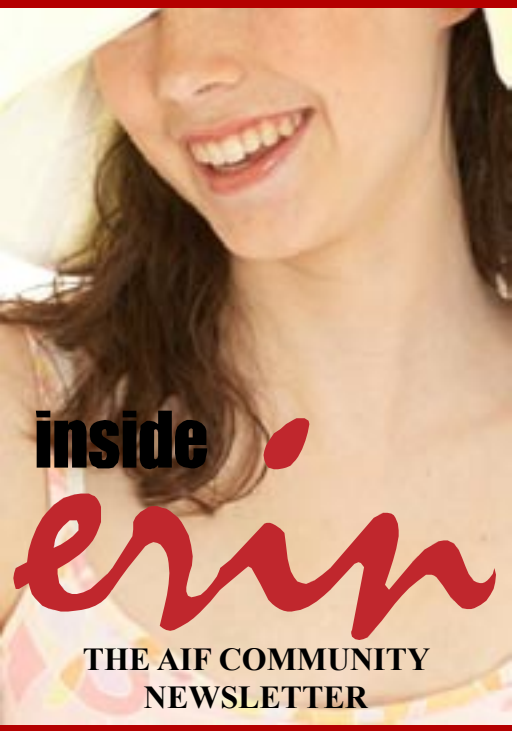
Volume 5 Number 4 April 2009



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.

Following last month's announcement of the yearly Mini-comp (see last issue for details) there arose a discussion on the forums that is becoming nearly as much of an annual tradition. It seems that every year there are questions about this game idea or that game idea, and whether it would fit into the rules or be in violation of them. This year I went so far as to write up a proposal of new rules to be included to try to clear things up. It was never my intention to actually change what the rules were calling for, but rather to more clearly spell out their intentions. Having received very little feedback on the rules addendum, I have decided not to bother changing anything this year, and the rules that appeared in last month's issue are still the official ones as they stand.

In the next year I plan on looking closely at the current rules to decide what, if anything, needs to be changed or added to. Someone recently suggested to me that the current rules actually allow too much freedom, and that to keep the competition "mini" we might want to do things like limit the characters to only two, and possibly only allow one sex scene. I'm not sure that we need go this far, but I do see his point. At any rate, I haven't even begun to make decisions on anything yet, and of course, I may well not be in any position to make them by the time the next comp come around (It all depends on if the men in the white coats catch up to me before then). If anyone has any thoughts on the subject I am, as always, anxious to hear them. For now, get to work on those games. After all, if we don't get any games for the comp, then how the rules are worded doesn't make a lot of difference does it? ◆

Top of the news this month would be the temporary closure of AIFGames.com. Here's a quote from Sexton, who runs the site:

```
Yes. I noticed some unusual mail activity and one of the scripts was being exploited to send out spam though a web console. I shut the site and removed that folder. There is an update and fix but it will be sometime later this week, possibly next weekend, before I can address this and test it. Sorry.
```

Hopefully it'll all be worked out soon and we'll be able to get back to it.

This month I started a new project on AIFGames.com – the AIF Tropes forum (though Purple Dragon actually set up the forum for me). Anyway, in this forum the community is now attempting to categorise, describe and give examples of common tropes in AIF, much like they do on tvtropes.org, which is a lot of fun and well worth checking out. AIF itself is replete with tropes, many of them unique to the format, and so it's rich territory to explore. What's more, I've got the sense that this added activity has added to the sense of community on AIFGames.com, and the forums there are starting to mature. I'd encourage people to go on there to check out or even contribute to the AIF Tropes project, discuss AIF games, or just have off-topic conversations in the "General Discussion" forum. In many ways it's a better place for discussion than the AIF Archive.

Continued on page 3

This Month At TFGamesSite

Last month was our first birthday, so I asked our illustrious leader to pen a few words. Now TinaB been very busy, but she's managed to pass on this little message.

"TFGamesSite was started when GoldenDawn's website ended. We had a great community going there and I didn't want to lose that community. Fortunately, several members stepped up to the task, creating a yahoo group, meeting up on GoldenDawn's yahoo groups as well, but the sense of community wasn't there for me without a forum to exchange and share ideas. I had the ability to quickly put together a forum and with help from the previous moderators, TestZero, Placibogal, and WindsongBard, we got the forum together and a 4SharedSite to hold the community's games.

We have recently had a host server change, allowing us to host games directly on the forums now. We have grown our community significantly over the last year, and we have found a lot of great supporters and active members. We support all transformation gaming platforms and styles at TFGamesSite and will continue to do so.

I want to personally thank everyone at TFGamesSite for helping to make it a fun, relaxed, friendly place to hang out. I wouldn't be doing this if it wasn't fun, and thanks to the great people at the forum, I'm still having fun and will enjoy my time as the forum administrator at least until GoldenDawn is ready to start up her site again."

Hi, me again. This month has been a little weird for the board. First we had a move to a new server, which now allows game writers to post files directly to our group.

Continued on page 3

Collectively Made...

This Month at the Collective...

By TeraS

It's springtime and a Mind Controller's thoughts turn to... Well... Actually they don't change that much really...

A busy month of March for the Collective as several people have posted some new games and updated a lot of other ones!

First, updated game links:

School of Lust
http://rapidshare.com/files/215222869/School_of_Lust_V1.1.rar

Genie Gone Wild
http://www.4shared.com/file/94687346/4e4df340/genie_gone_wild.html

University Sim2
http://rapidshare.com/files/213170229/dating_sim_posted_003.rag

New Game released!

Wightwashed released a game called *The House that Jack built*.

Game Thread here:
http://hypnopics-collective.net/viewtopic.php?f=11&t=15092

Download Game here:
http://rapidshare.com/files/208984708/The_House_Jack_Built_0.3.rag

Continued on page 3

AIF,Continued from page 1

There's been terrible confusion over the mini-comp rules for this year, even though the change to the rules was actually pretty minor. Oh, well. Purple Dragon has issued clarifications now, so hopefully that will smooth everything over, and I hope nobody's been deterred from entering. It would be nice to have a decent mini-comp this year, without having to give a deadline extension as happened with last year's debacle.

There was much abuzz about GoblinBoy's new game moving close to the beta stage, and what file service should be used to host it – I won't get into the details, but they're more complex than I would have thought (apparently the pictures push the file size to over 50mb). There's also been a lot of superhero discussion on the AIF Archive, some of it on-topic, and some of it off, with regards to the new *Watchmen* movie and other Alan Moore adaptations. I actually think this off-topic talk is probably a good thing, in as much as it gets everyone together comfortably chatting (as I mentioned above).

So anyway, your task for this month is to get into the community conversation and make some posts. See you next month. ◆

TF Games,Continued from page 2

The second was a proto-flame war, it all started with a legitimate comment about certain styles of RAGS games, made in a way which provoked comments. This then mutated into a discussion about what constitutes a game before mutating again to a discussion about paying for games and group collaborations. Everyone was fairly polite and things didn't go too far before the moderator stepped in, but I guess it's weird seeing your first flamish war. It's like a car-crash, you know you shouldn't watch, but you just can't help it.

On a brighter note we had another batch of games released to the world. In this months Hypnopic cross over we had Xiriels *Genie Gone Wild* (RAGS) while a week before Guinness Day (St Patrick's Day) Badhamad bought us *Pot O Gold* (RAGS). And just sneaking in before the end was Nathsatan's *School of Lust* (RAGS).

On our forum only releases we have *Experimentation* (RAGS) by AnoymousMan, whilst Fipse said hi with *Magic Ring* (RAGS). AnontheUnknown, has decided to work his/her way through all the main systems, first was *Damaged Ruins* in RAGS, which proved to them that RAGS wasn't for them, next RPGMaker.

Finally the March contest came to an end, of which I believe I mentioned before.

Turn out was a little low, but two game were entered, the excellent Popular (RAGS) by Kimberly Rex, and Once upon a Time (RAGS), by me.

So now we have a two horse race, even if one is a thoroughbred race horse (Kimberly) and one a sturdy tenacious pony (Me). I'll let you know who won next issue.

Well here's for another year. ◆

Collective,Continued from page 2

Game news update!

Darstan is aiming for an update to the *Bodywerks* game by the end of April he hopes....

Slavemaker 2 is in Beta now!

http://rapidshare.com/files/204266800/SlaveMakerBeta.rar

You can download the game at that link, you will need to overwrite the files in Slavemaker to make this work. Doing so will remove your current copy of the game. I suggest that you copy the files you have and then add this download to that duplicate folder.

And that's all for this past month at the Collective! ◆

Dear Mortal Men and Women,

Last month I promised to bring you the first letter from Eric and Sonia, a couple for whom I'd arranged their meeting in exchange for letters from each of them describing their first sexual experience together (you can read about how I hooked them up in my last few months' letters). Today, I present the letter from Eric. He hand-wrote it in ink with hardly any corrections, so it must have been very stream-of-consciousness, which is evidenced in my faithful transcription in his sometimes hurried syntax. I hope you enjoy it as much as I do.

Dear Veronica,

As promised, here is my letter describing my and Sonia's first encounter. Before I begin, I wish to express my gratitude to you, for the help you gave us in getting us together. You made it seem so simple that I wonder that I didn't just approach her on my own, or perhaps you have some special ability to make people come together. In any case, here's what happened. I'll skip past most of the evening leading up to us jumping into bed, except to tell you that we discussed how horny it made us to think about writing these descriptions, and that you would be reading them!

Veronica, you already know how much I admired Sonia. You knew I was watching her all those train rides, and how chilly and detached she seemed. One time I even sketched her from the other side of the car, and she didn't seem to notice or react. I'm including that sketch with my letter. Well, having her standing in front of me, ready to make herself mine, I realized I had never even scratched the surface. All along she had been concealing an amazing heat that now was ready to explode out of her, and I still hadn't even touched her yet. It was all in her face. Her eyes that always seemed inwardly focused when I watched her on the train, and just normally engaged when she and I talked, well, they started searing holes into me. It was almost frightening. Like there was something so elemental inside her that she was repressing, and I was pulling the ripcord that would let it out.

I suppose I have to tell you how she looked. Her body looked magnificent. She was wearing a thin, tight, light gray turtleneck top that was just vaguely translucent, and when my eyes fell over her chest, the brightness of her bra was faintly visible through the shirt. Her breasts are just perfectly sized, definitely not too large, and she was unmistakably holding them up and out for me. I almost couldn't wait to touch them. The shirt cinched and gathered a bit at her waist, giving it a narrow look. Her skirt was a smart charcoal pencil skirt, and it made her hips swing out wide from her waist. Below the skirt long, black sheer stocking-clad legs disappeared into knee-high pointy-toed leather boots. Only her hands and face were exposed, all set off by her incredible long red hair that flowed freely over her right shoulder and down her back. Her face was gorgeous, seeming at once delicate yet purposeful. All in all, it was a smart, incredibly sexy look, really powerful.

You know Sonia's a tall woman. When she wears heels, she's taller than I am. We were standing face to face and even though we were almost eye-to-eye, I felt like she towered over me. I didn't shrink away. I tried to make myself bulky, but it still seemed l was much smaller than I am, and smaller than she is. She clearly sensed that and began taking control of the situation. She pushed me back so I sat on the edge of the bed, then she took a firm stride toward me and quickly peeled off her top and without even pausing, took off the bra as well. My eyes fell on her breasts, wonderful, firm, slightly up-pointing, with small, hard, nuggetlike nipples; my mouth began to water, but she still wasn't done. She just as nonchalantly unzipped and dropped her skirt. Her black stockings came up to her thighs and stopped and above them was a garter and suspenders, but hot damn - no panties. The sudden appearance of her slight bronze-toned bush burned my eyes. Even as I write this I'm seeing the sight of her nude body like an apparition floating below the scratches of ink on the paper.

She smiled at me and turned this way and that, showing off her curves, and said, "I can tell you like what you see." I just nodded, struck dumb. She bent over and rested her hands on my thighs. Her face was an inch from mine and her breath was hot on my mouth. Then she kissed me, her lips and tongue locked on mine, a really sexy, needy, fantastic kiss, one that consumes your mind. I didn't even notice until the kiss broke some unknown time later that my pants and boxers were bunched around my knees.

She climbed up on the bed with me and sat on my lap. I held her body and kept myself sitting up, buried my face in her cleavage, kissing the warm, moist flesh between her tits. She ground her clit against the shaft of my cock, and I could feel how slick and wet she was already! I'm trying to relate how I felt -- the best words to describe it are amazed, astonished, and very, very eager. She kissed me again, her body hunched a bit, and her cascade of hair fell over our faces, creating a little curtain through which we were kissing, like I was Pyramis. Her hand fell into my lap and she pressed my penis into the flesh of her belly, getting a little more leverage to grind it against her clit, at least for a minute, and then she sank down with me inside her. I noticed the change in her expression before I noticed the change in sensation. She looked like she was totally concentrating and focused, her eyes boring into mine. Her body was still, but I could feel the muscles inside her pussy fluttering. It was really something.

She sat there for what seemed to be quite a long time. She was looking into my eyes the whole time, hardly blinking, hardly moving, just engulfing me, and watching me. I felt like there was an endless depth to her and I was only floating near the surface. At one point I shifted myself, moving myself inside her, trying to explore greater depth, but she shushed me sternly and I stopped, and did my best to sit still after that, tried to match her focus.

At last she looked up, brushed her hair back behind her head and nodded to herself, so slightly that I might have missed it, especially since I was captivated by the newly appearing tiny upcurl of the corners of her mouth. Then, rather serious-like, she got up and dismounted me quickly. I didn't know quite what to make of what she was doing, I'll admit, but I was enjoying myself enough not to question things.

After that, she sat next to me and we just kissed for a while. Maybe I pierced a little tender spot when I was inside her, because it seemed like the kissing was more loving and less desperate. I caressed her body, feeling her curves, tracing my fingers over her waistline, learning how she felt in my arms, and she held my face lightly in her long, expressive fingers. She turned my head to one side and kissed my ear a couple of times, then whispered, "you know how to give head, right?" "Mmm hmm," I said. My tongue was suddenly too swollen in my mouth for me to trust myself to talk. She stood up and took me by the hand. She led me over to a stuffed chair and sat down in it, with her ass scootched to the edge and her long legs spread very wide. It was obvious what I was supposed to do. I quickly and rather awkwardly got rid of my pants and shirt and kneeled down in front of her vagina.

I guess I'll leave it to her to describe to you how it was for me to give her head. All I know was that I really enjoyed the taste of her, especially that shocking first punch of aroma and flavor. That made me nearly swoon in a flash of intoxication. She tasted like a rich, condensed, vaguely spicy paprika, or maybe a blend of exotic spices. I wanted to bury my face inside her and find where that taste was coming from.

I definitely I felt like I was servicing her, and that she had arranged it for me to feel that way. I was down between her legs, which were held wide. Haughtily. Like I clearly should be happy to be there. Maybe that was the point, rather than for me to get her off. But I was happy to be there. Heck, I was out of my mind with happiness. And why shouldn't I be? Eating out the woman I'd lusted after for ages, tasting and touching her in the most intimate way imaginable. Trying to make her feel good. If she hadn't had other things she wanted us to do I could be there still.

I think after this, her eagerness may have gotten the better of her, because she led me back to the bed and had me lie down on my back. She climbed on top of me and straddled my waist. I looked up at her. Her face is wonderfully expressive, she can communicate so much with just a curl of her lips or a flash of her eyes. Her lips' shape now communicated her raw need. I held her thighs as she lowered herself onto me and started fucking me. The concentration and focus she showed before were lost in abandon. Her mouth was open and she was making fantastic moaning noises. I was enraptured, watching her fuck me, watching the harsh control that she always keeps over herself give way in the heat. She fell on top of me, mashing my body with hers, gyrating her hips in a rocking, back-forth motion, dragging her clit over my belly while she fucked me. I listened to her hoarse breathing and her moans muffled by the pillow. It felt incredible. I started to come very quickly, and when I came, so did she, her muscles all convulsing at once, her pussy clenching my cock. I think at first she wanted to draw it out, give us more to write to you about. But as it turned out, we both just had this incredible need to get off, and it couldn't wait any longer.

When we'd calmed down from our explosive mutual orgasm, she stayed on top of me, and we just kissed and made out lovingly. My cock stayed pretty hard, and she kept me inside her. Occasionally, one or the other of us would shudder from a post-orgasmic sensory flutter; I'd feel her squeeze my cock, or she'd feel me move inside her. We talked quietly about how we planned to write about this, and I told her we should do it right that minute, while it was fresh in our minds. We both got up, and still naked, sat down to do just that. She pulled out her computer and opened it on her lap; she gave me a pad of paper to scribble my thoughts.

Veronica, it is now the next day and I'm picking this up to tell you a little of what happened after that. We each spent about forty-

five minutes writing. We didn't talk to one another, but we did exchange ogling looks. I can't ever keep my eyes off her, and it was hard to concentrate, but I managed to write what you have now just read. We both really wanted to finish our letters, and writing them was like having another round of foreplay. We made love again, and it was a lot more balanced. We shared the control, we explored our sexual desires in a more deliberate way. We did it over and over that night. We were making love when the first light of morning came in her bedroom window. I can't even remember how many times we did it, the whole night blended into a spectacular sex-soaked gestalt.

Again, I owe you so much, and it feels like handing this letter to you today is going to seem an amazingly tiny gesture. It barely begins to express my gratitude. I truly hope it gives you what you need it to.

With love and many thanks,
Eric

***

Wow. Eric did an amazing job. And his sketch is quite good, I think. I obviously know what she looks like and I think he really captured her essence. I'm including it here along with his letter. Next month I'll relate Sonia's letter; it is just as much fun to read.

Until then, I wish you all wonderful love,

Aphrodite

A Night with Dani and Liz

Review by ExLibris

Game Info:	A Night with Dani and Liz
Author:	Rip_CPU
Release Date:	Jan 31, 2009
Platform:	Inform (zblorb)
Size:	370k
Content:	mf, mff
Type:	ANW
Length:	Short
Reviewed:	March 2009
Extras:	None

Basic Plot/Story

The plot is pretty straightforward. You are visiting Dani, your old friend (and old girlfriend). After a night on the town you've gone back to her place for a little bit of a booty call. However, Dani thinks that it would be a good idea to spice things up by getting you to have sex with her girlfriend Liz as well. Since Liz is (a) not happy about her girlfriend having sex with someone else, and (b) not particularly interested in sex with men, you have a certain amount of resistance to overcome.

Overall Thoughts

A Night with Dani and Liz was the only game completed for the abortive threesome competition. So whatever else I find to say about it, it deserves mucho bonus points just for that. On top of that, it's actually pretty good.

Puzzles/Game play

Most of the game is driven by interaction with the NPCs, but there is one straight out puzzle. Unfortunately this puzzle has a couple of weaknesses. As far as I can tell there's no way of knowing what drink Liz would like without having Dani tell you, and unless you're a drinker you'll probably have to resort to Mr Internet in order to find out how to make it. Or you could just refer to the walkthrough in the readme.

Sex

Let's focus on the good first. There is truly a colossal amount of sex in this game. By my estimate, a threesome scene is probably about three times as much work as a simple two person scene. That means there is the equivalent of six sex scenes crammed into this game. Moreover, it's all pretty well written, which given the volume is equally impressive. Having just started to slog my way through a single sex scene, I have a lot of respect for the effort the author has put into this game.

Let's focus on the bad now. I'll probably be labeled a blaspheming heretic for this, but I actually felt that there was *too* much sex in this game. It outweighed all the other elements of the game (e.g. characters, puzzles, setting, etc.) by a considerable margin. I think the game would have been enhanced by cutting out one of the two person scenes, and redirecting the effort involved into fleshing out the characters and the setting. That being said, it's actually pretty difficult to decide which scene should have been cut since they all flow together quite well and each plays a role in progressing the story.

Technical

In general the game feels pretty polished. If there were any major bugs, I didn't encounter them. There are a few minor quibbles. For example, in the shower Liz refers to you having anal sex in the kitchen whether you have or not. There's also the odd spelling mistake, and Liz gets referred to as Dani at least once. Additionally, the game is rather sparsely detailed; items mentioned in the room descriptions are often not implemented. All these issues are pretty common in games that have to be finished by a deadline, so I'm not inclined to be very harsh about them.

Intangibles

The paucity of characterisation means that the player will tend to build on whatever associations the basic characters have for them, probably in ways the author didn't intend. In my case Dani and Liz's relationship reminded me of a couple of 'open' relationships I've observed, which were open in the sense that one person did whatever (and whoever) they liked, and the other person put up with it.

So my sympathies started out heavily on Liz's side and didn't really shift through the course of the game. It's kind of hard to figure out Dani's motivations in the game, whether she's just pushing the PC at Liz to minimise her guilt over cheating on her, or if she genuinely thinks Liz would benefit from the experience. Overall, I think she comes across as being rather selfish and arrogant, which I doubt was the author's intention. In fact, playing through the game the first time I felt that a happy ending would have been Liz dumping Dani.

Final Thoughts

Overall, I think the good outweighs the bad. Being a minicomp game (and one that had to include a threesome scene at that) the author was never going to have enough time to do everything and something was going to be missed out. In this case it was characterisation. The author has said that he planned to include more conversation topics, which is something I think would have improved the game. On the other hand this game delivers good sex and a lot of it, so mission accomplished.

Rating: B

The Pizza Boy

Review by Purple Dragon

Game Info:	The Pizza Boy
Author:	Gray64
Release Date:	Jan 12th 2009
Platform:	ADRIFT 4
Size:	22KB
Content:	MF
Type:	ANW
Length:	Short
Reviewed:	April 2009
Extras:	None

Basic Story

What a night! If the raging thunderstorm weren't enough, two out of your last five deliveries turned out to be prank calls. The three that were legit tipped like it was the Roosevelt administration. The first Roosevelt administration. Oh well. No one ever became a pizza boy to get rich. One more stop and you can call it a night.

Overall Thoughts

That opening line not only sets the stage for the game, but also immediately engaged me in it. Why? Well, part of it probably has something to do with my brief stint as a pizza delivery boy way back in high school. I can identify with prank calls and nearly non-existent tipping. Part of it was also the way it was written. He could have just said that the tips from the three pizzas he delivered sucked, but adding the Roosevelt thing was a good joke (at least for us Americans) and raised my hopes for the rest of the game. Actually one thing that I really liked was the humor inserted throughout the game. It's tongue in cheek, quirky, and oh yes, funny. If you haven't tried driving away after delivering the pizza, but before going inside, try it.

Puzzles/Game Play

There is really not much in the way of puzzles in the game. There is one big one that you have to do before you can get to the good stuff, but it is so self-explanatory that it didn't work out to be much of an issue. Interestingly, this was evidently not always the case. This game was released, and when people were having a couple of problems with it the author pulled it, tested it, fixed it, and re-released it. Although I didn't actually download it until the fixed version was already out, I still tip my hat (if I were wearing one) to the author for going this extra mile to fix the game.

Sex

The sex was well written and hot with a couple of lengthy blocks of text to get your motor running. The only downside is the same problem that I had with the game as a whole, it was too short. You basically just get one run-through of the main (rub/lick/fuck tits/ass/pussy) sexual commands, and repeating any task just repeats the same block of text.

Technical

While there weren't any really bad bugs or errors, there was one thing that I found a bit annoying. In the lab there are three cylinders and three switches on the console, but there is not really any way of telling which switch operates which cylinder. It's pretty clear that Sheena just stepped out of one of them, and that her "sisters" are in the other two, but which do you open next? Opening the correct one ends the game, and there is no penalty for opening the wrong one first. Actually, the story flows better if you DO open the wrong one first. My problem was that I didn't, and then had to load the game back up and try it to see what happened. Sorry, I'm trying to discuss this without actually saying what I'm talking about so as not to spoil anything for anyone. Obviously an object in futility and it was a pretty damn minor thing anyway so just forget I said anything.

Final Thoughts

All in all, this was a good effort from a new author. Although I would have liked to have seen it a bit longer, with a bit more done along the way – and especially with the sex scene – I can certainly understand (and I completely agree) why a first time author would want to keep his first game short. What is here is well-written, funny, and engaging. If I have one MAJOR problem with the game, it's this. During all my many weeks as a pizza delivery boy, I never once had hot, sweaty sex with gorgeous sex clone created in a secret laboratory in the middle of nowhere. Some guys have all the luck.

Rating: C+

Sometimes things that we take for granted in our day to day lives turn out to be more than a bit of a challenge to translate into a game environment. Who reading this does not have a watch? Not many of you I'm wiling to bet, but how hard would it be to give the PC in our games that same basic necessity of modern life? That was the task that I gave our intrepid writers this month. One of the main reasons for this series of articles is to show the strengths and weaknesses of each of the different authoring systems under consideration. The differences between the systems have perhaps never been as clearly marked as they are this month as we take a look at which ones handle this task fairly easily, and which make you jump through more than a few hoops. I threw the staff a bit of a curveball with this one, and they have once again earned my admiration in that every one of them stepped up and made contact, if not completely smashing it out of the park.

Create a watch for the player. Wrist watch, pocket watch, pen watch, whatever, just something that he can wear or carry around with him. The player should be able to look at it at any point in the game to see what the game time is. One turn should equal one minute of time, and just so that everyone is at the same starting point, the game begins at 12:00PM (noon). In addition to being able to show the current time, the watch should beep every hour on the hour, and there should also be a way to set an alarm at a certain time, and have it go off at that time.

TADS 2 Segment by A. Bomire

This month in "Coder's Corner", we will be exploring time. Not in the "Set the WABAC machine for 1776, Sherman" sort of way, but in the way of keeping time within a game. In our example, the player will be equipped with a wrist-watch. To keep it simple, it won't be the ultra-modern 1001 function watch that you might pick up today. Our watch will tell the current time (based upon 1 minute per turn), beep upon the hour, and have an alarm that the player can set.

To set the stage, we won't require an elaborate game environment. I'll start out with a simple waiting room – perhaps for a doctor's office. In a real game, this environment would be equipped with chairs and tables, and some really old magazines. But, we won't need all of that. The focus of this sample will be something the player is carrying/wearing – his watch. So, a really simple room will do:

```
startroom: room
     sdesc = "Waiting Room"
     ldesc = "You are in the waiting room of the doctor's office, waiting for
              the doctor to see you. "
;
```

Simple enough. Now for our watch. Our watch will display the current time, which we will keep track of in three properties of the watch: hours, minutes and AMPM. We are nothing if not clever in our naming techniques! Examining the watch will display these properties in the usual manner: hours and minutes separated by a colon followed by an AM/PM indicator. To do so, we will make use of a TADS embedded expression, which uses angular brackets to enclose, evaluate and display TADS information (for more information on embedded expressions, please consult the TADS manual, Chapter 8 – Language Reference):

```
watch: clothingItem
    location = Me
    sdesc = "watch"
    ldesc = "This simple watch was a gift. It only has two functions,
        showing the current time and an alarm which can be set or
        cleared using  a button on the side.
        \bThe current time is: <<self.hours>>:<<self.minutes>> <<self.AMPM>>. "
    isworn = true
    noun = 'watch'
    hours = 12
```

```
        minutes = 00
        AMPM = 'PM'
    ;
```

You'll note that the watch is an article of clothing, so that the player can "wear" it. Also, it starts out in the player's possession and with its *isworn* property set to true (so that it is "worn"). This seems simple enough, but if you try it out you'll notice something odd. The time displays as: `12:0 PM`, not as `12:00 PM` as is usual for a digital watch/clock. This is because TADS automatically truncates the double-0 to a single 0. There are a couple of ways we can solve this. One way is to split the minutes into two other properties, tens and digits, and then track the time that way. Another way is to convert the *minutes* property to a string value ('00'), which would complicate changing the time. I'm going to use TADS string functions to correctly format the minutes.

The *cvtstr* function is designed to take a numerical or logical (true/false) value and convert it into a string suitable for output. It will take a two digit number such as 01 or 09 and convert it to a single character output such as '1' or '9' (sound familiar?). This doesn't appear helpful, but the output is a string value, which we can use with other string functions and manipulate. I'm going to pair this with the length function (which tells me the length of a string). By checking to see if the length of the converted string is less than 2, I know whether I need to pad the output with an additional '0'. And what is good about TADS is that I can do all of this within a TADS embedded expression, using the conditional form:

<< conditional test ? output if true : output if false>>

Using this, our new ldesc property of our watch will look like this:

```
    ldesc = "This simple watch was a gift. It only has two functions,
            showing the current time and an alarm which can be set or
           cleared using a button on the side. \bThe current time is:
            <<self.hours>>:<<length(cvtstr(self.minutes)) < 2 ? '0' +
           cvtstr(self.minutes) : self.minutes>> <<self.AMPM>>. "
```

That may look complicated, but if you examine it carefully you'll see that I test the length of the converted string, and if it is less than 2 I pad the output with a '0', or I simply output it as is. Using TADS embedded expressions will save you a lot of coding space, but it also is one of the reasons why people often refer to TADS as "cryptic".

Next we will need to increment the time every turn. We could use the techniques we learned in "Coder's Corner #1" in using fuses and daemons to set up a "time daemon" for the watch. However, we've already explored that technique. I'm going to use a different technique here. In this case, I'm going to take advantage of a function already built into TADS – the turn counter.

TADS already increments a counter every single turn, this counter being referred to as the "turn counter". Like much of TADS, we can, if we wish, alter this function to perform other tasks as well. Before we do, however, let's see what tasks are currently carried out:

```
    turncount: function(parm)
    {
        incturn();
        global.turnsofar := global.turnsofar + 1;
        scoreStatus(global.score, global.turnsofar);
    }
```

Okay, what is going on here? Well, first it increments the game by one turn (*incturn)*, which does things like carry out any fuses, daemons, and other background scripts. Next, it adds 1 to a property of the global object called *turnsofar*. As the name implies, this property keeps track of how many turns have passed. Lastly, it updates the status line with the current score and number of turns. Nothing very fancy, but let's be sure that when we modify this that we allow these tasks to complete.

To change the time, we will be adding 1 to *watch.minutes*. If the minutes reach 60, we will set them back to zero and increment *watch.hours* by 1. If *watch.hours* reach 13, we will set them back to 1. If *watch.hours* is equal to 12, and *watch.minutes* is equal to 0, then it is time to change *watch.AMPM* to either 'AM' or 'PM', depending upon its current value. All of this can be handled in just a few code statements. Because **turncount** is a function, we will need to replace it instead of simply modifying it:

```
replace turncount: function (parm)
{
    incturn();
    global.turnsofar := global.turnsofar + 1;
    scoreStatus(global.score, global.turnsofar);

    //here are our additions
    watch.minutes := watch.minutes + 1;
    if (watch.minutes = 60)
    {
        watch.minutes := 0;
        watch.hours := watch.hours + 1;
    }
    if (watch.hours = 13)
        watch.hours := 1;
    if (watch.hours = 12 and watch.minutes = 0)
    {
        if (watch.AMPM = 'AM')
            watch.AMPM := 'PM';
        else
            watch.AMPM := 'AM';
    }
}
```

That pretty much takes care of keeping track of time. Our next task is to allow the player to view and set an alarm. As can be seen by the description of the watch, this is done using a button on the side of the watch. This button doesn't yet exist, but we can create it easily enough. The alarm will be tracked using three new properties. They can be part of the button, but it is just as easy and perhaps more logical to make them part of the watch. I'll call them *alarmHours*, *alarmMinutes*, and *alarmAMPM* to correspond to our previous properties. When the *alarmHours* and *alarmMinutes* properties are set to 00, then I will consider the alarm to not be set. Any other values will set the alarm. If the alarm is set, the player will be prompted to clear it when he presses the button. If it is not set, he will be prompted to set the alarm. Then he will be prompted to input the alarm time.

All of this will involve getting and using the player's input. There are several ways of doing this, and I am going to go over two of them. The first is having the player answer a simple YES/NO question. This common occurrence is handled using a built-in TADS function called *yorn* (for yes or no). This function waits for the player's input, and returns a 1 if he typed 'Y' (or 'y', 'yes', 'YES' or other variants), and 0 if he types 'N' (or 'n', 'no', 'No', etc.). If the player types anything else, 'X' for example, a -1 is returned. The author is expected to provide a suitable prompt prior to accepting input. Here is an example:

```
"Do you wish to enter the bedroom? ";
if (yorn() = 1)
    Me.travelTo(Bedroom);
```

That will make it simple to determine if the player wishes to set or clear the alarm. Now for getting an alarm time. For other player input, TADS provides a separate function called *input()*. This allows the player to input a line of text, terminated by pressing ENTER. This can be any text. For our purposes, we wish the player to enter an alarm time, hopefully in the form of "07:00 AM" or something similar. However, the player could enter practically anything (and knowing players, someone will enter something odd), so we would need to account for it. One way of limiting the amount of testing and checking we do is to split up the input, and have the player enter the hours, minutes and AM/PM entries separately. Something like this:

```
"Enter the hours portion of the alarm time (ex. 7,  10, 05): ";
watch.alarmHours := cvtnum(input());

"\nEnter the minutes portion of the alarm time (ex. 30, 15, 25): ";
watch.alarmMinutes := cvtnum(input());

"\nEnter whether the alarm time is AM or PM: ";
watch.alarmAMPM := input();
```

You'll no doubt notice that I use another TADS function, *cvtnum*, to convert the string input into a number which is then stored within the appropriate property. This is all very nice, but it doesn't take into account that the player could enter anything. For example, suppose the player enters 23 as an hour (for what we Americans refer to as "military time"). Or, he enters "seven" instead of the number 7? We would need to test each input to verify that it falls within the acceptable limit for each property: hours are 1-12, minutes are 0-60 and AMPM is either 'AM' or 'PM'. These are relatively simple tests, and I will address them in a bit, but first I want to look at another way of getting the player's input.

Entering the hours, minutes and AM/PM as separate entries makes it easier on the author, but it certainly is clumsy for the player. And, as a good author we want to make things easier – not harder. Wouldn't
it be nicer if the player could enter the alarm time as one entry: "7:00 AM" or "10:30 PM", for example? Well, of course the player can. The *input()* function doesn't limit the player on how much information he can type. (Actually, there probably is a character limit, probably close to 256 characters or something similar. I couldn't find any documentation on this.) The problem then becomes taking this information and separating it into its component parts, something that is called "parsing".

To do so, we would need to make use of two more TADS functions: *find* and *substr*. *Find* is a wonderfully flexible function that allows us to locate an entry within a list, but it also can be used to find a portion of a string (called a "substring") from within a larger string. For example, *find('abcdefg', 'def')* would return a value of 4. The *substr* function allows us to "pull out" a portion of a string, given that we tell it where to begin and how many characters to retrieve. For example, *substr('abcdefg', 4, 3)* would return a value of 'def'. For more information on using either of these two functions, please consult the TADS manual, Chapter 8 – Language Reference.

Using these functions, we can figure from the player's input just what the hours, minutes and AM/PM entries should be. The time entered should have a colon (:) in it, and anything before the colon should be the hours. The two characters after the colon should be the minutes. And, we can use *find* to determine if the player entered 'AM' or 'PM', or even just 'A' or 'P', within the text. Here is some code to do just that:

```
local i, alarmTime;

"Please enter the alarm time, in the form of 'HH:MM AM/PM'. For example,
to set the alarm to 7:00 AM, enter '7:00 AM'. \nEnter alarm time: ";
alarmTime := input();

i := find(alarmTime, ':');
if (i <> nil)
{
   watch.alarmHours := cvtnum(substr(alarmTime, 1, i-1));
   watch.alarmMinutes := cvtnum(substr(alarmTime, i+1, 2));
   if (find(alarmTime, 'A') )
       watch.alarmAMPM := 'AM';
   if (find(alarmTime, 'P') )
       watch.alarmAMPM := 'PM';
}
```

You'll notice that I test to be sure that a colon is found before moving on to parse the player's input. Also, I use "i+1" and "i-1" to set the starting and ending positions of my substrings so that they skip the found colon. As with our previous method of getting the player's input, we would next need to test the input to verify that it satisfies our boundary conditions: hours are 1-12, minutes are 0-60 and AM/PM is 'AM' or 'PM'.

Let's put this all together. First we'll add the alarm time properties to our watch:

```
watch: clothingItem
    ...for clarity, I'm only including additions to our previous definition
   alarmHours = 0
   alarmMinutes = 0
   alarmAMPM = 'AM'
;
```

Next, we need to make a button. I'll make it part of the watch by setting its location to be our watch. This way, the button stays with the watch if it should be removed and left behind somewhere. I will use the pre-built *buttonItem* class object so that it will automatically respond to being pushed. And I'll use the **doPush** method to handle being pushed. If it is being cleared I'll handle it within **doPush**. However, setting the alarm is more involved and for clarity I'll handle doing that in a separate method.

```
button: buttonItem
   location = watch
   sdesc = "button"
   ldesc = "This button on the side of the watch is used to set or
            clear the alarm. Pressing it will display the current alarm setting,
            and allow you to either clear it or set it to a new alarm time. "
   noun = 'button'
   doPush(actor) =
   {
        if (watch.alarmHours = 0 and watch.alarmMinutes = 0)
        {
            "The alarm is not currently set to any time. Do you wish to set it
            (Y/N)? ";
             if (yorn() = 1)
                 self.SetAlarm;
        }
        else
        {
            "The alarm is currently set to
            <<watch.alarmHours>>:<<length(cvtstr(watch.alarmMinutes)) < 2 ? '0'
+
            cvtstr(watch.alarmMinutes) : watch.alarmMinutes>> <<watch.alarmAMPM
>>.
            Do you wish to clear it (Y/N)? h;
            if (yorn() = 1)
            {
                watch.alarmHours := 0;
                watch.alarmMinutes := 0;
                watch.alarmAMPM := 'AM';
                  "The alarm has been cleared. Press the button again if you
                  wish to set the alarm to a new time. ";
            }
          }
    }
    SetAlarm =
    {
       local i, alarmTime, passFail := nil;


       while (passFail = nil)
       {
            "Please enter the alarm time, in the form of 'HH:MM AM/PM'. For
            example, to set the alarm to 7:00 AM, enter '7:00 AM'.
          \nEnter alarm time: ";
          alarmTime := input();
          passFail := true;

          i := find(alarmTime, ':');
          if (i <> nil)
          {
            watch.alarmHours := cvtnum(substr(alarmTime, 1, i-1));
```

```
                watch.alarmMinutes := cvtnum(substr(alarmTime, i+1, 2));
                 if (find(alarmTime, 'A') )
                    watch.alarmAMPM := 'AM';
                 if (find(alarmTime, 'P') )
                     watch.alarmAMPM := 'PM';
               }

               if (i = nil or
                    (watch.alarmHours < 1 or watch.alarmHours > 12) or
                      watch.alarmMinutes > 59 or
                      not (watch.alarmAMPM = 'AM' or watch.alarmAMPM = 'PM') )
               {
                      "\bThe alarm time you entered, <<alarmTime>>, was not
                    correct, either in format or value. ";
                    passFail := nil;
               }
          }
    ;
```

You'll see that I use a pass/fail variable to loop until the player enters a valid time. In an actual game, I would probably extend the above to allow the player an option to exit out without entering a time. I would also further break down my error message to inform him of exactly what was wrong with his input (the hours are wrong, or the minutes, for example). However, the above is functional in getting the player's input and parsing it into the appropriate properties.

For the last bit of our watch, I need to actually use the alarm time and beep at the player when the alarm goes off. I won't actually "beep" as in playing a sound (although TADS does allow you to do that). Instead I will simply display a message. Along with this, I will also display a message having the watch beep upon the hour. The best place to do this is when we change the watch's time. This is currently done within the *turncount* function. Extending this function to display messages to the player is simple enough:

```
    replace turncount: function(parm)
    {
         ...Again, this portion is added to the code we've already defined for this
    function
        if (watch.minutes = 0)
           "\bYour watch beeps to inform you it is the top of the hour. ";

        if (watch.alarmHours = watch.hours and
             watch.alarmMinutes = watch.minutes and
             watch.alarmAMPM = watch.AMPM )
             "\bYour watch beeps several times to inform you that your alarm is
              going off. ";
    }
```

If you test the above, you may see something you consider strange. The watch will read 12:59 PM, for example, and will beep to say that it is the top of the hour. This is correct, as we increment the time by one minute before checking it to see if we need to "beep". However, this might look strange to the player, so to correct it simply move our "beeping" code to just before we increment the time. Like this:

```
    replace turncount: function (parm)
    {
      //the original turn counter increments
        incturn();
        global.turnsofar := global.turnsofar + 1;
        scoreStatus(global.score, global.turnsofar);

      //Check the hour and the alarm on the watch
      if (watch.minutes = 0)
```

```
        "\bYour watch beeps to inform you it is the top of the hour. ";

    if (watch.alarmHours = watch.hours and
        watch.alarmMinutes = watch.minutes and
        watch.alarmAMPM = watch.AMPM )
        "\bYour watch beeps several times to inform you that your alarm is
         going off. ";

      //Increment the time
      watch.minutes := watch.minutes + 1;
      if (watch.minutes = 60)
      {
          watch.minutes := 0;
          watch.hours := watch.hours + 1;
      }
      if (watch.hours = 13)
          watch.hours := 1;
      if (watch.hours = 12 and watch.minutes = 0)
      {
          if (watch.AMPM = 'AM')
              watch.AMPM := 'PM';
          else
              watch.AMPM := 'AM';
      }
  }
```

And that should do it! You should be able to find plenty of places to make use of clocks, watches, and timepieces of all sorts (last month's bomb example is a good one). Also, getting and using the player's input comes in handy as well, along with parsing strings into their subcomponents. Enjoy playing with these new tools, and I'll see you next month!

TADS 3 Segment by Knight Errant

Before we get started this month, I'd like to apologize for the lateness of last month's *Inside Erin*. It was delayed solely because my article for this *Coder's Corner* was late. *Mia culpa*.

With that out of the way, this month we're dealing with time systems. If you'd like something as complex as the time system in *Cruise* and *Weekend* by Pierre, I highly recommend downloading and looking at Timesys.t, by Kevin Forchione. It allows for scheduled events and a day/night cycle. Versions for TADS 2 and 3 are available from the IF Archive. Since not every turn realistically takes the same amount of time, TADS 3 ships with an optional "subjective time" module, subtime.t, which provides a more imprecise modeling of time. Check it out for more details. What we'll be dealing with is more along the lines of what A. Bomire coded in "Last Minute Gift", a simple clock with one minute passing per turn. Quite frankly, if I were using this in a game, I'd use Timesys and find a way to adapt it to my needs. However, as that doesn't allow us to compare different coding methods, I have to reinvent the wheel here.

First, we begin with the property libGlobal.totalTurns. This is exactly what it sounds like, the number of turns that have elapsed since the start of the game. We need to create a method to convert this from a number of turns to an hour:minute format. Many thanks go to Fellatrix for showing me the nuts and bolts of how to do this. I'm putting the function in an object called time:

```
    time: InitObject
        // m is number of "minutes" in game.
        m()
        {
            local min=libGlobal.totalTurns;
            // Game starts at noon, so we add 12 hours worth of minutes.
            min += 60*12;
```

```
            // If we're past midnight, we only need minutes since then.
            min %= 60*24;
            min %= 60;
            return min;
        }
        turnsToTime()
        {
            local min=libGlobal.totalTurns;
            // Game starts at noon, so we add 12 hours worth of minutes.
            min += 60*12;
            // If we're past midnight, we only need minutes since then.
            min %= 60*24;
            // Get number of hours
            local h = min/60;
            min %= 60;
            // h is the hours (0-23) and m is the minutes (0-59)
            local ampm = (h < 12 ? 'am' : 'pm');
            h = (h+11) % 12 + 1;
            return "<<h>>:<<min/10>><<min%10>> <<ampm>>";
        }
    ;
```

Now that we have this function, we can set up the watch. The description is easy, all we need to do is insert a call to time.turnsToTime() in the description. The reason we have m as a separate function here is so we can run a call to m after every turn, so that we can have an hourly alarm.

```
    + watch: Wearable 'watch' 'watch'
        "It's a watch.  The displayed time is <<time.turnsToTime()>>"
        wornBy=me
        afterAction()
        {
            if(time.m==0)
                "<.p>The watch beeps.";
        }
    ;
```

That's all well and good so far, but if the PC has a watch, he may want to set an alarm (who knows why). TADS 3 includes a "set to" verb that we can utilize here. Unlike other verbs we've seen in "Programming Erin" and "Coder's Corner", Set To is an action that takes a literal statement instead of an indirect object. Instead of matching something in the PC's scope, we can attempt to Set To any random text that follows. We can utilize Set To like this:

```
      dobjFor(SetTo)
        {
            verify(){logical;}
            action()
            {
                // Get literal
                local txt = gLiteral;
                // Convert literal to string
                if (dataType(txt) == TypeObject) txt = txt.obj_;
                // Break string into hour and minute and convert to numbers
                local hour = toInteger(txt.substr(1,(txt.find(':')-1)));
                local min = toInteger(txt.substr((txt.find(':')+1)));
                local turns = hour*60+min;
                // If turns is in the past, add 12 hours.  Keep doing so until
    turns is in the future.
                while(turns < libGlobal.totalTurns)
```

```
                    {
                        turns *= (12*60);
                    }
                    // Find how many turns until the alarm goes off.
                    local turnsToAlarm = turns - libGlobal.totalTurns;
                    // Set a fuse to count down the alarm.
                    alarmID = new Fuse(self, &alarm, turnsToAlarm);
                    "Alarm set for <<hour>>:<<min<10?'0':''>><<min>>";
                }
            }
            alarm()
            {
                "<.p>Your watch alarm activates.  Beep!";
            }
        ;
```

This is a fairly complex bit of code. I've added comments in the code (the lines that begin with //) to help explain what's going on. We have to specify "verify(){logical;}" so that the watch is counted as a valid target for the Set To verb. We can access the literal statement of the SetTo command with gLiteral macro, and store it to txt. The following if-statement converts the literal to a string. After that, we have a three-layer function. Txt.substr divides the string into a sub-string based on the parameters we pass to it. One of the parameters for each run is a call to txt.find, which finds the position of the specified string. The net effect of this is that we store everything from the beginning of the string to the ':' in the variable 'hour'. Everything from the ':' to the end of the string is stored in the variable 'min'. Of course, both parts are cropped out of the string as strings, so we need to use the toInteger function to convert the strings to numbers. After that we convert the hours and minutes to just minutes and store that as turns, since we're running with one minute per turn.

Finally, since we don't know what time it is when the player sets the alarm, we compare the 'turns' variable to the number of turns elapsed in the game. We use a 'while' loop, which means as long as turns is less than the totalTurns, we keep adding 12 hours worth of minutes until we get a future time. Now that we know what turn number the alarm should activate, we can set up a fuse. We subtract libGlobal.totalTurns from turns to get how many turns in the future the alarm should activate. The fuse waits until turnsToAlarm is reached and triggers the alarm method, which displays a simple message letting the player know his alarm went off. The last bit to explain is the verification message we provide right after we set the fuse. We provide a simple message to let the player know the alarm has been set. <<hour>> inserts the value of the hour variable. The minutes are a little more complicated, we need to check if min is less than 10, so that we can provide a leading zero before the minutes.

This isn't the only way to code an alarm in TADS 3, and it may not even be the best way, but it works. I hope this has all been clear enough. See you next month!

Inform 6 Segment by 'trix

Hello, minions.

This month we're programming a watch. In addition to showing the current time in the game (which advances by one minute per turn), the watch will chime the hour, and have an alarm that can be set by the player. The setting is going to be minimal: just a room for us to stand in while we check the watch is working. Here's the starting code:

```
Constant STORY = "Watch";
Constant HEADLINE = "^An interactive watch^";

Include "Parser";
Include "VerbLib";

Object cave "Cave"
 with  description "An uninteresting cave."
 has   light;
```

```
Object -> watch "watch"
 with  name 'wrist' 'watch' 'wrist-watch' 'wristwatch' 'digital',
 has   clothing;

[ Initialise;
   location = cave;
];

Include "Grammar";
```

The watch is clothing so that the player can wear it.
Now, game-time: the standard I6 library provides a global variable `the_time`, which is used by the optional feature to show the time in the status bar. The time is stored as minutes (i.e. turns) since midnight. Our game is supposed to start at noon, which we can arrange by setting `the_time` when the game starts. Then we have to have our watch tell the player the time when it is examined.

```
Object -> watch "watch"
 with  name 'wrist' 'watch' 'wrist-watch' 'wristwatch' 'digital',
       description
       [;
           "It's a digital watch. The time is ",
             (Time12h) the_time,".";
       ],
 has  clothing;

[ Initialise;
   location = cave;
   the_time = 12*60; ! noon
];
```

The printing rule `Time12h` takes a time in minutes since midnight and prints it out as a time in 12 hour format. Since this rule doesn't exist, we have to write it.

```
[ Time12h m    f h;
   h = (m/60);
   m = m%60;
   h = h%24;
   if (h < 12) f = " am";
   else f = " pm";
   h = (h+11)%12 + 1;
   print h, (char)':', (m/10), (m%10), (string) f;
];
```

This will display times from "`12:00 am`" to "`11:59 pm`".
The `%` operator is the modulo operation. `m%60` means the remainder of `m` after you take off as many `60`s as you can.
If you wanted a 24 hour display, that can be done pretty similarly.

```
[ Time24h m    h;
   h = (m/60);
   m = m%60;
   h = (h%24);
   print h, (char)':', (m/10), (m%10);
];
```

This will display times from "`0:00`" to "`23:59`".
You should now have a working watch.

The next thing is that the watch is supposed to chime the hour. This is one of those things that happens in a particular turn but is not prompted by the player's action that turn; that means we have to use a daemon, a timer or an `each_turn` rule. A daemon would work, as long is we made sure to start it. A timer would work as long as we made sure to set the timer every hour for the appropriate delay. I'm going to use an each_turn. This has the simplifying benefit that it is only run when the watch is visible to the player, which is the only time we need it to run anyway.
Add this `each_turn` property to the watch:

```
each_turn
[;
     if ((the_time-1)%60==0)
          "The watch chimes the hour.";
],
```

The condition `((the_time-1)%60==0)` means that the time since noon (the start of the game) is an exact number of hours. The `-1` term is there because `each_turn` is run after the time is incremented, so `(the_time-1)` is the time at the player's most recent turn, which feels more logical when you're playing. You may want to leave the `-1` off if you want the time to be (for instance) 1 o'clock on the turn *after* the chime goes off instead of the turn before it.

Lastly we have the alarm. This is by far the most complicated part of the exercise, because it involves parsing a new kind of information from the player's command.
Here is the routine that parses a time from the player's command. Brace yourself: it's pretty long and complicated looking.

```
[ ParseTime    h m arr arr1;
   if (consult_words ~= 1 or 2) return NULL;
   arr = WordAddress(consult_from);
   arr1 = WordLength(consult_from);
   if (arr1 < 4 || arr1 > 7) return NULL;
   arr1 = arr + arr1;
   if (arr->1 == ':')
   {
      if (arr->0 < '0' || arr->0 > '9') return NULL;
      h = arr->0 - '0';
      arr = arr + 2;
   }
   else if (arr->2 == ':')
   {
      if (arr->0 < '0' || arr->0 > '2') return NULL;
      if (arr->1 < '0' || arr->1 > '9') return NULL;
      h = 10*(arr->0 - '0') + arr->1 - '0';
      if (h > 23) return NULL;
      arr = arr + 3;
   }
   else
   {
      return NULL;
   }
   if (arr1 - arr < 2) return NULL;
   if (arr->0 < '0' || arr->0 > '5') return NULL;
   if (arr->1 < '0' || arr->1 > '9') return NULL;
   m = 10*(arr->0 - '0') + arr->1 - '0';
   arr = arr + 2;
   if (arr1 - arr ~= 0 or 2) return NULL;
   if (arr == arr1 && consult_words == 2)
   {
      arr = WordAddress(consult_from+1);
      arr1 = arr + WordLength(consult_from+1);
   }
```

```
    else if (consult_words==2) return NULL;
    if (arr1 - arr == 2)
    {
       if (arr->1 ~= 'm' or 'M') return NULL;
       switch (arr->0)
       {
        'a', 'A': if (h == 12) h = 0;
        'p', 'P': if (h ~= 12) h = h + 12;
        default: return NULL;
       }
       arr = arr + 2;
    }
    if (arr ~= arr1) return NULL;
    if (h > 23) return NULL;
    return m + 60*h;
];
```

I won't try and explain it all, but just a couple of details: `consult_from` and `consult_words` are the index of the word we're looking for information at in the player's command and the number of words we're looking at. `WordAddress` returns a word in the player's command as an array, and `WordLength` gives the length of that array. The array has one byte per entry, and the `->` operator gives access to its elements. `NULL` is an I6 constant (holding the value `-1`), which is used to mean "no information". The routine `ParseTime` will return a time as a number of minutes since midnight if it's called after a suitable command from the player, a suitable command being one with a `topic` token. To give the player a suitable command to use, add the following verb grammar to the end of your source code (after `Include "Grammar";`):

```
Extend 'set' replace
   * noun 'to' topic                    -> SetTo
   * noun 'for' topic                   -> SetTo;
```

This will ensure that the player can use commands like "`Set the alarm for 9:30 pm`".

Now the difficult bit is suitably glossed over, we can program the alarm itself. Because setting a watch is different from setting an alarm, it suits the simulation to code the alarm as a separate object.

```
Object -> -> watch_alarm "alarm"
 with  name 'alarm',
       article "the",
       description
       [;
          if (self.number==NULL)
             "The alarm is not currently set.";
          else
             "The alarm is set for ", (Time12h) (self.number),".";
       ],
       number NULL,
       before
       [ m;
        SetTo:
          m = ParseTime();
          if (m==NULL)
             "You can only set the alarm to times
             expressed like ~8:02 pm~ or ~20:02~.";
          self.number = m;
          "You set the alarm to ", (Time12h) m,".";
       ],
 has   scenery;
```

Include that definition directly after the watch so it will be inside (or in this case, part of) the watch. Give the watch the `transparent` attribute so that the alarm will be visible to the player.
Since we have an `each_turn` in the watch already, we can use that again to code what happens when the alarm goes off. Here's the complete watch code:

```
Object -> watch "watch"
 with  name 'wrist' 'watch' 'wrist-watch' 'wristwatch' 'digital',
       description
       [;
           "It's a digital watch. The time is ",
             (Time12h) the_time,".";
       ],
       each_turn
       [;
          if (the_time-1 == watch_alarm.number)
          {
             "BEEP BEEP BEEP";
          }
          if ((the_time-1)%60==0)
             "The watch chimes the hour.";
       ],
       before
       [;
        SetTo: "The watch is already correct.";
       ],
 has   clothing transparent;
```

Putting together all those various bits should give you a working watch with a functional, settable alarm. I glossed over some bits, particularly the parsing routine, because I didn't want to get bogged down in it here, but if anyone's interested (unlikely as that may seem), feel free to ask.

Inform 7 Segment by Dudeman

Hello again everyone. Welcome once again to the Inform 7 section of this month's Coder's Corner. Today we will be creating a nifty watch for the player which can keep time, beep on the hour, and have an alarm that can be set to any time the player wants. This may sound a little complicated, but it is actually surprisingly easy to do in Inform 7 as it already tracks the time of day for you in a value conveniently called "the time of day". To show you how it works, let's try creating a watch.

```
Your bedroom is a room. "Just your typical bedroom.".

A wrist watch is a thing in your bedroom.
The description of the wrist watch is "The wrist watch is a fairly standard
looking electronic watch. Reading the time on it, you can see that it is [the
time of day].".
The wrist watch is wearable.
```

This is all the code we need for the watch to be able to tell the player what time it currently is by looking at the watch. As a default, Inform 7 games start at 9:00 AM and every turn that is taken in the game advances the time by 1 minute. The second part of this default is fine, but for uniformity we will have our game start at 12:00 PM so we just need to declare the time of day manually in our code.

```
The time of day is 12:00 PM.
```

There you have it, a watch that can tell time. However, if we want to make the watch beep every hour on the hour like many electronic watches can do, we can do that too. For this, we just need a way of checking if the time is exactly on the hour. This

can be done because Inform 7 stores the "hours" part of the time and the "minutes" part of the time in separate values that can be checked independently. Since every new hour means that the minutes value is 0, it is a perfect way to achieve this. Given this, we can write an every turn rule that will cause the watch to beep every hour when the watch is carried by the player.

```
Every turn while the wrist watch is held by the player:
let M be the minutes part of the time of day;
if M is 0, say "[bold type]*Beep*[roman type]

Your wrist watch quickly beeps to notify you that the hour has changed to [the
time of day].".
```

Now for the somewhat more complex, but still not too hard part of making an alarm that can be set to go off at any time the player sets it to. First off, lets create features of the watch that will allow us to set an alarm such as an either/or value that will determine if the alarm is set or not and a time value that will store the alarm time.

```
The wrist watch can be set or unset. The wrist watch is unset.
The wrist watch has a time called the alarm setting.
```

Now we need an action that will allow the player to set the alarm and store the values so that they can be used to set off the alarm. In this case, we will create a new action called "setting the alarm" which applies to a time and then create a set of rules which will prevent the player from setting the alarm if they aren't holding the watch and will store the given value.

```
Setting the alarm is an action applying to one time.
Understand "set alarm to [a time]" as setting the alarm.

Check setting the alarm while the player is not holding the wrist watch:
say "You don't have a time keeping device to be able to set an alarm right
now." instead.

Carry out setting the alarm:
change the alarm setting of the wrist watch  to the time understood;
now the wrist watch is set.

Report setting the alarm:
say "Fiddling with the buttons, you manage to set the alarm on the wrist watch
to [the time understood].".
```

Now all that is left is to create a rule that will check the current time and if it is equal to the time the alarm is set to it will give off a message to the player of the time and turn off the alarm.

```
Every turn while the wrist watch is set:
if the time of day is the alarm setting of the wrist watch begin;
say "[bold type]*Beep Beep Beep*[roman type]

You hear the alarm on your wrist watch go off and quickly hit the reset button
to stop it. The alarm was set to remind you that it is now [the time of day]
and you make sure to take note of that.";
now the wrist watch is unset;
end if.
```

With that our task is complete. We now have a watch which can tell time, beep on the hour, and have an alarm that can be set by the player and go off at the right time. These same basic rules can be used to create any type of clock or watch an author wants and can be altered to fit the situations of your particular game. I hope this feature has been helpful to you and as always, if you have any questions feel free to contact me personally at any time. Thanks for your time.

ADRIFT Segment by BBBen

This month's task is interesting, as it seems on the face of it to be a little challenging in ADRIFT. I would probably argue that it's not worth the trouble most of the time to make in-game watches like this, but certainly I have used timers, so let's give this one a go. The ADRIFT generator is actually reasonably well set up to handle this kind of thing provided you are comfortable with variables and events. This is an advanced tutorial, so I won't be explaining every little step or covering all the basics.

We'll make this a 24 hour watch to save some trouble. All tasks that I tell you to create in this tutorial should be able to be completed in all rooms. Oh, and you'll need at least one room to make this work, of course, as with all ADRIFT games.

Advancing one minute per turn
First up, we'll obviously need a variable (more than one, actually); call this one "minutes". Next create a task called "advancingminutes", have it increase the "minutes" variable by exactly 1, and make it repeatable. Create an event called "Advancing Minutes" and have it trigger the "advancingminutes" task every turn (I've covered this kind of thing in previous tutorials, so if you've been following them, you should be familiar with the technique). So now we've got a timer that will increase by one each turn.

Advancing the hour every sixty turns
Create another variable, this one called "hours". This integer variable should start at "12". Then create a task called "advancinghours", which will require the "minutes" variable to be 59, and will set the minutes variable to -1 (otherwise it seems to start the next hour on minute 1, rather than minute 0). This task should also increase the "hours" variable by 1, and be repeatable. Create an event, as before, to attempt to trigger this every turn (call it "Advancing Hours"). Now the hours will advance every sixty turns.

While we're at it, let's put in that hourly beep. In the description box of the "advancinghours" task, put the text "Your watch beeps. It is now %hours% hours and %minutes% minutes."

Resetting the clock every 24 hours
Create a task called "resetclock", have it require that the hours variable be at 24, and have it change the "hours" variable to 0 (the minutes are already taken care of). And of course it must be repeatable. Now create another event called "Reset Clock" to trigger the "resetclock" task. Okay, so now the clock should work, we just need to be able to read it.

Creating the watch
This bit's pretty simple, just create a dynamic, wearable object that starts the game worn by the player. In the description, type "This 24 hour watch shows %hours% hours and %minutes% minutes." Notice that the variable names are between percent signs, which will make the game show the current number of the variable. Trying to get the clock to read like a real watch is a bit more trouble than it's worth, since the minutes variable, when showing the numbers 0-9 will not read as "01, 02, 03," etc., but rather as "1, 2, 3", and I think that's too much bother to fix.

Setting the alarm
It is not too hard at all to have the alarm go off at some time that you, the author have scripted. However, it would be difficult to make it possible for the player to set the alarm themselves. In fact, I can't off the top of my head think of a way to do that (which probably means that there is a way, but it would be more trouble than it's worth). Let's say that the game has the player needing to make a call once the time has passed 3:30 PM, so the PC sets his or her watch to beep at 3:30. Create a task called "threethirtyalarm", make it require that the "hours" variable be 15 and the "minutes" variable be 30. If this is a one-time alarm then make sure it isn't repeatable. In the descriptions box put the text: "Your watch alarm beeps loudly. It is now 15 hours and 30 minutes – time to make that call." Now create an event to trigger that task in the same way as before, and call it "Three Thirty Alarm". You can make as many of these alarms as you like.

So anyway, I think that pretty much covers the brief, to an extent that should hopefully be adequate to whatever your needs are.

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 six 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.

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*.

Dudeman has released one game and is working on a second. He has also released an impressive Inform 7 sex extension to help make it easier for others to write games of their own.

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.