door in the hallway. This is one type of door. Another type is a door which can be locked and unlocked. You designate this type of door by adding the *lockable* class to the object definition, and the locked status is tracked with the *islocked* property. A *lockable* door, however, can be unlocked without a key. Most times, if you are going to bother locking a door, you will want to require a key. For this type of object, you will want to use the *keyedLockable* class instead of *lockable*. The only difference is that a *keyedLockable* object has another property called *mykey* which has a value of a *keyItem* class object which can be used to lock and unlock the object. We'll turn Erin's door into a lockable door which requires a key:

```
Erinsdoor: doorway, keyedLockable
location = ErinsOffice
sdesc = "Erin's door"
ldesc = "This is the door to the hallway. It is currently
    <<self.isopen ? "open" : "closed">>. "
noun = 'door'
adjective = 'Erin\'s'
isopen = nil
islocked = true
mykey = ErinsKey
doordest = Hallway
otherside = Halldoor
;

ErinsKey: keyItem
location = Erin
sdesc = "bronze key"
ldesc = "A bronze door key. "
noun = 'key'
adjective = 'bronze'
;
```

Note that I placed the key into Erin's possession. By default, TADS will not allow the player to take it from her; she has to be told to give it to him. By putting it in her "hands", we are effectively keeping it from the player. Getting it from her could be a puzzle, or perhaps at some point she will leave it someplace that the player can find it - those are decisions which you can make. Oh - and don't forget to also make the hallway door a *keyedLockable* object whose *mykey* property is ErinsKey.

Now that we've done this, the player cannot get in to see Erin in her office. So, how is he to know she's there? Well, he'll probably guess it - after all there's only three rooms and he's seen two of them! But, let's assume that this is a much bigger game, and Erin could be anywhere. So, we'll alert the player that Erin is in her office by displaying a message. There are a bunch of ways of doing this, and one way is to display the message the first time that the player enters a room.

To do this, TADS has provided a special descriptive property that only gets displayed the first time that the player enters a room, or as TADS refers to it, when the room is first "seen". TADS tracks whether a room has been "seen" (i.e., described to the player) using the *isseen* property. You usually don't need to refer to this property but it is important to know it is there - you can use it to keep track of whether the player has been in a room during a certain time of the game - or you can reset it to nil to treat a room as unvisited. This latter is useful if you've changed a room description for some reason and want to display the full room description instead of the terse version a player usually sees when he re-visits a previously visited room. Regardless of all of that, just before a room's *isseen* property is set to true, TADS will display the *firstseen* property of that room. We will use this to display a message to the player letting him know that Erin is in her office, and by placing it within the hallway the player will see it when he first enters the hall:

```
Hallway: room
     sdesc = "Hallway"
     ldesc = "To the south is your office and to the north is the
  door to Erin's office. The hallway also leads further east
  to other offices. "
     north = Halldoor
     south = startroom
     east = "Everyone else has gone home; all of their offices are locked. "
     firstseen =
     {
        "\bAs you enter the hall, you hear Erin's voice through
  the door. Something about the tone of her voice catches
  your interest, so you step closer to the door.
```

```
            \b\"Yeah, I'm looking at the pictures right now, I can't believe
            you talked me into letting you take them!  That was so hot.\"
            There's a brief pause before she continues, she must be talking
            on the phone. \"Oh, you're naughty!  I can't now,
      though, there's a coworker just across the hall, and I can't
      go somewhere more private without some kind of excuse ...\"
      The rest of the conversation is too low to hear. ";
            incscore(1);
         }
;
```

You'll note that I am placing a blank line (\b) at the beginning of this text, in order to separate it from the normal room description. Otherwise, the above text will be displayed immediately following the room description. Also, I am using TADS method of displaying quotes (\") to surround Erin's dialog with quotes. The last thing we do is increase the score by one point.

As you can read from the above, Erin needs a reason to exit her office. Well, let's give her one! We'll have the player give her a call. This involves something that is a little more difficult - addressing verbs to objects that are not visible to the player. Warning: This is a little more advanced in TADS 2, and involves making changes to default functions of TADS. It is not something that the beginner may immediately grasp - but it isn't difficult.

Before we begin, we have to have a brief description of how verbs work. When you say "open door", for example, TADS does a few things. First, it checks to see if the open-verb is allowed to be used upon an object. This is set up when you define the verb. Then it checks to see if there is a door nearby. Lastly, it checks to see if the player can actually reach that door. These last two are automatically defined for all verbs, but we'll need to override them in order to call Erin. After all, Erin will neither be visible nor within the player's reach when we attempt to call her. First, we'll define the "Call" verb:

```
callVerb: deepverb
      sdesc = "call"
verb = 'call' 'phone'
doAction = 'Call'
;
```

Pretty simple, isn't it? The *sdesc* property acts much as it does for any object. TADS displays it when displaying messages regarding the call verb. The *verb* property is much like the *noun* property for other objects - it defines the words the player can use to invoke the verb. Lastly, the *doAction* property both tells TADS that the verb can be used upon other objects (do = direct object), and defines the methods that will be used to test and carry out that action (more on this later). There are a couple of other methods which are inherited from the *deepverb* class, *validDoList* and *validDo*. I could easily devote an entire article to just these two methods, and ways of using them - but my illustrious editor would do mean, nasty, ugly things to me. So, let me recommend that you get the TADS manual and read about these methods. In the meantime, here is the short version:

validDoList - goes through ALL of the objects in the game, and weeds out the ones which are not visible to the player. This much shortened list gets passed to:
validDo - takes the shortened list of objects returned by *validDoList*, and weeds out the objects which are not within the player's reach.

Okay, that's what they do, now how do we change them? Well, *validDoList* is easy: TADS documentation tells us that if *validDoList* is set to nil, then ALL objects are passed to *validDo*. A very simple change to make. Now, we just have to figure out what to do to make the right objects "reachable" by the player. This is handled by *validDo*, which normally looks like this:

```
validDo(actor, obj, seqno) =
{
return obj.isReachable(actor);
}
```

The *obj* in the above method is the object in question. The *actor* is the actor who executed the command (usually, but not always, the player), and finally *seqno* designates which object in the list returned by *validDoList* this object is: 1, 2, 3, etc. It usually isn't important. To alter the above, we need to consider just what we are doing. In our case, we are "call"-ing someone. Note that is "someone", not "something". It wouldn't make sense to allow the player to call a desk, would it? But, it would make perfect sense to allow the player to call Erin (or another character). In other words - objects of *Actor* class. So, what we'll do is modify *validDo* to automatically return true for all *Actor* class objects. For all other objects, we'll let *validDo*'s normal code take over:

```
validDo(actor, obj, seqno) =
{
if (isclass(obj, Actor)) return true;
else return obj.isReachable(actor);
}
```

Let's put this all together to define the call verb:

```
callVerb: deepverb
      sdesc = "call"
verb = 'call' 'phone'
doAction = 'Call'
validDoList(actor, prep, iobj) = nil
validDo(actor, obj, seqno) =
{
      if (isclass(obj, Actor)) return true;
      else return obj.isReachable(actor);
}
;
```

Two things to note here: the arguments used in the *validDoList* property are really not important in this instance, since we are setting the entire thing to nil, but in other instances you may want to pay attention to them. Also, I am using the built-in TADS function *isclass* to determine the class of the object within *validDo*. This function returns true if the object is of the class I specify - in this case *Actor*.

Now if you place this within your game, and attempt to "call" Erin or other objects, you'll see a bunch of messages saying "I don't know how to call *object name*." That's because we've told TADS that we want to call things, but not what to do when we attempt to call them. For this, we have to define two other methods: *verDoCall* and *doCall*. Why are they called this? Well, the first one, *verDoCall*, verifies (the *ver* part) the call verb when used on direct objects (the *Do* part). And we told TADS that we wanted to refer to this as "Call" (note the capitalization) when we defined the call verb. This was the second function of the *doAction* property of that verb. The second method, *doCall*, tells TADS what action to carry out when we call a direct object. These methods have to be defined for each and every object in the game that you wish the player to be able to call. That is a lot of objects (in a large game), but we can take a short cut.

There is a default object which is a catch-all object for ALL other objects. It is called *thing*. Any changes made to this object will affect every other object in the game. Sounds perfect for setting up our new call verb, doesn't it? To change this object's definition, we will use TADS modify structure to change only portions of the object:

```
modify thing
    verDoCall(actor) =
    {
       "That isn't something you can call. ";
    }
;
```

When you define a verification routine, you display a response when the verification fails. This both gives a message to the player, and let's TADS know that the verification routine has failed and it should not proceed onwards to the direct object action method. You can use if-conditions and other test to determine pass-fail, but in our case we want every case to fail and display a message. So, if the player attempts to call his desk, his chair, himself, or any other object - he will see the above message.

Note: Never ever modify anything within the verification method. Only test conditions and display messages. There is a long and complicated reason for this, but the simplest explanation is that if you do weird and odd things will start happening. So - don't do it!

Now, we don't want the above response to display for every case - we want the player to be allowed to call Erin. (That's why we are defining this call verb, remember?) To allow for this, we'll override the verification method for the Erin object. There are several things we need to check: is Erin in the same room? Is the player in the hallway? Is Erin in her office? Is he in Erin's office? These last two we'll check by testing the visibility of Erin's desk (chosen at random – it could have been any object within her office). We could just check the player and/or Erin's location, but either could be in her office, in her chair, on her desk, etc. So, just checking the location would not be good enough. Okay - let's write the verification:

```
verDoCall(actor) =
{
if (Erin.isVisible(actor))
   "Why call her? She's right here! ";
else if (actor.location = Hallway)
   "There isn't a phone nearby. ";
else if (Erinsdesk.isVisible(actor))
   "You attempt to call Erin from her office. Hmm..her phone is busy! ";
else if (not Erinsdesk.isVisible(Erin))
   "You attempt to call Erin. Her phone rings and rings, but no one answers.
";
}
```

Lastly, let's define what will happen when the player does call Erin. We do this by using the action method defined for the call verb: *doCall*. When we do, Erin will answer. We'll ask her to leave and move her out of her office. We'll also unlock and open her door to allow the player to enter. And lastly, we'll move Erin out of her office. In a larger game, we would probably move her into whatever location the copier can be found. In our small game, we will simply move her out of the game environment, which we do by moving her to nil:

```
doCall(actor) =
{
"You call Erin's office, and she picks up the phone. \"Hello?\"
\b\"Hi Erin, it's me.  I know you're working on the Anderson file,
but I need to see the expense report.  Could you make a copy for me?\" you
ask.
It's just an excuse to get her out of her office so you can look for the
pictures, but you hope it works.
      \b\"Oh, sure,\" she says. \"Just a moment.\"
\bYou hear her office door open, and Erin walking down the hallway. ";
Halldoor.isopen := true;
Halldoor.islocked := nil;
Erinsdoor.isopen := true;
Erinsdoor.islocked := nil;
Erin.moveInto(nil);
}
```

You'll notice that when we manually unlock and open Erin's door, we have to do both "sides" - the one in Erin's office and the one in the hallway. In game, if the door is opened and/or unlocked, the other side is automatically adjusted. We only have to do it this way because we are doing it manually. Let's see how this looks when we add it to the Erin object we defined last month:

```
Erin: Actor
     location = ErinsOffice
     sdesc = "Erin"
     adesc = "Erin"
     thedesc = "Erin"
     ldesc = "Erin is your lovely coworker. Her slender body and long legs
        have been the object of many of your fantasies. "
     noun = 'Erin'
     adjective = 'lovely' 'young' 'coworker'
     isHer = true
     verDoCall(actor) =
     {
    if (Erin.isVisible(actor))
        "Why call her? She's right here! ";
    else if (actor.location = Hallway)
        "There isn't a phone nearby. ";
    else if (Erinsdesk.isVisible(actor))
        "You attempt to call Erin from her office. Hmm..her phone is busy! ";
    else if (not Erinsdesk.isVisible(Erin))
        "You attempt to call Erin. Her phone rings and rings, but no one an-
swers. ";
     }
     doCall(actor) =
```

```
        {
      "You call Erin's office, and she picks up the phone. \"Hello?\"
      \b\"Hi Erin, it's me.  I know you're working on the Anderson file,
      but I need to see the expense report.  Could you make a copy for me?\" you
ask.
      It's just an excuse to get her out of her office so you can look for the
      pictures, but you hope it works.
            \b\"Oh, sure,\" she says. \"Just a moment.\"
      \bYou hear her office door open, and Erin walking down the hallway. ";
      Halldoor.isopen := true;
      Halldoor.islocked := nil;
      Erinsdoor.isopen := true;
      Erinsdoor.islocked := nil;
            Erin.moveInto(nil);
        }
;
```

All of the above assumes that there are actually phones in your and Erin's office. And, that there are also some pictures to be found. As homework, try creating these objects yourself. Give the pictures object a property called *isKnown*, and set it to true after hearing Erin's conversation (hint: modify the *firstseen* property into a method). Lastly, change the verification for calling Erin so that calling her before you overhear her conversation gives a different result from calling her afterwards. Hint: Use the *isKnown* property of the pictures to affect this.

ADRIFT Segment by BBBen

I've had a look at Knight Errant's entry for this month, and I've found it a little hard to do a direct transfer of the specifics he's doing into the ADRIFT platform; it seems we've hit a point where the platforms are seriously diverging, so this month I'll try and tackle "tasks" and "events" in brief, as KE is talking about more complex interactions and tasks and events are the way you do most of them. I will not try to address all the details of what KE is doing, because the platforms are very different. I don't really *need* to explain all the specifics, but I do need to give you a run-down of the basics.

A task is basically what ADRIFT does in response to the player typing a specific command. For instance, if you type "kiss betty" and the game gives you a response, that's a task that you've just triggered. Tasks can do other things too – changing the score, or variables, moving characters and objects, and several other things. A lot of the time when you want to do something in ADRIFT the 'brute force' approach is just to use tasks to override ADRIFT's default functions. Tasks will override most anything else, so if you want a result for "open door" that has nothing to do with what you can do with your "door" object, then you will probably be resorting to a task. Tasks are not all that difficult to master, and you will be using them a lot. Out of all the columns (rooms, objects, tasks, events and characters) "tasks" will be the one that is by far the most full.

Now, if KE's entry on the TADS way to do things makes your head spin in the way it does mine, then ADRIFT is probably for you – doing things with tasks is easy. In the ADRIFT generator just click on the "Add task" button and up will pop a little box. The boxes are self-explanatory, so I won't explain everything. The "Restrictions" tab deals with everything that might prevent the task from triggering when the correct command is typed, and what will happen instead. The "Actions" tab is for defining any results of the task apart from the text you want to display (like moving objects, changing variables, etc.). The "Reversible" tab deals with whether the task can be undone by another command (I tend not to use this function, preferring to use variables instead) and also whether the task can be repeated – which is important. Finally the "hints" tab is, obviously, if you want to put hints in, although I wouldn't really advise using that function. ADRIFT 5.0 is apparently going to use a new hints system, so then it might be more useful.

I'll address the basics of "events" here too, since one of the things KE's article deals with (overhearing the phone call) would be best dealt with through an event. Basically, click the "Add event" button and up will pop the box. Now, events are a little weird, and probably on the face of it they aren't all that useful, but they have one important function: they can trigger tasks. The thing that I use events for most of all is to trigger tasks, and doing so is pretty easy. Start off by making sure that the event should start "immediately", that the event should last between 1 and 1 turns (I know that's bad grammar, I'm just going by what it says in the generator) and that the "Restart this event as soon as it finishes" box is ticked.

Now, go to the "advanced" tab. There are a bunch of functions you can experiment and play with here, but the one we're interested in is the "Execute task" function down the bottom (it can also be changed to "Undo task", but we're not interested in that right now). In the drop-down box next to it you can find a list of all the tasks you have made (so it's best to have created the task you want to trigger first, though you can come back later and select the task from here – just don't forget that step). Select

whichever task you want to trigger, and now that task will try to trigger every turn!

The trick to this is to then go into the "Restrictions" tab of the task you want to trigger, and make sure that the restrictions prevent the task from going off except when you want it to. Also, setting the location for the task will be important, because if you have no other restrictions then the task will simply trigger when the player enters the appropriate room. Generally speaking you would not want this task to be "repeatable", although that's really up to what you want to do with it. For the example we're working with (in which the player overhears a phone call) we will not be making the task repeatable.

So, a quick run-down of what to do to hear that phone call:

1. Create a task, and call it something like "overhearingthephonecall" – this is a name that you will be able to recognise in the "tasks" menu, but which the player is unlikely to ever type (not that it matters all that much, but it shouldn't be something that the player would type often, like "west").
2. Make that task located in the hallway. We could put on other restrictions, but we won't do so today.
3. Now create an event. You could call it "Overhearing the phone call". The player can't trigger this by typing anything, however, so the name is just for your reference.
4. In "When should event start" select "Immediately".
5. In "Event should last between" make sure the values are "between 1 and 1 turns".
6. Tick the box next to "Restart this event as soon as it finishes."
7. Go to the "Advanced" tab and in the drop-down box at the bottom (next to "Execute task") find the task called "overhearingthephonecall" and select it.

And you're done. That task will now trigger when the player enters that room for the first time, and will not trigger again. Now go into the "overhearingthephonecall" task and in the "Message upon completion" box, enter the text:

```
<br><br>As you enter the hall, you hear Erin's voice through the door.  Some-
thing about the tone of her voice catches your interest, so you step closer to
the door.<br><br>"Yeah, I'm looking at the pictures right now, I can't believe
you talked me into letting you take them!  That was so hot."<br><br>There's a
brief pause before she continues, she must be talking on the phone.  "Oh,
you're naughty!  I can't now, though, there's a coworker just across the hall,
and I can't go somewhere more private without some kind of ex-
cuse ..."<br><br>The rest of the conversation is too low to hear.
```

That's pretty much it for now; when the player enters the hallway for the first time, that message will pop up after the room description. It's worth mentioning again at this point that with all the little prompts within the ADRIFT generator, the way to discover the basics of ADRIFT is just to open the generator program and poke around in it for about ten minutes. After that you will be ready to start making basic games, and the more advanced stuff will come in time.

Inform 6 Segment by 'trix

From hereon in the tutorial I'll be relying on a lib file that parses genitive expressions. These are easy to write (if you're me), and if anyone requests it, I'll post a suitable one on the Yahoo group. I'll also post a file with the code to exclude body parts from inventories. The way genitive libs generally work is that objects have a property owner, which, if set, allows for parsing possessive expressions with that specific owner's name or genitive pronoun. NPCs need to provide a genitive property, which holds an array of words that are genitive versions of their names. So Erin, from the last instalment of PE, should have the property

```
    genitive 'woman^s' 'girl^s' 'erin^s',
```

added to her definition.
With that done, we can write the bare bones of a BodyPart class.

```
    class  BodyPart
      with owner [; return parent(self); ],
           before
           [ p;
```

```
            Take, Remove: p = self.owner();
              print_ret (CTheyreOrThats) self, " part of ",
               (theNameS) p, " body.";
            Drop, Insert, PutOn: p = self.owner();
              if (p == player)
                 print_ret (CTheyreOrThats) self, " part of ",
                 (theNameS) p, " body.";
        ],
    has  scenery;
```

In the above code, two printing rules are used: CTheyreOrThats is defined in the standard library, and prints "They're" or "That's" (or "He's" etc.) depending on which is appropriate to the object. The printing rule theNameS is not in the standard library, but it's easy to write.

```
[ theNameS obj;
   if (obj == player) print "your";
   else print (the) obj,"'s";
];
```

[NB. There is a subtle problem with this printing-rule, but it won't become an issue in this little game: it will always put 's at the end of a name, but sometimes just an apostrophe is more appropriate. The general fix for this is to define a string property on objects so they can override the default behaviour — left as an exercise (hee hee).]

OK, next on the agenda is hearing Erin in her office when the player is in the hallway outside.

A good way to do that in this case is to use each_turn. That property always holds a routine, and any object that provides each_turn will have it run every turn that it is in scope.

So stick this code in the Hallway object.

```
...
each_turn
[;
   if (self has general) return;
   print "As you enter the hall, you hear Erin's voice through
      the door. Something about the tone of her voice catches
      your interest, so you step closer to the door.
      ^~Yeah, I'm looking at the pictures right now. I can't
      believe you talked me into letting you take them!
      That was so hot.~ There is a brief pause before she continues;
      she must be talking on the phone. ~Oh, you're naughty! I
      can't now, though - there's a coworker just across the hall,
      and I can't go somewhere more private without some kind of
      excuse ...~^The rest of the conversation is too quiet to hear.";
   give self general;
   score = score + 1;
],
   ...
```

In the printed string, the tilde (~) means a quote-mark, and the caret (^) means a new-line. The global variable `score` is the player's current score, and a message along the lines of "(Your score has increased by 1 point)" is automatically printed at the end of the turn if the score has changed. The line `give self general`; sets the attribute `general` on the hallway, so that we can check if the routine has already been run instead of running it every turn.

It is fairly straightforward to organise your scoring by tasks, so that the player can type "FULL" and get a list of how they achieved their score. But there are a number of steps involved, which you can look up in the DM4: look for `PrintTaskName` in the index.

Next we're going to write the code for a new action: calling someone on the phone. This is what I6 grammar looks like:

```
Verb 'call' 'phone' 'telephone'
   * creature       -> Call;
```

This allows commands "call erin" or "phone young woman" etc. to be parsed as referring to the `##Call` action.
That grammar, however, would only allow the player to call someone currently in scope. So instead, use this:

```
Verb 'call' 'phone' 'telephone'
    * scope=CallScope      -> Call;

[ CallScope;
   switch (scope_stage)
   {
     1: rfalse;
     2: PlaceInScope(Erin); rtrue;
     3: "You can't make that call.";
   }
];
```

The grammar-scope routine `CallScope` returns false at scope_stage 1 (which means "Don't accept multiple items"); places Erin and nothing else in scope at scope_stage 2; and at scope_stage 3 (if the player tried to call something uncallable), it will print "You can't make that call.".

The grammar should now parse "Call Erin" into an attempt to perform the action `##Call`. That action only exists if we write an action routine (which must be called `CallSub`), which determines what happens when the action is executed.

```
[ CallSub;
   if (TestScope(yourPhone)==false)
       "Your phone isn't here.";
   if (RunLife(noun, ##Call)) return;
   "There is no answer.";
];
```

The routine above checks if the player-character is near his phone (which is needed to make a call), then calls RunLife (so the NPC receiving the phone call can intercept the action and provide a suitable response), and prints "There is no answer." if nothing intercepts the action before that point.

Life is a property provided by NPCs to define NPC-like responses, such as being spoken to, being attacked, and being given objects.

Stick the yourPhone object in the PC's office.

```
Object -> yourPhone "phone"
  with name 'phone' 'telephone',
       article "your",
       owner [; return player; ],
       before
       [;
         Take, Remove:
           "It would do no good to unplug the phone and take it
            somewhere else.";
       ],
  has  static;
```

Now to intercept the case we want: when the player calls Erin.

Put this code in the definition of Erin.

```
  life
  [;
    Call:
      if (hallway hasnt general)
         "You have nothing to talk to Erin about right now.";
```

```
        if (erin notin erinOffice)
            "The phone rings, but there's no answer. She
             must have left her office.";
        print "You call Erin's office. She picks up the phone.
          ^~Hello?~
          ^Hi, Erin, it's me. I know you're working on the Anderson
          file, but I need to see the expense report. Could you
          make a copy for me?~ you ask. It's just an excuse to get her
          out of her office so you can look for the pictures,
          but you hope it works.
          ~Oh, sure,~ she says. ~Just a moment.~^";
        give erinOfficeDoor open ~locked;
        remove erin;
        rtrue;
    ],
```

This is mainly stuff already seen: give commands set the attributes for the specified object (in this case making the office door open and unlocked). The `remove` command simply moves Erin from where she is, to nowhere.

Next month, apparently, interacting with Erin directly. What larks.

Inform 7 Segment by Purple Dragon

Well, we now have a basic map and the player can move around and examine a few things in a couple of the rooms. Still, it is certainly not a very exciting game at the moment. For one thing, there are no characters to interact with. Yes, we have created Erin, but at the moment the player has no way of knowing that she is even there. He will certainly assume that she is and also probably guess that she is in her locked office, but he has no way of telling for sure and no way to get to her.

We can easily handle letting the player know that she is in her office, and we'll do it by allowing him to overhear her talking on the phone. The simplest way (although far from the only way) is to show the conversation the first time the player enters the hallway between Erin's office and his own. If all we wanted to do was to print the conversation, then it is very easy to do by just adjusting the description of the hallway.

```
Hallway is north of your office.  The description of hallway is "To the south
is your office and to the north is Erin's office.  The hallway also leads fur-
ther east to other offices.[if unvisited][paragraph break]Erin's telephone
conversation.[end if]"
```

One of the many things that Inform keeps track of is whether or not the player has visited a certain room. So in this case, if the hallway is unvisited (ie: this is the first time the player is entering it) then the phone conversation is printed, otherwise, only the regular room description is. Note the square brackets in the description. Square brackets within double quoted text are used for either text substitutions or for conditions. [if unvisited] is a condition that checks if something is true before moving forward and it must have an [end if] or [otherwise] at the end to let Inform know when the conditional text stops. Here it is at the end but that is not always the case. [paragraph break] is a text substitution. Actually, in this case the "text" is just a blank line but it's important since it would look kind of funny without that.

I said above that we could do it this way if all we wanted to do was print the added text, but in this case we also want to give the player a point for the momentous accomplishment of finding his way out of his office so we'll do it a bit differently. Again, there are many ways to handle this and we could keep the above text and just give the point a different way, but the following method will keep it all together and is the way I prefer to handle cases like this.

```
Hallway is north of your office.  The description of hallway is "[Hallway De-
scription]."

To say Hallway Description:
say "To the south is your office and to the north is Erin's office.  The hall-
way also leads further east to other offices";
      If Hallway is unvisited begin;
            say paragraph break;
say "As you enter the hall, you hear Erin's voice through the door.  Something
about the tone of her voice catches your interest, so you step closer to the
```

```
door.

'Yeah, I'm looking at the pictures right now, I can't believe you talked me
into letting you take them!  That was so hot.'  There's a brief pause before
she continues, she must be talking on the phone.  'Oh, you're naughty!  I
can't now, though, there's a coworker just across the hall, and I can't go
somewhere more private without some kind of excuse ...' The rest of the con-
versation is too low to hear";
Increase score by 1;
Now erin's pictures are known;
      end if.

A thing can be known or unknown.

Erin's pictures are a thing.
```

First we replace the entire description of the hallway with a text substitution. You can call this anything you want so if you want to call it "Dave" or "The meaning of life" or "THX1138" then feel free. Just keep in mind that you are probably going to want to know what the hell is going on later so it's usually best to keep it simple. We've said where the text substitution happens, now we just have to say what it does.

In a previous month I gave the basic layout of a rule and you may recognize the format above. Actually, this is not a rule at all but something that Inform calls a "phrase." However, since the format is almost identical I won't pause here to talk too much about the differences. From a practical point of view, it's works the same way except for what begins it. In this case we start with the phrase "To say Hallway Description:" (note that it ends with a colon). Just as is the case with rules, all further statements will end with semi-colons until the last one, which will end with a period.

The first say command will be printed every time the program reaches the text substitution "Hallway Description" but the next section we only want printed the first time they enter the hallway. To do this we simply test to see if the hallway has been visited before and if not, we print the phone conversation, increase the score, and tell inform that the player now knows about the pictures. Of course, this means that there has to be some pictures for the player to know about so we create them along with a way to track if they are known or not. Right now there is no description and no location for the pictures so they start the game out of play, but that doesn't matter for our purposes here. When we figure out where and what the picture are we can add it then.

The nice thing about handling the phone conversation this way is that it is easy to add to and change. Lets say that later on in the game we want the player to enter the hallway, notice that Erin's door is cracked a bit, and we want to tell him what's going on inside (for example). We just need to add another of the "If ... begin ... end if" statements like the one above and we can add in whatever we want without having to define a whole new text substitution.

You may have noticed the formatting above with the indentions and such and be wondering how precise this has to be. Well, strictly speaking, it is not necessary at all. In fact, the following will work exactly the say way.

```
To say Hallway Description: say "To the south is your office and to the north
is Erin's office.  The hallway also leads further east to other offices"; If
Hallway is unvisited begin; say paragraph break; say "As you enter the hall,
you hear Erin's voice through the door.  Something about the tone of her voice
catches your interest, so you step closer to the door. 'Yeah, I'm looking at
the pictures right now, I can't believe you talked me into letting you take
them!  That was so hot.'  There's a brief pause before she continues, she must
be talking on the phone.  'Oh, you're naughty!  I can't now, though, there's a
coworker just across the hall, and I can't go somewhere more private without
some kind of excuse ...' The rest of the conversation is too low to hear"; In-
crease score by 1; Now erin's pictures are known; end if.
```

I'm sure you'll agree that the first version is quite a bit easier to read than this one but from a programming point of view, Inform doesn't care which method you use so do whatever works for you.

It sounds like Erin is just itching for an excuse to get out of there so lets give her one. We will set things up so that the player can call Erin, ask her for something, and give her an excuse to leave her office. Inform already comes with many known verbs that the player can use to do things, but unfortunately, calling isn't one of them. That's ok, creating our own actions is usually pretty simple.

```
Calling is an action applying to one visible thing.  Understand "call
[someone]" or "phone [someone]" as calling.
```

Actions can apply to one thing, two things, or nothing. Here we make "calling" apply to one thing, so the command will be "call erin" and not something like "call erin on phone." I'll explain why we added the word "visible" in a moment. The second sentence above tells Inform what the player must type to actually get the command to work. Note that we use the text substitution [someone] meaning that the command will only work with people. We could have use [something] instead, which would also have allowed the command to include a person, but would have allowed the player to try calling any inanimate object in the game as well. Since that isn't what we want and we would just have to write extra code to not allow the player to call the desk or chair, it's easier to just stop it here.

```
After deciding the scope of the player while calling:
      place erin in scope.
```

The problem with the calling action (or any action) is that inform will usually only allow the player to specify a person or object in a command if that person or object is actually present in the same location. Inform (and most other languages) calls this the scope of the player. Something that Inform does every single turn is to decide what is and is not in scope. So the above text tells Inform that when the action that the player is attempting is "calling" to go ahead and decide what is in scope like normal, but then add Erin to the list. Putting Erin in scope like this makes her visible to the player, but not touchable, which is the reason for the word "visible" when we created the action. Otherwise Inform will insist that the player be able to touch the item or person in question. Note that this all happen over the course of a single turn. The player types the command, Erin is brought into scope, the appropriate text is printed out, and that's it. If the player, on the very next turn, tried to examine Erin he would meet with an error, because she would no longer be in scope.

Whenever you create a new action Inform in turn creates three corresponding rulebooks for that action call "check," "carry out," and "report." In general, check rules are where you put things that you want to intercept the action before it happens, carry out rules are where the action actually happens, and report rules tell the player what happened. There are exceptions to all of that, but go with it for now.

```
Check calling:
      If the noun is the player begin;
say "I've heard of people talking to themselves, but that's a bit ridiculous
isn't it?" instead;
      end if.

Check calling erin:
      If the location is not your office begin;
say "If you want to call someone you should use the phone in your office." in-
stead;
      otherwise if erin's pictures are unknown;
say "You call Erin and chat for a few minutes, but there's really nothing you
need from her right now." instead;
      otherwise if the location of erin is not erin's office;
say "The phone rings and rings, but there's no answer.  She must have left her
office." instead;
      end if.
```

What we are doing here should be pretty self-explanatory. We check if the player is trying to call himself, call from somewhere other than his office, call before he has heard the conversation about the pictures, or call after Erin has already left her office. And if so we say something else instead. Note that the word "instead" that ends each of the lines is important because it stops the action at that point, which means that in the case of the "Check calling Erin" rule above the player will only ever see one of the responses at a time. By the way, all of the above could have been included under the "Check calling" rule with no need for the "Check calling Erin" rule, but it would have been a bit more complicated in this case so I separated them.

If...else statements are common in most different programming languages. Above, I use the word "otherwise" instead, but "else" will work just as well if you wish to use it. I use otherwise because it seems more natural from a language point of view and that is, after all, what Inform 7 is all about.

```
Carry out calling erin:
        Now erin's door is unlocked;
        Now erin's door is open;
        Remove erin from play.
```

Ok, so our action has made it all the way through our list of checks unscathed, so now what happens. You can, of course, do pretty much anything that you want at this point but what we do is to open and unlock Erin's door and remove her from play. Think of it kind of like an actor leaving the stage. She is waiting in the wings and we can bring her back on whenever we like.

```
Report calling erin:
            say "You call Erin's office, and she picks up the phone.
            'Hello?'[paragraph break]'Hi Erin, it's me.  I know you're working
            on the Anderson file, but I need to see the expense report.  Could
            you make a copy for me?' you ask.[paragraph break]It's just an ex-
            cuse to get her out of her office so you can look for the pic-
            tures, but you hope it works.  'Oh, sure,' she says.  'Just a mo-
            ment.'"
```

This is the easiest step. Once the action is completed, we just have to say what happened. In this case we don't actually mention that the action has unlocked and opened Erin's door of course, but the player should get the idea.

And that's it. It is far from perfect but it will work for our purposes at the moment. For instance, did you notice that in the entire discussion above I made no mention of an actual telephone? That is because the way we set it up, a phone isn't actually necessary. There should obviously be a phone in the player's office and for extra credit, you can try to figure out how to set up the action so that it will only work in the presence of a phone. If telephones and phone calls to multiple people were a larger part of the game, we would certainly want to set it up like that, but with circumstances as they are, the above will work fine.

That's all for this month. Next time, the two ships passing in the night will finally collide and we'll show how to handle some direct interaction with the object of our desire.

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.