

Why Inform 7?

How can writing teachers use Inform 7? The essay “Inform 7 and the Writing Process” (http://bdesilets.com/if/inform7_writing.pdf) tries to answer this question. It includes an introduction to interactive fiction and a tutorial on Inform 7, too. For most readers, “Inform 7 and the Writing Process” is a prerequisite for the essay you are reading now. Readers who are unfamiliar with Inform 7 really **must** start with that essay. But **why**, exactly, should teachers choose Inform 7? The current essay will look at this question in two ways. First, it will answer in terms of two broad advantages. These advantages are the development of clear thinking through programming, and the creation of essays that simultaneously represent two different modes of discourse, exposition and narration. Then, “Why Inform?” will describe a series of tools for writers that Inform 7 provides, showing how these tools correspond to familiar writing-process implements, such as outlines and storyboards.

Clear Thinking Through Programming

Inform 7 is a programming language, though it’s much friendlier and easier to learn than other programmers’ utilities. For many years, scholars have noted that the process of writing good computer code is similar to that of writing good stories and essays, and have wondered whether the right sort of programming experience might improve students’ composition skills. Perhaps the most prominent of these scholars is Seymour Papert, whose seminal book *Mindstorms: Children, Computers, and Powerful Ideas* (1981) captured the imaginations of many educators. Papert advocated the use of child-friendly programming language called Logo to help students apply “powerful ideas,” such as constrictive repetition (recursion) and the organizational relationship of parts to wholes. (For an example of how Logo can help writing teachers, have a look at “Logo and Extended Definition,” at <http://bdesilets.com/if/Logo.pdf>). Today, Linda Sandvik and her associates teach programming to ten-year-olds in the UK, following a similar rationale and a variety of coding software, including Scratch, which is especially relevant for very young children. Inform 7 requires, and, to some degree, teaches powerful ideas, too. However, for learners who are in their teens and beyond, Inform 7 can be more useful for writing teachers than other programming environments because it requires, and enables, students to write computer code that consists of ordinary English sentences. These sentences add up to an expository essay of the process-analysis type. In addition, Inform 7’s code results in output that takes the form of another familiar mode of discourse, a prose narrative.

“Two Birds” (Two Modes of Discourse)

One reason to use Inform 7, then, is its unique way of providing students with rigorous practice in at least two of the traditional modes of writing, exposition and narration. As a very simple example, consider this ridiculously-brief interactive story.

A Riddle

An Interactive Fiction by Brendan Desilets

Release 1 / Serial number 101120 / Inform 7 build 6E72 (I6/v6.31 lib 6/12N)

Riddle Room

This is a room with a big sign and a number of toys scattered around.

The big sign poses a riddle. "What is tall as a house, round as a cup, and all the king's horses can't draw it up? Pick up the correct toy to answer the riddle."

You can also see a the rubber duck, a Tonka truck, a Barbie doll, a miniature robot, an old cell phone, a toy well, a wooden plane, a plastic hammer and a picture book here.

(At this point, the reader sees a prompt like this ">," indicating that he or she can type in any response that he or she likes. Let's say that the reader types the following.)

```
>take well
```

"Well" done! You've successfully solved the riddle, thus completing the story.

In order to produce this very simple narrative, the student would have to write "source code" consisting of a brief process-analysis essay, which tells the computer how to present the story. The source code would look something like what follows.

```
"A Riddle" by Brendan Desilets
```

```
The Riddle Room is a room. "This is a room with a big sign and a number of toys scattered around."
```

```
The sign is in the Riddle Room. "The big sign poses a riddle. 'What is tall as a house, round as a cup, and all the king's horses can't draw it up? Pick up the correct toy to answer the riddle.'"
```

```
The rubber duck, the Tonka truck, the Barbie doll, the miniature robot, the old cell phone, the toy well, the wooden plane, the plastic hammer, and the picture book are in the Riddle Room.
```

```
Instead of taking the toy well:
```

```
say "Well done! You've solved the riddle.";  
end the story, saying "Congratulations! You have won."
```

In order to get even a very simple story to work, an author has to think through the narrative elements that teachers often ask their students to consider, such plot and setting. However, Inform 7 has its own unique way of pushing students in the right direction. With respect to setting, for example, Inform 7 insists that the author create at least one location. Inform does not allow a simple example story without an assertion like "The riddle room is a room." Without such a sentence, Inform will not create a story that "compiles" or runs at all. With respect to plot, Inform requires that the author declare explicitly when an ending of the story has been reached. In our simple story, we need a statement like "end the story." Otherwise, the story continues forever, in a blatantly unsatisfactory way, without any clear resolution.

In helping their students to write process analysis, or “how to” essays, teachers often stress the need for writers to create a series of clear, articulated steps. Inform 7, to a considerable degree, mandates such steps. In our “Riddle Room” example, for instance, the story would be utterly incomprehensible with the step that creates the sign and provides its description. Without the step that creates the toy well, the rubber duck, and the other playthings, it would become quickly obvious that the reader would be unable to make the slightest progress. Without such progress, the story really would have no plot at all.

Familiar Tools for Writing -- the Outline

Another way to understand Inform 7's usefulness for writers, and for teachers of writing, is to consider some traditional tools for composition and Inform's corresponding utilities. One such tool is the outline. Writers use outlines in various forms for varying purposes. Sometimes outlines are quite formal; at other times, they are “quick and dirty.” In the course of prewriting, some authors make outlines to organize their material before drafting. Other writers favor outlining as a revision technique, used to check a draft's unity and/or organization. Some use outlining in both ways. Teachers of writing sometimes like outlining because it helps students to understand what we mean by “unity,” “coherence,” and “organization.”

Inform 7 provides several different outlining tools, all of them useful for at least some writers. The most obvious of these allows an author to categorize the divisions of a story in a built-in hierarchical system, which writers usually employ to map a story. The system looks like this:

Volume
 Book
 Part
 Chapter
 Section

Inform lets writers use any, none, or all of these categories, but it rewards authors for using them by keying its error messages to the categories. In other words, when Inform detects an error in an author's code, it points to the category in which the problem has occurred. Thus, Inform provides student writers with a special incentive to use a type of outlining as a prewriting technique. Of course, many student projects will not require all five of Inform's category types, but many projects become more manageable through the use of one or two.

Another of Inform's outlining tools enables the writer to quickly see which of the story's objects enclose other objects. This tool appears under the Index/World tab of the Inform 7 application. Let's think of our “Riddle Room” example to illustrate. When the story begins, the Index/World tab reveals an outline that looks like this:

Source | Errors | Index | Skein | Transcript | Game | Documentation | Settings |

< > Contents | Actions | Kinds | Phrases | Rules | Scenes | World

World Index

A map of the geographical layout; 🔍 contents of each room; and 🔍 an A to Z of rooms and things.

What's the map? ?; *Using regions* ?; *If the map looks wrong...* ?; *Exporting to EPS* ?



- + **Riddle Room** - *room where play begins* ? ?
 - + big sign ?
 - + rubber duck ?
 - + Tonka truck ?
 - + Barbie doll ?
 - + miniature robot ?
 - + old cell phone ?
 - + toy well ?
 - + wooden plane ?
 - + picture book ?
 - + plastic hammer ?
 - + yourself - *person*

Even in the case of this very simple story, this sort of outline can be useful as the writer revises. If, for example, the author had made the common mistake of creating the object called the “wooden plane” but had forgotten to place it in the Riddle Room, the outline would clearly show the error. Inform 7 updates this outline throughout the testing of the story. If, for example, we ask Inform to generate this outline after the player has picked up the rubber duck, we would get

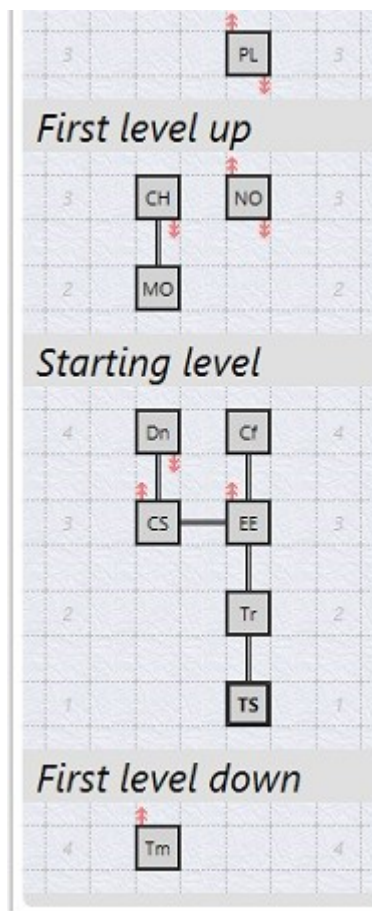
Riddle Room--room where play begins

- big sign
- Tonka truck
- Barbie doll
- miniature truck
- old cell phone
- toy well
- wooden plane
- picture book
- plastic hammer
- yourself -- *person*
- rubber duck

It's easy to imagine how useful this sort of outline can be, as an author creates more rooms and objects, some of which can contain one another.

Familiar Tools for Writing -- the Map

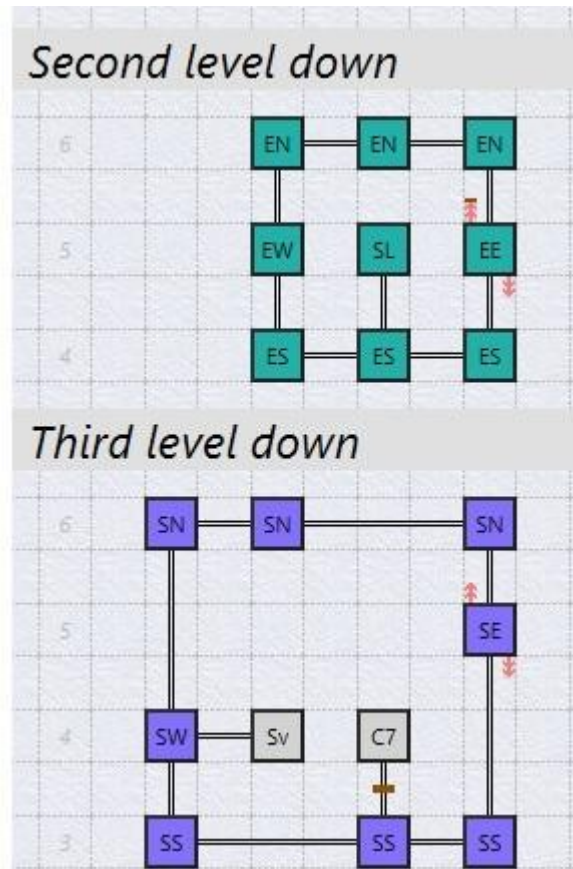
Inform 7 offers no facility for creating ideas maps, such as those generated by computer programs like Inspiration and FreeMind. However, Inform can help writers to make maps of their stories' settings. Many writers of conventional narratives, and of other works, often use such maps as prewriting and revising tools. Mapping is especially helpful in interactive fiction because most works of interactive literature, unlike our ultra-quick "Riddle Room" example, require the reader to move from place to place. Inform 7 automatically creates a map of each of a story's locations, showing the connections among them. Inform displays these maps in the "Index/World" part of the program. Here's an instance, which represents, in part, a story that takes place on several floors of a building:



Notice that this map shows not only the locations, with their names abbreviated to two letters, but also the ways in which they are connected and the levels on which they exist. Though the locations in this story are not particularly numerous or complex, this sort of map makes it easy to identify and repair the kinds of problems that frequently occur, as students write interactive stories. The room that appears at the bottom of this map is labeled "Tm" (for "Tomb" in this case). The red arrows at the top of "Tm"

indicate that the player can go in the direction “Up” from this location. But suppose, for example, that the author had erroneously created the room abbreviated “Tm” without connecting it to the room above it. The map would immediately show the error.

Inform automatically generates the “levels” that appear in our example map, whenever the author creates a location that is up or down from another location. However, Inform also allows the writer to invent his or her own regions, which can contain various locations. Inform color-codes these regions on its map, as in the following:



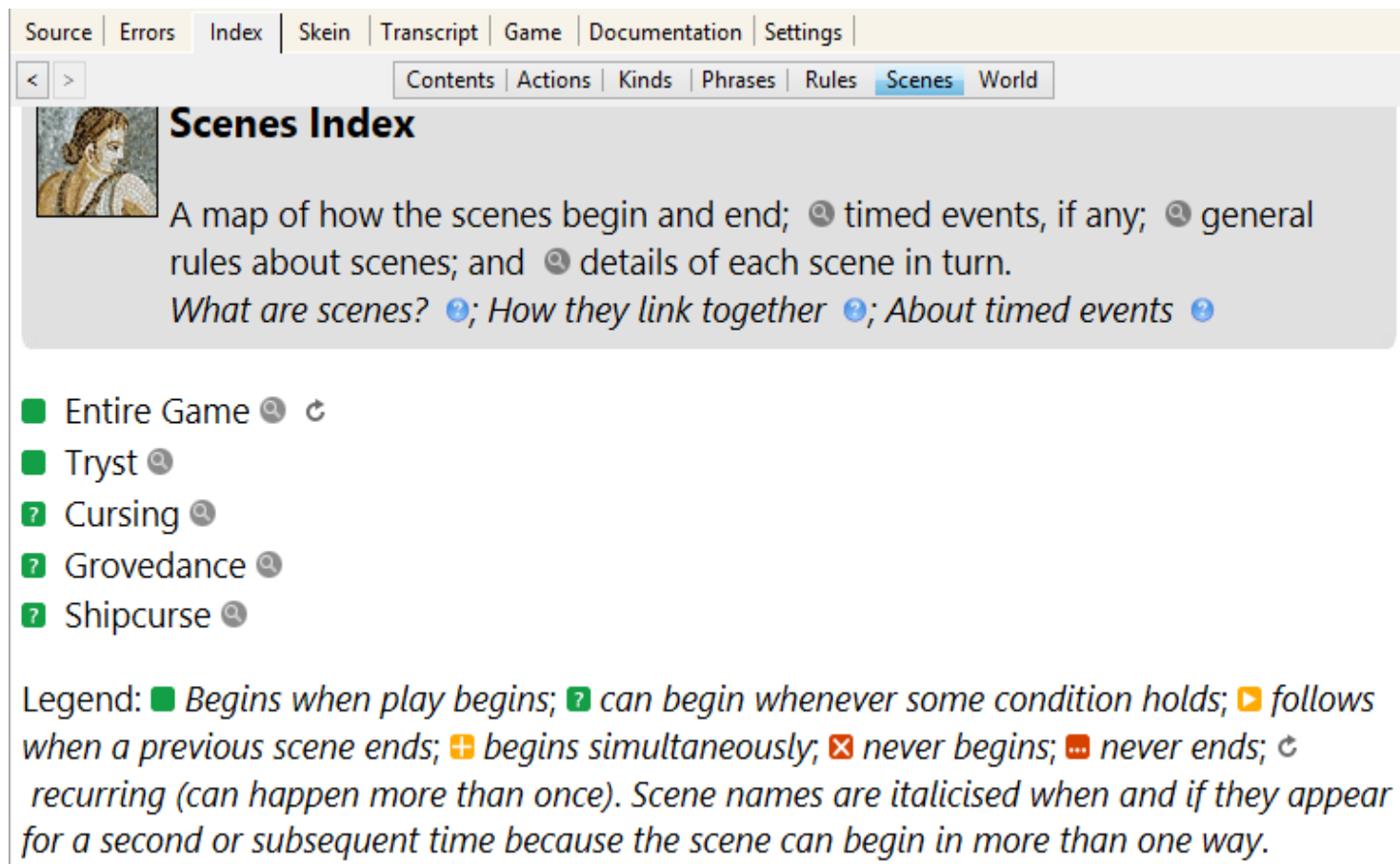
This map shows that all of the rooms in the “Second Level Down” (“down” from the starting level, that is) are in the same region as one another. Inform has mapped these rooms in pink. The writer has created a second region to contain most of the rooms in the “Third Level Down.” This region is mapped in purple. Two of the rooms on the “Third Level Down,” however, are not in any region at all, as shown by their white coloration.

Inform’s mapping facility, then, offers student-writers a friendly mapping tool for their story planning and for their revising and debugging. This mapping system is largely automatic in its handling of levels and connections, but it’s also easy to customize through the use of author-created regions. An Inform 7 map often provides a clear and useful visualization of a story’s physical coherence (or, perhaps, its lack of coherence).

Writers of stories often make storyboards to plan out the scenes of their narratives. Storyboards often depend largely on pictures, but, of course, they can make good use of text, too. Inform 7 sticks with the words. Inform doesn't force its authors to formally create scenes, but it makes it easy for them to do so. Once scenes have been created, Inform represents them, using two different tools. The author of an extremely simple story like "The Riddle Room" would probably not employ scenes, but writers of more complex tales almost always use them, often to the extent that managing scenes becomes a major debugging (revising) task.

The Scenes Index

One of Inform's tools for representing scenes is in the Index/Scenes part of the program. This utility lists all of the story's scenes and offers useful information about each of them. Let's say that a writer has created four scenes in a story based on the opera *Dido and Aeneas*. Inform's Index/Scenes tab might reveal something like this:



Source | Errors | Index | Skein | Transcript | Game | Documentation | Settings

< > Contents | Actions | Kinds | Phrases | Rules | Scenes | World

Scenes Index

A map of how the scenes begin and end; ⌚ timed events, if any; ⌚ general rules about scenes; and ⌚ details of each scene in turn.

What are scenes? ⓘ; How they link together ⓘ; About timed events ⓘ

- Entire Game ⌚ ⓘ
- Tryst ⓘ
- ⓘ Cursing ⓘ
- ⓘ Grovedance ⓘ
- ⓘ Shipcurse ⓘ

Legend: ■ Begins when play begins; ⓘ can begin whenever some condition holds; ⓘ follows when a previous scene ends; ⓘ begins simultaneously; ⓘ never begins; ⓘ never ends; ⓘ recurring (can happen more than once). Scene names are italicised when and if they appear for a second or subsequent time because the scene can begin in more than one way.

Note that the "Entire Game" counts as a scene, which encompasses all the other scenes. The Index/Scenes tab also provides more detailed information about each scene, as appears here:

The Entire Game scene recurring

The Entire Game scene is built-in. It is going on whenever play is going on. (It is recurring so that if the story ends, but then resumes, it too will end but then begin again.)

Begins when: the story has not ended

Ends when: the story has ended

The Cursing scene

Begins when: the player is in the Lair of the Sorcerer

Ends when: the Magician is triumphant

The Grovedance scene

Begins when: the player is in the Grove and the Magician is triumphant

What happens:

(move Aeneas to the Grove; ...)

Ends when: the Magician is prophetic

The Shipcurse scene

Begins when: the player is in the Beach and the Magician is prophetic

What happens:

(move the enchantresses to the Beach; ...)

Ends when: the Magician is finished

The Tryst scene

Begins when: play begins

Ends when: Dido is off-stage

Here, we can see the events that mark the beginning and end of each scene. If particular event happens at the beginning of the scene, we can see what that event is. If a scene can happen more than once, we can see that the scene is “recurring.” In our example, each of the scenes will occur only once, except the “Entire Game” scene.

The “Scenes” Debugging Command

When problems with scenes develop during the writing of an interactive story, as they very frequently do, the information in the Scenes Index can be quite helpful, but the “Scenes” debugging command usually works even better. The writer uses this command while testing the story, as it runs within the Inform application. For the most part, this test run of the narrative looks exactly as it would look to a reader, except that the author has some special “debugging” commands that will not be available to readers. Let’s suppose that the writer of our “Dido and Aeneas” story types the command “scenes” as soon as the story starts. The output will look like this:

```
>scenes
Scene 'Entire Game' playing (for 0 mins now)
Scene 'Tryst' playing (for 0 mins now)
(Scene monitoring now switched on. Type "scenes off" to switch it off
again.)
```

A few turns later, the “scene monitoring” function that the author activated reports

```
[Scene 'Tryst' ends]
```

Now, the “scenes” command shows the following:

```
>scenes
Scene 'Entire Game' playing (for 5 mins now)
Scene 'Tryst' ended
(Scene monitoring now switched on. Type "scenes off" to switch it off
again.)
```

The information that the scene “Tryst” started and ended when it was supposed to is of considerable importance to the writer. When something goes wrong, as it often does, the “scenes” command assumes even greater importance, since it points, quite specifically, toward the problem.

In effect, then, Inform 7 automatically creates a detailed storyboard in a way that helps writers construct an clear and coherent narrative. Thus, Inform offers students a dramatic instance of the value and techniques of effective storyboarding.

Most writers prefer to write with a word processor. Interactive fiction authors, though, are often especially appreciative of the text editor that comes with Inform. Like many programmers' text editors, the Inform utility makes the writer's work more efficient through a helpful system of color coding. In Inform, instructions to the computer appear in black, and text that is to be printed to the screen is dark blue. Sometimes, Inform programmers include instructions to the computer within printed text. These instructions appear in light blue. The programmer's "comments," which make the source code easier for humans, including the author, to understand, are green. The example that follows make look as if it's contrived just to show off Inform's color system, but, actually, the source code comes from a real IF story and is quite typical.

[Rooms]

The Palace Knoll is south of the Path. The description of the Palace Knoll is "A secluded green knoll, with a park bench, near the stately Palace of Dido, the Queen of Carthage. An overgrown path leads north." The description of the Path is "[if visited]A simple path, leading toward a dark, craggy area to the north. You can also follow another path to the east.[otherwise]You enter a simple path, branching toward a dark, craggy area to the north and a lovely forest to the east; however, immediately after leaving the Palace Knoll, you are attacked by a pack of vicious hounds.[end if]"

At the top of this example text, we find a comment, [Rooms], suggesting that the next section of code will implement the various locations of the story. The brackets identify this text as a comment, and Inform marks the comment in green, helping the writer to avoid such common mistakes as failing to close the brackets correctly. The next bit of text is

The Palace Knoll is south of the Path.

This text appears in black type because it addresses the computer itself (or perhaps it addresses Inform, depending on how you look at the nature of a programming language). This text creates two locations, or "rooms," called "Palace Knoll" and "Path," the former located immediately south of the latter. A few words later, we find the description of the Palace Knoll, enclosed in quotation marks and colored dark blue. If the writer had inadvertently ended the description with an apostrophe rather than a closing quotation mark, the color coding would clearly indicate the typo.

The description of the Path is a bit more complicated, since it actually consists of two different descriptions. The descriptions are within quotation marks and are coded in blue. One description, the one that appears first in the text, is the usual description of this location. It is the description that the reader would see if he or she had visited this room before. Thus this description is marked [if visited], with the bracketed text shown in lighter blue, since the bracketed words do not actually print on the reader's screen. The second description would appear only if the reader had never visited this location before.

Familiar Tools for Writing -- the Usage Checker

Many word processors, including Microsoft Word, include software that checks for various

usage errors, such as mistakes in punctuation and sentence structure. These checkers are usually so unreliable that few writers use them. Oddly, these checkers often do not check for errors that they actually could get right consistently, such as unpaired parentheses or quotation marks.

Inform 7, though, like most other programming languages, requires the use of a specialized sort of usage checker called a compiler. The compiler carries out the essential task of translating the source code that writer creates into a form that a computer can read.


Of course, all programmers, like all other writers, make mistakes. In a programming environment, many, though not all, of those mistakes cause the compiler to be unable to do its work. When the compiler finds that it cannot translate source code in a machine-readable form, the compiler tries to point the programmer toward the code that is problematic, often with a line number and a brief, cryptic description of the error.


The Inform 7 compiler is much friendlier than those of most other languages. Since Inform 7 doesn't usually use line numbers, the Inform 7 compiler provides the writer with a link to the erroneous source code. Most of the time, Inform also offers a reasonably well-developed and clear description of the problem it has found in the code.

To illustrate, let's start with a simple of example of problematic source code:

```
The Kitchen is a rom. The description of the Kitchen is "An ordinary food-preparationn area."
```

When we try to compile this code, we get the following result:

Problem. The sentence 'The Kitchen is a rom'  appears to say two things are the same - I am reading 'Kitchen' and 'rom' as two different things, and therefore it makes no sense to say that one is the other: it would be like saying that 'John is Paul'. It would be all right if the second thing were the name of a kind, perhaps with properties: for instance 'Abbey Road is a lighted room' says that something called Abbey Road exists and that it is a 'room', which is a kind I know about, combined with a property called 'lighted' which I also know about.

Problem. You wrote 'The description of the Kitchen is "An ordinary food-preparationn area."' : but this seems to say that a thing is a value, like saying 'the chair is 10'.

The compiler has found two errors in this source text. Clicking on the orange arrows that

appear in the problem reports will cause the offended passages to be highlighted in the writer's source code, showing, with some precision, where the difficulty is.

The compiler first identifies "The Kitchen is a rom" as problematic. Inform is not smart enough to guess that "rom" is just a misspelling, but it does provide an error message that would help most writers. In fact, Inform offers a highly relevant example of good code, "Abbey Road is a lighted room."

Then the compiler finds a second mistake, "The description of the Kitchen is 'An ordinary food-preparationn area.'" Once again, Inform does not realize that the problem is really a spelling error ("description"). This time, though, Inform's description of the error, though perfectly accurate, would be much less helpful to a novice writer."

Notice that the compiler does not find a third spelling error, "food-preparationn." Because this misspelling occurs in text that is to be printed to the screen, the compiler does not try to understand the word "food-preparationn," nor does it check the word's spelling. Instead, the compiler assumes that its job is simply to see that the text appears for the reader, exactly as the writer has typed it in. Inform does have its own spell checker, which works separately from the compiler. This spell checker would have found all three errors, though most writers run the compiler more often than the spell checker and so would have found the first two errors more quickly, when the compiler pointed them out.

Of course, the vast majority of coding errors are not just spelling mistakes or typographical errors. The more advanced errors offer writers an opportunity, and some motivation, to check into Inform's well-indexed documentation, or to ask for help from other authors. In any case, the compiler enforces a certain care and precision in any writing of Inform code.

Familiar Tools for Writing -- Transitional Words and Phrases

Teachers of writing often suggest to students that they check the coherence of their writing. One tool for establishing coherence is the transitional word or phrase, such as "however," or "on the other hand." Most writing texts offer lists of commonly used phrases of this sort, but writing teachers usually put just as much stress on the use of transitional devices that arise more spontaneously in particular pieces of writing. For instance, in order to improve coherence, a writer may use a technical term, such as "compiler," and judiciously repeat this word and its forms.

In interactive fiction, programming structures called "values" provide great transitional power. In fact, writers sometimes think of Inform's values as ways for one part of an Inform story's code to send "messages" to another part. Suppose, for example, that, in one part of an interactive story, the main character may or may not drink a triple shot of whiskey. Suppose, further, that, an hour later in game time, the character may or may not decide to drive a car. The Inform author will likely use a value to send, to the later part of the story, a message about whether the character imbibed heavily earlier in the game.

Inform tries to make the use of values both simple and powerful. In order to do so, it enables the writer to set up values in more than one way. Perhaps the most powerful way to implement a value is to create it as a noun, along with adjectives that are related to the noun. Here's an example:

```
Sobriety is a kind of value. People have sobriety. The
sobrieties are drunk and sober. A person is usually sober.
```

This source text enables Inform to understand “drunk” and “sober” as adjectives that may or may not apply to any person in the story, including the “player/character,” the character that the reader controls. The phrase “A person is usually sober,” in the source text as we have written it, establishes the starting condition for this value. In other words, the player/character, and all other characters, start out sober, unless the writer explicitly declares otherwise. However, events in the story may cause the player's condition to change.

Now that we have created the adjective “drunk,” the phrase “when the player is drunk,” is a condition to which Inform can respond in whatever way the author deems appropriate. For instance, the writer could notify the reader of the player/character's sobriety after every turn of the story, using the following source code:

```
Every turn when the player is sober, say "You are sober."
```

Once the sobriety value is established, the writer can change its state. For example, the following code would produce a change when the player/character imbibes.


```
The player carries some whiskey.
```

```
Instead of drinking the whiskey: say "You drink the whiskey";
now the player is drunk.
```

The use of values in interactive fiction is a bit abstract and elusive for many new writers. However, Inform 7 offers two tools to help. One of these is a listing of values which, like our “sobriety” example, are set out as “types of value.” This listing appears in the Index/Kinds tab of the program. For our “sobriety” example, the relevant section of the Index/Kinds tab looks like the following:

Source | Errors | Index | Skein | Transcript | Game | Documentation | Settings |

< > Contents | Actions | **Kinds** | Phrases | Rules | Scenes | World



Kinds Index

Table of all the kinds; ⓘ how arithmetic affects them; and ⓘ details of each kind in turn.

What are kinds? ⓘ; More about kinds of object ⓘ; And kinds of value ⓘ

Value	Default Value
Sobriety[2]	sober

Inform also offers a debugging command that an author can use while testing a story. This command, “showme,” offers a listing of information that includes the state of relevant values. In our example, “showme me” yields this information about the player/character, if he or she has not consumed the whiskey:

```
>showme me
yourself - person
  whiskey
location: Kitchen
singular-named, proper-named; unlit, inedible, portable; male
printed name: "yourself"
printed plural name: "people"
indefinite article: none
description: "As good-looking as ever."
initial appearance: none
carrying capacity: 100
sobriety: sober
```

After the inbibing, we’d get the following:

```
>showme me
yourself - person
location: Kitchen
singular-named, proper-named; unlit, inedible, portable; male
printed name: "yourself"
printed plural name: "people"
indefinite article: none
description: "As good-looking as ever."
initial appearance: none
carrying capacity: 100
sobriety: drunk
```

For almost all inexperienced writers, it's difficult to create a strong sense of coherence in any sort of writing. For writers of interactive fiction, values can help to develop good coherence, but only if they are implemented correctly. By using Inform's tools for creating well-thought-out values, novice writers can sharpen their understanding of just what it takes to write coherently. Indeed, writers of interactive fiction really **have** to write coherently, using values and other techniques, if they are to produce works that look like IF stories at all. In its own way, Inform 7 enforces coherence.

Familiar Tools for Writing -- Dictionaries and Thesauruses

Like all writers, IF authors have to choose the right words. For this purpose, they have to find words that are potentially "right." However, in using a programming language, the notion of "right word" takes on a whole new meaning, because the language understands only a small subset the words from any natural language, such as English. Inform offers the writer two important tools for identifying words that the language can understand. One of these tools focuses on what Inform calls "actions," and the other offers information about "phrases."

A Thesaurus of Actions

In Inform, the following code will compile and work:

```
Instead of taking the anvil, say "That's too heavy to lift."
```

But this code will not compile:

```
Instead of picking up the anvil, say "That's too heavy to lift."
```

The problem centers on the way Inform handles actions that the reader can invoke.

Inform 7, in its native form, allows **readers** to use a total of 143 verbs. "Take" is one of those verbs, and so is "pick up." However, Inform 7, in its native form, understands only eighty different actions. The sixty-three verbs that are not actions serve as synonyms that the reader can use. The **writer**, though, must use the single unique word that Inform recognizes as an action. Let's look at our example rule once again:

```
Instead of taking the anvil, say "That's too heavy to lift."
```

If the writer has implemented the anvil, this rule will compile and work. In a story that uses this rule, the player can type "Take the anvil," and get the response, "That's too heavy to lift." The player can also type "Pick up the anvil," and get the same response. If the player types, "Grab the anvil," the response will be "That's not a verb I recognize."

If a writer wants to allow new synonyms for an action, he or she can do so quite easily. To add the “grab” synonym for “take,” the writer would add the following code:

```
Understand “grab [something]” as taking.
```

Implementing new actions is a bit more complicated than creating synonyms, but Inform writers can make as many new actions as they like, too.

Inform 7 provides several tools to help readers with its specialized vocabulary. One of these is accessed through Inform’s Index/Actions tab. This tool lists all of the actions that Inform understands, including any new actions that the author has created, along with the synonyms of each action.

Inform 7 also offers an “actions” debugging command, which an author can use while playing through a story. With the many ways in which actions and their synonyms can interact, this command can help a writer to sort through what’s really happening behind the scenes of an IF work. The “actions” command can prove especially helpful when a story is not behaving the way the author expects.

A Thesaurus of Phrases


Of course, not all problems with Inform 7 vocabulary involve actions. Consider another bit of code that will not compile:

```
Rather than taking the anvil: say “That’s too heavy to lift.”
```

In this instance, Inform’s rigid preference for “instead of” over “rather than” does not hinge on any particular action. To sort out this sort of vocabulary problem, Inform offers its Index/Phrasebook tab. The tab offers rather long list of phrase types, each followed by a magnifying glass icon that links of an explanation. Here’s a look at part of what this tab offers.

Source | Errors | Index | Skein | Transcript | Game | Documentation | Settings

< > Contents | Actions | Kinds | **Phrases** | Rules | Scenes | World



Phrasebook Index

A guide to the phrases allowed; a lexicon of words; a table of relations, and of verbs.

What are phrases? ; And descriptions? ; Relations? ; Verbs?

Saying

Values, Names with articles, Say if and otherwise, Say one of, Paragraph control, Special characters, Fonts and visual effects, Some built-in texts, Saying lists of things, Group in and omit from lists, Lists of values

Values

Making conditions true, Giving values temporary names, Changing stored values, Arithmetic, Enumerations, Truth states, Randomness, Tables, Sorting tables,

Using Inform's Thesauruses

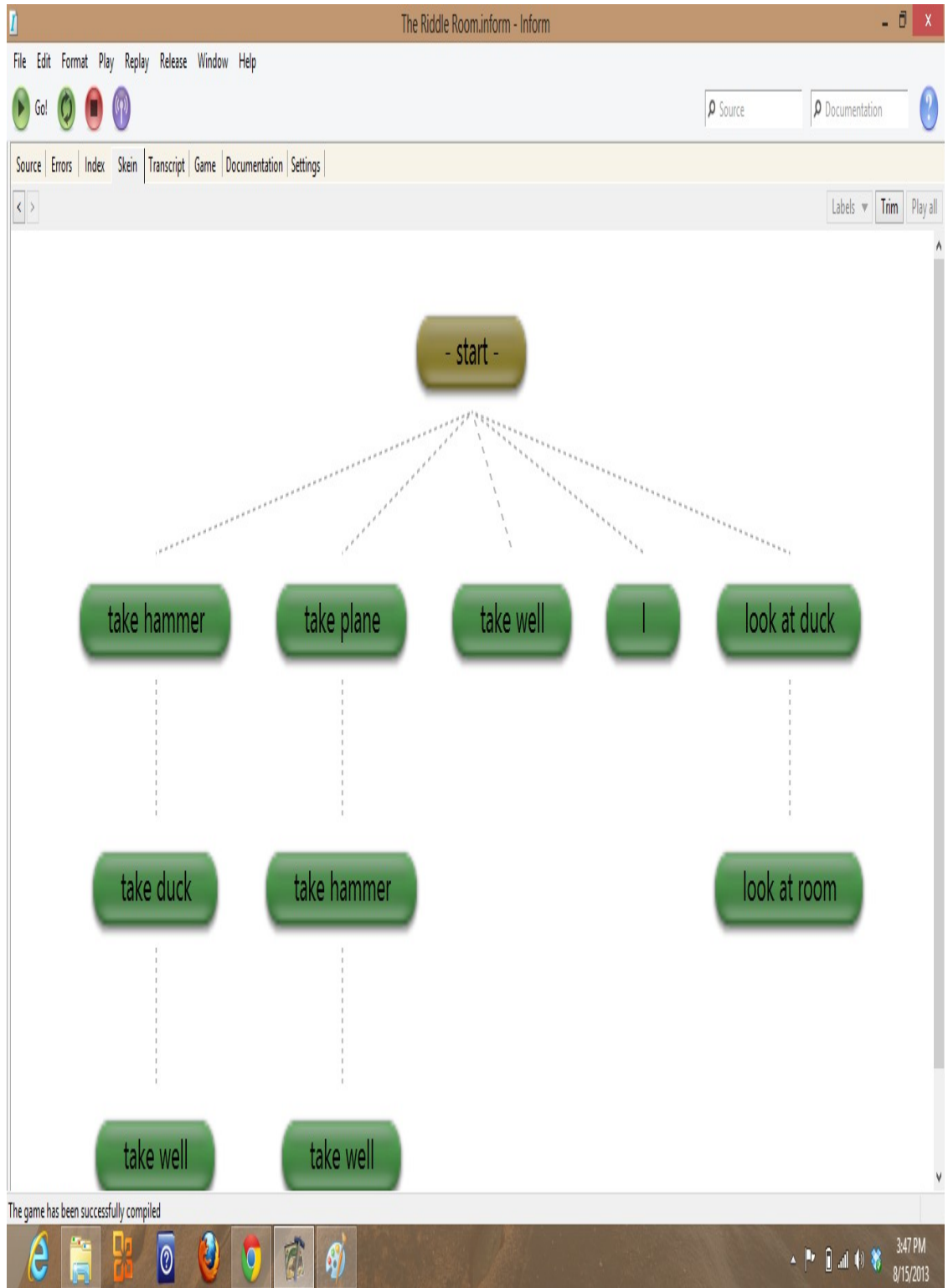
A good dictionary or thesaurus can help students to use words more clearly and precisely. In its own, demanding way, however, Inform goes much farther. In order to get a story to compile and run, a student author of an Inform story must use the vocabulary of action words and function words precisely. Sooner or later (usually sooner), Inform writers develop a considerable facility with the tools that help them to achieve this kind of precision.

Familiar Tools for Writing: Rereading to Revise and Edit

Student writers usually learn a variety of techniques for rereading their work to spot likely sites for improvement. Some become experts at rereading once for unity, a second time for coherence, and a third time for clarity. Some use mnemonics like CUPS (capitalizing, usage, punctuation, spelling) to help them with their editing. Many learn to read the work aloud in order to “hear” problems. Writers of interactive fiction face a particularly time-intensive task when they seek to reread a story, since any particular reading requires the input of a series of commands that the reader might type. As a story becomes more complex, this list of commands can become very lengthy.

Inform 7 addresses this issue with a tool called the “skein,” which records all the input

that the writer uses in testing a story and allows for the nearly-instantaneous playback of that input. The skein is so important that it has its own tab in the Inform application. Let's take a look at a skein that represents five iterations of our "Riddle Room" story.



This skein shows three run-throughs of the story that reach its conclusion, which occurs with the taking of the well. However, the skein recalls all the iterations the author has tried, including, in this case, two iterations that did not reach a conclusive result. Let's suppose

that an author has made some changes to the code that involves the hammer and wants to see the result. Instead of inputting a series of commands by hand, the writer can open the skein and double-click on any of the skein's nodes, thus causing the story to run itself up to that point. In this case, the author might click on the "take hammer" node. Then, the author could examine the output to see that her changes to the code produced the desired result.

As one might imagine, the skein can be handy in revising even a very simple story like our "Riddle Room" example. For revising a more typical story that requires fifty or one hundred commands to reach its conclusion, the skein is invaluable.

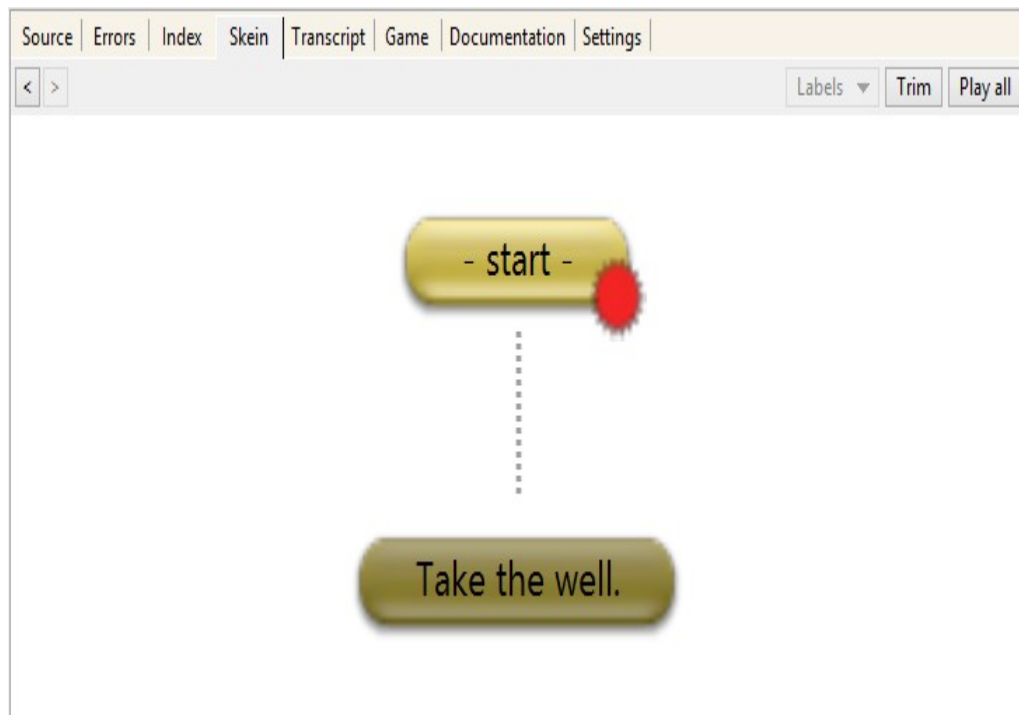
The skein offers an unusual pedagogical advantage for writing teachers in that it emphasizes an aspect of revising that students often overlook. When students make significant revisions, as they often should, students frequently fail to identify the impacts of their revisions on later sections of their essays. Even the most superficial experimenting with the skein shows how a change in one part of a piece of writing can have unanticipated effects on the rest of the text.

Familiar Tools for Writing: Side-by-Side Comparisons of Drafts

When students get into difficult tangles in revising their essays, they often find themselves wondering whether an earlier draft is actually better than a revised version. Comparing the two, side by side, can help to resolve this sort of conundrum.

Inform 7, in its Windows and MacOS versions, offers writers an extension of the skein tool that can help them to make intelligent comparisons of drafts. This tool, which does not appear in the Linux version of Inform, is called the transcript. Like the skein, the transcript has its own tab in a typical Inform window.

If a writer right-clicks on a node in any Inform skein, the resulting options will include "Show in transcript." This option, at first, simply shows the output of the chosen skein, up to the chosen node, in the "Transcript" tab. For example, consider the screen shot of a skein that appears below. It's from "The Riddle Room."



This skein would lead to the successful completion of the story. If the writer wanted to view the output of this skein in the Inform transcript, he or she would simply right-click on any node of the skein and choose the option “Show in transcript.” Now, the transcript tab would show two columns, one of which would show with the output of the skein that the author chose. In our case, this output would be as follows:

```
Pick up the toy that solves the riddle.
```

```
The Riddle Room
```

```
An Interactive Fiction by Brendan Desilets
```

```
Release 1 / Serial number 130816 / Inform 7 build 6G60
```

```
(I6/v6.32 lib 6/12N) SD
```

```
Riddle Room
```

```
This is a room with a large sign and a bunch of toys  
scattered around.
```

```
The big sign poses a riddle. "What's tall as a house, round  
as a cup, and all the king's horses can't haul it up?"
```

```
You can also see a rubber duck, a Tonka truck, a Barbie  
doll, a miniature robot, an old cell phone, a toy well, a wooden  
plane, a picture book and a plastic hammer here.
```

```
>Take the well.
```

Congratulations! You've solved the riddle.

In addition to showing the writer the output of this run-through of the story, the transcript offers a series of clickable buttons that let the author “bless” the skein up to any particular node. Typically, an author chooses to bless a “good” skein, a skein that represents a desirable sequence of moves that produces a useful result.

Now, let’s suppose that the author continues to work on the story, and, in the course of doing so, inadvertently changes a word in the description of the sign from “king’s” to “kink’s.” With this error in place, if the author goes to the skein and double-clicks on the “Take the well” node, the transcript will compare the new version of the story with the blessed skein, underlining all the differences between the two. The transcript would look, in part, like this:

The big sign poses a riddle. "What's tall as a house, round as a cup, and all the <u>kink's</u> horses can't haul it up?"	The big sign poses a riddle. "What's tall as a house, round as a cup, and all the <u>king's</u> horses can't haul it up?"
---	---

The highlighting of this typo could, of course, be useful to a writer, but, in a longer, more complex story, the transcript can be much more effective. Typically, in Inform, as in other programming languages, single erroneous revision in source code can result in dozens of embarrassing errors in the program’s output.

When used with a story of any significant length, the Inform 7 transcript is likely to reveal a significant, and sometimes surprising, list of differences between versions of a tale. For this reason, the transcript often underscores the story-wide implications of revision in a way that a convention comparison of drafts cannot.

Do Students Actually Use These Inform 7 Tools?

In a programming class, a teacher might well assign the use of various tools that a language offers, tools like Inform 7’s skein. However, in a typical writing class, an instructor would probably not do so. Instead the professor would likely work with two kinds of assignments, short easily-programmed exercises for all students, and optional, more ambitious projects for the ten percent of students who like an unusual challenge. In response to the first assignment, students might, after an hour or so of instruction, produce very simple stories like “The Riddle Room,” which do not require the use of many of the tools described here. The tools that students must use in even the simplest stories are still significant, however. These include the color-coded text editor (corresponding to a word processor) and the compiler (usage checker). Even in very quick projects, most students will also use the skein (reading for revising) and the actions tab (thesaurus).

Students who take on longer stories will almost always use more of Inform 7's tools. Of course, the writers' choice of implements will depend partly on the nature of their stories. Stories that take place in one room, for example, won't make many demands on the mapping utility. Nearly all authors of longer stories, on the other hand, will use values and scenes.

Works Cited

Papert, Seymour. *Mindstorms: Children, Computers, and Powerful Ideas*. New York, New York: Basic Books, 1980. Print.

Sandvik, Linda, perf. "Interview With Linda Sandvik of Code Club." *Ubuntu UK Podcast*. Ubuntu UK Local Community Team, 27 Jun 2013. Web. 19 Jul 2013. <<http://podcast.ubuntu-uk.org/2013/06/27/s06e18-a-midsummer-nights-ubuntu/>>.