Descriptions and Locale

Part of the Well-Versed Informer Series

2/12/2010
Jeff Nyman
Version 0.3

Contents

Introduction

Starting Out

The focus of this guide is on the descriptions that an author provides as part of a work on text-based interactive fiction (referred to as textual IF in this guide). These descriptions are the central mechanism that an author uses to convey information about a model world to a reader. Nearly every object and room in a textual IF has a description. From a reader/player standpoint, this is a block of text that is associated with a location or object. This block of text is printed after a LOOK command (for a location) or a LOOK AT or EXAMINE command (for an object).

In most cases the description refers to a *physical description*, by which I mean concise statements about what a given location or object looks like. There is a lot, however, that can be conveyed by a description. For example, in the case of locations, a description can provide the reader with directional information or provide spatial orientation. Descriptions, if care is taken, can also provide information regarding how the model world is evolving and even reflect changes in knowledge on the part of the reader as more information is gained. As with descriptive text in a novel, descriptions in a work of textual IF can – if well-written – serve to convey mood, set tone, and establish atmosphere.

With all this being said, descriptions also serve an important pragmatic purpose: they provide the nouns that a reader of your work should focus on. This is done by presenting information about what can be further investigated in the model world and also what objects are present that the reader can manipulate.

This guide is not going to focus on the nature of descriptions or what they can provide in terms of information. What this guide *is* going to focus on are the mechanics that make all this possible in Inform, providing general understanding of how Inform allows you, as an author, to craft descriptions and various ways that you can manipulate these mechanics to achieve certain effects.

Descriptions

Structuring a Location Description

Why is every place a "room"?

One thing I want to get out of the way quickly is an opinion of mine. That opinion is that Inform has a somewhat poor design decision underlying its presentation in that it refers to *every* location as a room, even if the location represents a cave, a boat in a river, an outdoor spot, etc. From an implementation standpoint it really makes no difference but from a conceptual standpoint, particularly in terms of how Inform presents other elements, I think it's an unfortunate choice. (With that being said, Inform is certainly not the only textual IF system to make this choice. Such systems are, if nothing else, mired in their traditions.)

With the above opinion stated, just note that I may use the terms "location" and "room" somewhat interchangeably although when talking about Inform source text, I'll always use the term "room" since that's what Inform uses.

What's a "room"?

Locations – what Inform universally calls "rooms" – represent places in which story characters can initially be (or go to) and in which objects can be initially placed (or moved to). It's really no more complicated than that.

What's a room description?

The room description would, at a glance, seem to be simply a bit of text that describes the room itself as well as the various items that are located in that room. With that seemingly simple explanation provided, the printing of a room description is a lot more involved than it might initially seem to be. The reason for this is that Inform has to consider not just all of the objects that are in the room to start with, but also all of the objects that the protagonist (or other characters) might have brought into the room and dropped there. Inform also has to figure out which objects are on visible supporters or in visible containers, and then decide how to group and list all of those objects.

All of this behavior is handled as part of how Inform processes the 'looking' action (generated by a 'look' command on the part of the player). Since this involves how the 'looking' action is carried out, you can find the relevant rules in the carry out looking rulebook.

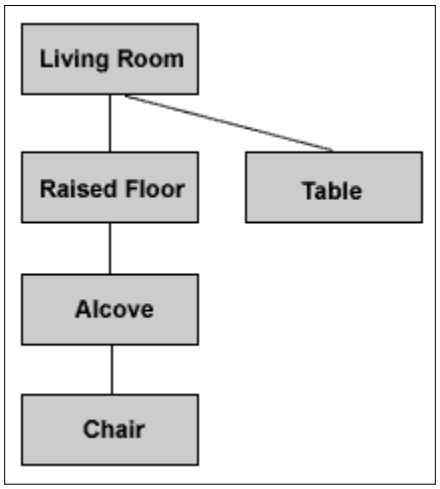
How does the looking action work?

The 'looking' action describes the protagonist's current location and any visible items in that location. That notion of "visible items" is often what complicates things. Inform calculates the notion of 'visibility' by dividing a given location into levels of visibility. This, in turn, is based on the notion of an object tree.

The **object tree** refers to the relationship between Inform objects. Inform uses an object tree hierarchy for containment, support, carrying, wearing, and so on. What this means is that Inform uses this tree to

represent which items are contained inside which other items, which items are placed on top of other items, what location the protagonist is currently in, and so forth.

Here's an example of an object tree that I'm going to be using for a bit as we go forward:



This shows you a set of object relationships. For example, the parent of the table is the Living Room and the parent of the Raised Floor is also the Living Room. You could also say that Raised Floor and the Table are children of the Living Room. The Alcove is a child of the Raised Floor. But a parent of the Chair. That gives you an idea of how Inform would create a tree of objects.

With that out of the way, let's create some source text that will make this object tree a reality to Inform:

SOURCE TEXT

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

A raised floor is in the Living Room. The raised floor is an enterable supporter.
An alcove is on the raised floor. The alcove is an enterable container.
A chair is in the alcove. The chair is an enterable container.
A table is in the Living Room. The table is a supporter.
```

If you start your story, you should see this output:

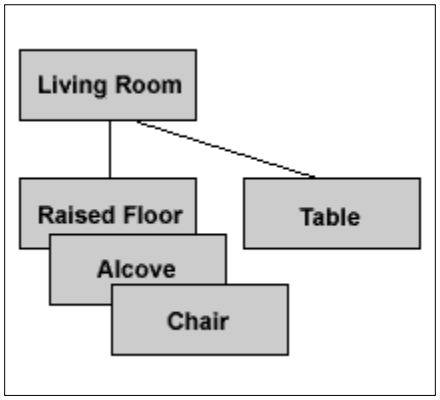
STORY TEXT

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

You can see a raised floor (on which is an alcove (in which is a chair (empty))) and a
table here.
```

That initial bit of text is going to serve as the basis for a lot of the discussion that follows. If you check the 'World' index page you will see that Inform's internal representation of that source text looks like this:

Conceptually, what you have is a situation that looks more like this:



Here you can imagine that you have a Living Room that has a raised floor partition, on which is an inset alcove. Inside that alcove is a chair. The table is also in the Living Room, but not on the raised floor partition. So the layering of the elements in the above diagram may make this more clear as I go forward discussing how Inform models these situations. If nothing else, this can initially help you start thinking in terms of "levels."

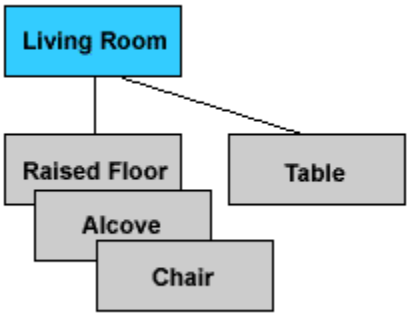
Inform uses the notion of a **visibility ceiling** to indicate the object that represents the uppermost visible level. What that means is the highest object in the tree that is visible to the protagonist. A **visibility level** is then calculated, which is the "distance" in the object tree from the level the protagonist is at to that uppermost level. So let's say the protagonist is simply in a location. In the case of our example, the protagonist is in the Living Room. In that case, the visibility ceiling is the location itself (Living Room) and thus there's only one visibility level. That means the protagonist is at the highest level of visibility for that room.

Logically it may seem that the visibility level should be 0 in this case since the "distance" from the protagonist to the Living Room is effectively is no distance at all. That's not how Inform does it, however. What that "distance" is showing you is how far the protagonist is from the topmost level, which is going to be a room. A room will thus always have a visibility level of 1. To make this point a little clearer, let's consider a slightly more complicated situation. Let's say you have the protagonist in the Living Room, sitting on the chair that's located in that room. But, remember, it's not only that: that chair

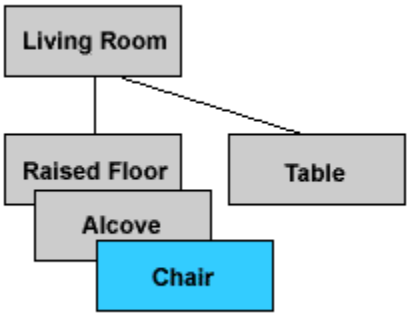
is inside an alcove and that alcove is on a raised floor. This would lead to four visibility levels: chair, alcove, raised floor, and Living Room.

The rules that are processed during a 'looking' action use a special phrase – the **visibility-holder of *X*** – to go up from one level to the next, where *X* is whatever object Inform is considering. So, with this example, the visibility-holder of the chair is the alcove and the visibility-holder of the alcove is the raised floor. Finally, the visibility-holder of the raised floor is the room itself, i.e., the Living Room. That latter would apply for the table as well. The visibility-holder phrase allows Inform to determine how to move from one level to another. Along with that, Inform also uses a **visibility level count** to indicate the specific number of levels that the protagonist can actually see.

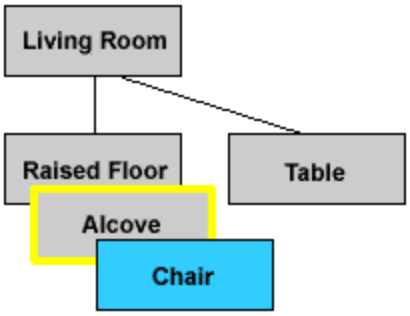
Let's break this down a little into some specifics.



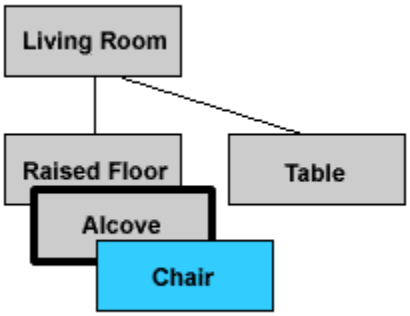
- If the player moves the protagonist into the Living Room, then the visibility level count is 1 and the visibility ceiling is set to the Living Room. The visibility holder of the protagonist would also be the Living Room.



- If the player has the protagonist sitting on the chair – and *assuming the alcove is open and non-opaque* – the visibility level count would be 4 and the visibility ceiling would be equal to the Living Room. The visibility holder of the protagonist would be the chair.



- If the player has the protagonist sitting on the chair – and *assuming the alcove is closed and opaque* – the visibility level count would be 2 and the visibility ceiling would be equal to the alcove. The visibility holder of the protagonist would be the chair. An important caveat to this is tha the alcove must be lit. (That's what the yellow border around alcove is attempting to suggest.)



- If the player has the protagonist sitting on the chair – and *assuming the alcove is closed and opaque and unlit* – the visibilty level count would be 0, the visibility ceiling would be nothing, and the visibility holder of the protagonist would be the chair. An important caveat to this is tha the alcove must not be lit. (That's what the dark border around alcove is attempting to suggest.)

	**Think About It** Read the above bullet points and make sure you understand what they are saying. Particularly make sure you understand why the visibility ceiling and levels differ in the various bullet points.

Light has to be available in order to see anything at all. So if the protagonist is in darkness, the visibility level count is 0, like I already mentioned. However, I indicated that the visibility ceiling is nothing. That's not strictly true. In actuality, you will find that the visibility ceiling in this case is an object called the "(darkness object)".

Incidentally, there's a handy say phrases that you can test out some of these things for yourself and see how Inform is treating the various aspects of visibility levels and the visibility ceiling. You can simply add the following to your source text:

SOURCE TEXT

```
A report looking rule:
    say "Visibility Holder (of the Player): [the visibility-holder of the player].";
    say "Visibility Level Count: [visibility level count].";
    say "Visibility Ceiling: [visibility ceiling]."
```

This can be really useful in terms of checking out how Inform is tracking what "level" the protagonist is at in the object tree.

What rules are processed during a looking action?

Let's consider what happens during a looking action.

Looking (Action)		
set action variables for	looking	determine visibility ceiling rule
carry out		room description heading rule
carry out		room description body text rule
carry out		room description paragraphs about objects rule
carry out		check new arrival rule
Report	an actor looking	other people looking rule

An action variable is useful to keep track of a few values throughout the processing of the action. The setting action variables rulebook is run through even prior to the the before rules, and it has no power to stop or change the action. All that happens is that rulebook variables are set. In this case, this rule handles setting the visibility ceiling, visibility level and visibility-holder variables that I discussed previously. In fact, this is why my handy source text above must be placed in the context of a looking rule. The variables in question are only available during a 'looking' action.

Then there's a series of carry out rulebooks that are processed. Let's take them in order.

- The first carry out rule processes the room description heading rule. Unless the visibility count is 0 (meaning the protagonist can't see a thing), then the visibility ceiling is printed.

- The second carry out rule processes the room description body text rule. This essentially does nothing more than print out the author-supplied textual description of a room.

- The third carry out rule processes the room description paragraphs about objects rule. What this does is handle the actual printing of the room's locale: meaning all those objects that are situated inside this room.

- The fourth and final carry out rule processes the check new arrival rule. This basically makes sure that a room is marked as visited.

The report rule is not something I'm going to worry about here; it essentially just has to do with what might be printed if other characters are in the room and they respond to a 'looking' action.

Room Name and Description

The 'looking' action begins by printing the name and description of the room the player is in. This is information that you, as author, provide as part of your source text. Let's consider a snippet of the source text you already have in place:

SOURCE TEXT

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

Here I've implicitly named the room (Living Room) by the assertion and I have explicitly provided how the room should be described, by asserting the description text for the room. What I've actually done is give two specific pieces of information: properties that Inform uses called the printed name and the description.

STORY TEXT

```
Living Room                          ← printed name
This is the Living Room description.  ← description
```

It may seem more logical to you that 'printed name' should really be 'room name'. However, Inform uses the notion of a 'printed name' for any objects in a story, not just rooms.

Knowing this information about the properties, it's possible to introduce variations into room (printed) names and descriptions by changing the printed name and description properties of a given room. To see this in action, add the following to your source text:

SOURCE TEXT

```
After looking in the Living Room:
  change the printed name of the Living Room to "Living Room (Modified)";
  change the description of the Living Room to "This is the (modified) Living Room description."
```

What this new source text does is add a rule that after the 'looking' action has taken place in the Living Room, the two properties of that room should be modified. Here's what this looks like in your story:

STORY TEXT

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

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

It's also possible to change either of these properties (or remove their display altogether) by altering the rules in the carry out looking rulebook. Remember that this rulebook is what handles how the printed name and the description are displayed after a 'looking' action is generated.

Before altering any rules, let's set up a situation where you have two rooms. Add the following to your current source text:

SOURCE TEXT

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

The Dining Room is east of the Living Room.
The description of the Dining Room is "This is the Dining Room description."
```

Room Names

Reiterating for a moment: in terms of the room name being printed, this happens during the carry out looking rulebook. One such rule in this rulebook is the room description heading rule. What this rule does is print the name of the visibility ceiling that I referred to earlier.

Let's remove that rule about printing the heading entirely. Add the following to your source text:

SOURCE TEXT

```
The room description heading rule is not listed in the carry out looking rules.
```

This is a sledgehammer effect in that since the rule about displaying the room description heading now no longer exists in the carry out looking rulebook. What this means is that *any* time the carry out looking rulebook is executed (as a result of a 'looking' action), the room's printed name (also called the room description heading) will not be printed.

Try It!
Put the above source in place and notice how the printed name for the "Living Room" no longer appears. Move to the east and you'll see the printed name for the "Dining Room" also does not appear. This change affects every single room in the story.

There is another way to achieve the exact same effect. The act of printing the name (of something) is an activity in Inform. So to remove the name of the room entirely, you could override how this activity works by having it do nothing at all. Make the following modification to your source text:

SOURCE TEXT

```
~~The room description heading rule is not listed in the carry out looking rules.~~
Rule for printing the name of a room: do nothing.
```

This is a blanket rule that overrides the activity in all cases. In other words, any time the printing the name (of something) activity executes and the "something" is a room, then nothing will be printed.

Look Closely.
Notice with the above how I just had you step through achieving the same effect, but at different levels of implementation. One involved the removal of a rule; the other involved overriding an activity.

If you just wanted to achieve the 'no room name displayed' effect for one particular room, the above would clearly not be your best approach. One nice thing about the 'override activity' technique is that it paves the way for a more granular way to handle things. For example, you can make the condition more isolated for a particular room, like this:

SOURCE TEXT

```
Rule for printing the name of Living Room: do nothing.
```

Now instead of overriding the activity for any room, this new statement overrides the activity for just one particular room (the Living Room). Any other room (like the Dining Room) would continue to display as normal.

Room Descriptions

The above examples handle getting rid of the room name. Getting rid of the room description itself in a 'sledgehammer' fashion is just as easy as with the room name. However, the granularity part is a bit more difficult given the nature of how room descriptions are constructed. For example, one part of the room description is called the "locale" and is constructed based on the items that are located in the room. The idea of the "locale" is something I'm going to be discussing more at length later in this guide. It's actually a fairly huge topic and will be the main part of my focus after we get through some preliminaries with the name and description aspects of the room.

For now, one way to get rid of the all room descriptions is by realizing that during the execution of the carry out looking rulebook, one of the rules is the room description body text rule. You can remove that rule from the rulebook by adding the following:

SOURCE TEXT

```
The room description body text rule is not listed in the carry out looking rules.
```

Try It!
Put the above source in place and notice how the description for the "Living Room" no longer appears. Move to the east and you'll see the description for the "Dining Room" also does not appear. As with removing the room name using the same technique, this is a global change that affects all rooms.

There is an extra complication here and it involves that term "locale" that I've thrown about a few times. The locale is part of the description, but it's a part that Inform automatically provides. I don't want to get too mired in this yet, but I do want you to have some context for what I mean. Before we made all these modifications, here's what your output looked like:

STORY TEXT

```
Living Room                          ← printed name
This is the Living Room description.  ← description

You can see a raised floor (on which is an alcove (in which is a chair (empty))) and a
table here.                          ← locale
```

So with rules in place that remove the room description body text rule, you can see that your output is now this:

STORY TEXT

```
Living Room

You can see a raised floor (on which is an alcove (in which is a chair (empty))) and a
table here.
```

Look Closely.
Notice how the initial Living Room description you provided is removed. However, this had no effect on Inform telling you about an item that you placed in that room.

And, in fact, if you kept in the rule that removed the room description heading rule (or the override of the printing the name (of something) activity), you wouldn't even see the room's printed name in the above output. In all cases, though, you'd be seeing the locale information.

The fact that objects have been placed in the room means that Inform doesn't just have to consider your room name or your room description. It also has to consider what Inform refers to as the "locale," which is made up of the objects that are in that room. This is one reason why location (room) descriptions are not as simple as location (room) names. This is the part that I'm going to be going over quite a bit in this guide since the room description itself is largely static in that you, as author, decided what to provide as as a textual description. The locale, however, is something that Inform generates as part of its normal operation.

Before getting into that however I want to say that another reason why location (room) descriptions are potentially not as simple is that the level of description provided is partly up to the person interacting with the story. By way of example, let's consider another way to get rid of the room descriptions that simulates the way a reader could do it. Specifically you can force your story to use what are called 'abbreviated' room descriptions.

Add the following to your source text, somewhere near the top:

SOURCE TEXT

```
Use abbreviated room descriptions.
```

This forces the story to be in what's called "superbrief" mode, which essentially never prints out the room descriptions. Unlike the situation above where you removed a rule from the carry out looking rulebook, the reader is free to override this abbreviated view of descriptions. Let's consider how they would do this.

The reader is free to use a 'verbose' command (which Inform translates to an 'preferring unabbreviated room descriptions' action) and a 'brief' command (a 'preferring sometimes abbreviated room descriptions' action). Both of these will override that initial setting above. There is a 'superbrief' command (a 'preferring abbreviated room descriptions') that the player can use as well, which would then take the story back into that initial setting.

This next part is a bit of a digression that I'm not entirely sure I want to get into but in the interest of completeness I'll say this: should you want to make sure that this level of control is not possible for the reader (in terms of changing your initial settings), you could remove certain rules that allow the reader this sort of control over the room descriptions. With the above declaration about using 'abbreviated room descriptions' still in place, add the following to your source text:

SOURCE TEXT

```
The prefer unabbreviated room descriptions rule is not listed in the carry out preferring unabbreviated room descriptions rulebook.

The prefer sometimes abbreviated room descriptions rule is not listed in the carry out preferring sometimes abbreviated room descriptions rulebook.

The standard report preferring unabbreviated room descriptions rule is not listed in the report preferring unabbreviated room descriptions rulebook.

The standard report preferring sometimes abbreviated room descriptions rule is not listed in the report preferring sometimes abbreviated room descriptions rulebook.
```

Try It!
Put the above source in place and then try to use the commands VERBOSE and BRIEF. You should find that they don't work and that's because the source above removes the rules that carry out those commands and reports the results of those commands.

All of what I talked about above handles the binary decision of not showing the room descriptions at all. As with the room name, however, you can selectively modify the descriptions.

You've already seen how it's possible to change the entire room description (by using the 'change the description of' phrase). There is, however, a lot of narrative possibility when you consider that you can change the room description based on certain conditions being true or false. That change to the description can either be in its entirety or selectively.

Let's consider a few examples of how this works. First, let's get that source text of ours down to a minimum. Change your source text so it looks like this:

SOURCE TEXT

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

The Dining Room is east of the Living Room.
The description of the Dining Room is "This is the Dining Room description."
```

Now, as an example of modifying the room description in part, let's say you want to change the description of a room depending on whether the protagonist has been in that room before. To do that, replace the existing Living Room description:

SOURCE TEXT

```
The description of the Living Room is "[if unvisited]This is the 'unvisited' Living Room description. [otherwise]This is the 'visited' Living Room description. [end if]This sentence shows up every time."
```

Here is how this plays out:

STORY TEXT

```
Living Room
This is the "unvisited" Living Room description. This sentence shows up every time.

> LOOK
This is the "visited" Living Room description. This sentence shows up every time.
```

As you can see, one part of the room description remains consistent but another part changes based on a certain condition, which in this case is whether the room has previously been visited before.

Look Closely.
Notice how the above interaction does not have the protagonist leaving the room and yet the room description still changes. That may seem odd to you if you think of 'visited' as having been to the room once and then returning. However, Inform keeps track of a visited property for each room that is only dependent upon a successful 'looking' action taking place.

There are other ways that can modify the room description. For example, perhaps you want to modify the description based on a specific number of times the protagonist has visited the room. Or perhaps you want a modified room description based on whether *another* room has been visited by the protagonist. Below is example of both situations. Type in the source text below in its entirety:

SOURCE TEXT

```
Use full-length room descriptions.

The Living Room is a room.
The description of the Living Room is "[if the Dining Room is visited]This is the 'Dining Room visited' Living Room description. [otherwise]This is the 'Dining Room unvisited' Living Room description. [end if]This sentence shows up every time."

The Dining Room is east of the Living Room.
The description of the Dining Room is "[if the player is in the Dining Room for the third time]This is the 'third time' description of the Dining Room. [otherwise]This is the 'first time' and 'second time' description of the Dining Room. [end if]This sentence shows up every time."

Test me with "east / west / east / west / east".
```

Look Closely.
Notice the 'Use full-length room descriptions' statement. With what you read about earlier, this corresponds to the VERBOSE command, or what Inform calls 'unabbreviated room descriptions'. I'm using this here to make sure that the room descriptions – which do change – are always visible to the reader so that they are more likely to see those changes.

One thing you'll note about the above source text is the "test me" statement at the bottom of the text. This will come in useful at various points in this guide so let me just take a second to talk about that.

A 'test' statement provides you, as author, with a way to set up a test scenario as part of your source text. The idea is that by typing in a 'test' command, you can execute that test scenario as part of the story to see how the story operates. Use of this particular technique will help me avoid putting too many "STORY TEXT" examples in this guide since you can simply run a provided "TEST ME" command to see the results. That being said, when I think there relevant aspects to point out within the source text, I will still produce some "STORY TEXT" sections for you.

In fact, let's try this out right now. With the above source text in place, if you execute a 'test me' command, you'll see the following:

STORY TEXT

```
Living Room
This is the "Dining Room unvisited" Living Room description. This sentence shows up
every time. [Notice how this description says "Dining Room unvisited" ...]

> TEST ME
(Testing.)

> [1] EAST

Dining Room
This is the "first time" and "second time" description of the Dining Room. This
sentence shows up every time. [First Dining Room description]


> [2] WEST

Living Room
This is the "Dining Room visited" Living Room description. This sentence shows up
every time. [Notice how this description now says "Dining Room visited" ...]

> [3] EAST

Dining Room
This is the "first time" and "second time" description of the Dining Room. This
sentence shows up every time. [Second Dining Room description; same as the first.]


> [4] WEST

Living Room
This is the "Dining Room visited" Living Room description. This sentence shows up
every time.

> [5] EAST

Dining Room
This is the "third time" description of the Dining Room. This sentence shows up every
time. [Third Dining Room description; different from first or second.]
```

Go through the above source text and make sure you understand what is happening at each step. Also make sure you understand *why* certain things are happening when they happen. This kind of analysis into Inform output will serve you in good stead when you start dealing with the locale and when you start to use extensions that allow you to modify the output that Inform generates.

Let's consider another example of room description modification. You can change the description of a room when the player has performed some specific action, such as examining a particular item. To see this, modify your source text as below.

SOURCE TEXT

....
The description of the Living Room is "[if we have examined the table]This is the 'table examined' Living Room description. [otherwise]This is the 'table unexamined' Living Room description. [end if]This sentence shows up every time."

....
A table is in the Living Room.

Test me with "examine the table / look".

(Note that you've modified the textual description of the living room, added a table object, and modified your test statement.)

Try It!
With the above source text in place, make sure to run the TEST ME command. Notice how the room description changes based on the action that is taken.

In case you're wondering – as I did – the "we" in the above text substitution for the Living Room is trying to be generic about exactly who did the examining. In other words, this could refer to the player character (protagonist) but it could also refer to any non-player character. What's being checked is if a particular action has taken place with a given object – regardless of who performed the action. That's when Inform seems to require the 'we'.

I bring this up because it contrasts with my previous example a bit. Let's consider the two statements:

- if we have examined the table ...

  vs.

- if the player is in the Dining Room for the third time ...

The reason you could use "if the player is in the Dining Room" (rather than, say, "if we are in the Dining Room") is because that doesn't refer to an action, but rather to Inform's notion of 'containment'. In Inform, containment history is handled differently than action history.

If this seems odd to you, you're not alone. This is an area – and there are a few – where Inform doesn't necessarily seem to take a very user friendly path in terms of presentation. I've found that this is usually because (1) Inform is exposing its implementation details and/or (2) Inform made design decisions based on its underlying implementation model.

Room Object Descriptions

After handling the room name and the room description, Inform determines what objects in that room are visible to the protagonist and need to be described. By default, these items never include the protagonist or items that are asserted to be 'scenery' in the source text. With those exceptions in mind, any other items in the location will have their marked for listing property set to true by Inform. That means literally what it sounds like: Inform has decided that those items should be listed as part of the entire room description. This is also the stage at which Inform chooses the order in which those items will be listed.

You can modify the ordering that Inform chooses by changing the priorities of the objects. You would probably do this if you wanted some objects to be described to the reader first or last in the room description. Beyond the ordering, you can also force items to be left out of the description entirely, essentially taking them off of Inform's "marked for listing" list.

What I'm going to do here is provide you with a very quick summary of ideas regarding how object descriptions are handled. This is, as mentioned before, dealing with the "locale" and it's all material that I'm going to revisit in the next section.

I want to get our source text to a place where I can have you build off some simple ideas, so let's start with this completely new set of text:

SOURCE TEXT

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

A table is in the Living Room.
A couch is in the Living Room.
An armchair is in the Living Room.
```

This is what you should see in your output with the addition of these objects to the Living Room:

STORY TEXT

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

You can see a table, a couch and an armchair here.
```

So Inform essentially gave the room description that you provided for the Living Room and then gave a separate paragraph that detailed each item that you, as author, indicated was in the Living Room.

Look Closely.
It's important to note that the "locale" portion of the room description is in a *separate paragraph*. This is very important because the first paragraph is the one you directly provide as part of authoring the source text but the second paragraph is generated by Inform based on what you indicate is part of the room.

Anything that Inform prints the name of gets a certain mentioned property set. In the above case, the table, couch, and armchair would all be considered as mentioned. The fact that a paragraph was written about each item is enough for Inform to consider each "mentioned." That being said, note how while a paragraph was given for the items, each item did *not* get its own separate paragraph. This distinction matters, as you will see.

After setting up the "marked for listing" items and a priority ordering scheme for the items that are marked for listing (the latter of which I have not discussed yet), Inform then looks for any rules in your source text that provide a specific writing a paragraph about activity for a specific item. If any such rules exist, Inform then writes out the paragraph of information that you provide. Just to put this in context, add the following rule to your source text:

SOURCE TEXT

```
Rule for writing a paragraph about the table:
    say "Specific paragraph about the table."
```

With this in place, here is the output:

STORY TEXT

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

Specific paragraph about the table.

You can also see a couch and an armchair here.
```

Look Closely.
Notice with the above that the table now has its own paragraph, leaving the other two items to be combined into one generic paragraph. This idea of how and when Inform generates separate paragraphs for items is a *very important* one in terms of how your story presents location descriptions.

Now let's change that rule a little bit. Modify the source text so that the rule for writing a paragraph about the table reads as such:

SOURCE TEXT

```
Rule for writing a paragraph about the table:
    say "Specific paragraph about the table. This paragraph also mentions the [couch] and the [armchair]."
```

The output to this is:

STORY TEXT

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

Specific paragraph about the table. This paragraph also mentions the couch and the
armchair.
```

What the text substitutions did was make sure that the couch and the armchair are "mentioned" along with the table. What that means is that Inform will not refer to them again through the rest of the room description, including the locale.

Look Closely.
Contrast the above response with the one you got with the previous version of the rule. Notice how this time there was no 'generic' paragraph (for the couch and the armchair) along with the specific paragraph (for the table). This time all items in the location were given a specific paragraph.

I've just shown you a couple of variations and I want to make sure those are clear. Here's a table that summarizes what you've seen in terms of the type of output.

Fully Generic Paragraph	You can see a table, a couch and an armchair here.
One Specific Paragraph, One Generic Paragraph	Specific paragraph about the table. You can also see a couch and an armchair here.
Fully Specific Paragraph	Specific paragraph about the table. This paragraph also mentions the couch and the armchair.

I don't want to belabor the point too much here about specific and generic paragraphs since I'll have further to say about that as I go on with further examples. I do, however, want to make sure that these concepts have at least been presented to you.

After handling any specific rules dealing with the writing a paragraph about activity, Inform then prints the initial appearances of items that are "marked for listing" *and* that are not marked as "mentioned." To see this in context, add the following source text:

SOURCE TEXT

```
An armchair is in the Living Room. "Initial appearance for the armchair."
A fireplace is in the Living Room. "Initial appearance for the fireplace."
```

What you are adding here is an initial appearance for the armchair. You're also adding a fireplace, which will also have an initial appearance.

Here's the output:

STORY TEXT

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

Specific paragraph about the table. This paragraph also mentions the couch and the
armchair.

Initial appearance for the fireplace.
```

Note that even though the armchair now has an initial appearance, it already gets flagged as "mentioned" (due to the 'writing a paragraph about the table' rule) and so only the fireplace gets its own separate mention in a separate paragraph.

Try It!
Remove the text substitution for the armchair in the 'rule for writing a paragraph about the table'. Then execute the story and you should see something a little different.

When doing this, make sure you are starting to understand how these separate paragraph mechanisms work. If you do try this, make sure to put the text substitution for the armchair back in the rule.

After the initial appearances for items are handled, Inform then lists any visible, non-scenery items that sit on scenery supporters. That may sound a little obtuse at first glance so here's an example of what you can add to the source text to check this out:

SOURCE TEXT

```
A mantel is in the Living Room.
The mantel is a supporter and scenery.
[On the mantel is a newspaper.]
```

With these changes alone, the room description will be unaltered from the last example. The mantel, being scenery, will not be mentioned. If, however, something worth mentioning were to be on the mantel, the situation would be different. To show what I mean, uncomment the assertion about the newspaper and you'll see that your output now is different:

STORY TEXT

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

Specific paragraph about the table. This paragraph also mentions the couch and the
armchair.

Initial appearance for the fireplace.

On the mantel is a newspaper.
```

In this case, the mantel is still not listed but the newspaper is, because the newspaper – unlike the mantel – is not scenery. However, since the newspaper is listed, the mantel also gets mention because it indicates the situational aspect of the newspaper (i.e., being supported by the mantel).

Something To Keep In Mind
These latter kind of situational paragraphs can be manipulated with the printing the locale description activity. This is something I'll be coming back to later.

After all of the above handling of items, Inform then collates the remaining items (which are considered "non-descript") into a final paragraph that sums up everything in the room. You've already seen this in action. Any time Inform prints a "You can see ..." or a "You can also see ..." paragraph as part of a room description, you are essentially getting a listing of the non-descript items in that room. Here "non-descript" simply means that there was no descriptive element to these items at all, at least in terms of any initial appearances or rules saying otherwise.

Something To Keep In Mind
You can preempt items from appearing in these non-descript paragraphs or you can change how items show up in this list. This involves the listing nondescript items activity. You can also use this activity to replace the "You can also see..." wording with something else that you find more appropriate.

I should note that when Inform compiles the list of what it considers nondescript items, it adds 'situational tags' such as "(open)" or "(empty)" or "(on which is a newspaper)" to the names of containers and supporters. To see this, add the following to your source text:

SOURCE TEXT

```
A wall mount is in the Living Room.
A candle is on the wall mount.

A trash can is in the Living Room.
The trash can is a container.
```

The output then looks like this:

STORY TEXT

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

Specific paragraph about the table. This paragraph also mentions the couch and the
armchair.

Initial appearance for the fireplace.

On the mantel is a newspaper.

You can also see a wall mount (on which is a candle) and a trash can (empty) here.
```

Some people like this kind of thing, others don't. It really depends on how much you think these situational qualifiers interfere with the reading of the room description. Since this is often an issue for people encountering this for the first time, I want to at least briefly talk about how to deal with these tags just to give confidence that it is possible. Later sections will go into more detail.

You can suppress or change the "(empty)" tag with the printing room description details of activity. For example, you could add the following to your source text:

SOURCE TEXT

```
Rule for printing room description details: stop.
```

That rule would do this for *every* item in the room that had such 'room description details.' (And, in fact, this would happen for *every* item in *every* room.) You can make the rule more specific as well. First, let's add another object that matches the characteristics of the trash can:

SOURCE TEXT

```
A box is in the Living Room.
The box is an openable container.
```

Try It!
Notice how you now have the trash can and the box in the Living Room. With the 'stop' edict in place for displaying room description details, neither item has the qualifier "(empty)" after it.

Now to make that rule more specific, replace it with this:

SOURCE TEXT

```
Rule for printing room description details of the trash can: stop.
```

Here you would find that only the trash can has its "(empty)" qualifier removed while the box's qualifier remains. In fact, to give you some idea of the power of Inform's language, you can even be more explicit with the rule while generalizing a bit. Change the above rule to this:

SOURCE TEXT

```
Rule for printing room description details of a closed container: stop.
```

Try It!
Here the box would show the "(empty)" qualifier tag until a CLOSE BOX command was given.

I don't want to get too far off topic here. So as a reminder of how we got to where we are: I was showing you how you could remove one element of qualifier, which is the room description details for a given item in that room. You can also suppress the "(open)" and "(on which is...)" types of situational qualifiers by telling Inform to omit the contents of particular items. This will happen in the context of a rule. Here's an example you can add to your source:

SOURCE TEXT

```
Rule for printing the name of the wall mount:
  say "wall mount";
  omit contents in listing.
```

With this in place, Inform would simply print the name of the wall mount without the qualification "(on which is a candle)".

Something to note with this is that this rule is overriding what Inform would normally do – which is printing the name of the wall mount. The fact that the wall mount's contents are listed is a secondary part to that. What that means is that you do have to indicate what the name of the item is, which is why I have the 'say' phrase in place in the rule. The 'omit contents in listing' phrase is what prevents the contents of the wall mount from being listed as part of the name and that's great – but if you didn't have the 'say' phrase in place, the naming of the wall mount would also be overridden.

Structuring a Locale Description

Before continuing I'm going to present a whole slew of source text that you should put in place.

SOURCE TEXT

```
A thing can be large or small.
A thing is usually small.

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

A table is in the Living Room.
[The initial appearance of the table is "This is the initial appearance of the table."]
The table is large.

A wooden chair is an enterable supporter in the Living Room.
[The initial appearance of the chair is "This is the initial appearance of the chair."]
The chair is large.

Jones is a man on the chair.
Persuasion rule for asking people to try doing something: persuasion succeeds.

A device called the alarm clock is in the Living Room.
A device called the standing lamp is in the Living Room.
The standing lamp is switched on.

A plastic box is in the Living Room.
[The initial appearance of the plastic box is "A simple box."]
The plastic box is closed, openable and transparent.
In the plastic box are a rubber chicken and a light bulb.

A high shelf is a thing in the Living Room.
On the high shelf is a jug of mayonnaise.

A service alcove is an enterable container in the Living Room.
The service alcove contains a waiter, a food tray, and a bottle of fresh water.
The waiter is a man.

A coat rack is scenery in the Living Room.
On the coat rack is a jacket.
```

At this point you've seen that when handling a LOOK command (and thus a 'looking' action), Inform will normally print the room name and the room description, both of which are followed by what's called a locale description for the room. I've touched on this subject here and there but now it's time to get a little more in depth.

A **locale description** is a segment of the text produced by a LOOK action. The **locale** is a set of objects at a given level in the object tree. You've already seen that most room descriptions consist of a top line indicating the name of the room, followed by an author-provided description of the room, and then a single (though often, as below, multi-paragraph) locale:

STORY TEXT

```
Living Room     ← first the room name
     ... now the room description ...
This is the Living Room description.
     ... now the locale for the Living Room ...
On the coat rack is a jacket.

You can also see a table, a wooden chair (on which is Jones), the alarm clock, the
standing lamp, a plastic box (closed) (in which are a rubber chicken and a light
bulb), a high shelf (on which is a jug of mayonnaise) and a service alcove (in which
are a waiter, a food tray and a bottle of fresh water) here.
```

A locale typically contains a sequence of paragraphs specific to "interesting items" – especially those not yet picked up by the protagonist – followed by a paragraph which lists the "non-descript" items, meaning those not given paragraphs of their own, such as the table and the wooden chair, among all those others that follow the phrase "You can also see..." in the story text above.

Some items – typically scenery, but also the object representing the protagonist – are not even considered non-descript and thus do not appear at all. So in the above output, the coat rack (which is scenery) is actually not mentioned, but a jacket is on the coat rack. Since the jacket is mentioned, the coat rack is as well.

Try It!
Comment out the jacket from the source text and you'll find any mention of the coat rack is removed from the locale description. If you do this, just make sure to uncomment the jacket again.

It's certainly possible for a locale to contain no interesting items (meaning no separate paragraphs per item) or no non-descript items (meaning no single paragraph for a lot of items). A locale can even contain neither type of item: that is, the locale can be entirely empty.

When the protagonist is in or on something, multiple locales are described:

STORY TEXT

```
Living Room (in the service alcove)     ← first the room name
     ... now the room description ...
This is the Living Room description.
     ... now the locale for the Living Room ...
On the coat rack is a jacket.

You can also see a table, a wooden chair (on which is Jones), the alarm clock, the
standing lamp, a plastic box (closed) (in which are a rubber chicken and a light
bulb), a high shelf (on which is a jug of mayonnaise) here.
     ... now the locale for the service alcove ...
In the service alcove you can see a waiter, a food tray and a bottle of fresh water.
```

To sum up, the text produced by a looking action consists of a header (produced by the carry out looking rules) followed by one or more locale descriptions, which are produced by an activity called printing the locale description.

When describing a locale, Inform essentially stores an internal table of interesting objects, and all of those objects are associated with a priority. The priority is nothing more than a number indicating how important the object is. Here "important" refers to how near to the top of the description the object is in terms of being listed. This is handled by the printing the locale description activity and that's what I'll discuss next.

So let's recap a bit. A "locale description" is Inform jargon for the part of a room description which catalogs the visible items in that room. When a LOOK command (and thus looking action) takes place, Inform will normally print the description of the room itself, followed by a locale description for the room. But, as shown earlier, if the protagonist is in or on something – like in the service alcove or on the chair – there will be two locale descriptions: one for the room itself, then another for the container or supporter.

So what this means in an operational sense is that the printing the locale description (of something) activity is used to write the locale description for a single "domain." That domain equates to the "of something" and the "something" can be either a room, an enterable container, or an enterable supporter.

Here's a breakdown of how this activity works.

- The before stage works out which items in the domain might be "interesting."
- The for stage actually prints paragraphs about those items.
- The after stage initially does nothing, but can be used to insert extra information into a room description.

I'll give a high-level overview regarding the before and for stages of the activity and then I'll go back and cover the details in more depth.

One of the before rules for the printing the locale description (of something) activity is called the find notable locale objects rule. All this rule does run the choosing notable locale objects activity. The task of this latter activity is to identify the items which might, by virtue of their location in the domain, appear in the locale. Then each of those items is assigned a priority number. (By default, the "notable objects" are exactly the children of the location, and they all have equal priority, which is 1.)

The for rule of the printing the locale description (of something) activity is called the interesting locale paragraphs rule and what this does is goes through all of the "notable objects" chosen at this stage (from the choose notable locale objects activity), in order of priority, and offers each to yet another activity: the printing a locale paragraph about activity. This latter activity can either print a paragraph related to the item in question, or demote it as being not even non-descript (and thus never mentioned), which is done by changing the priority of the item to 0. The default result of this activity is to do nothing, in which case the item becomes non-descript.

So that's the overview. Now let's dig in a bit.

Printing the Locale Description (of something)

What you've probably picked up from the overview above is that the default behavior of the printing the locale description activity is a bit involved. Briefly, though, the first thing that happens is execution of the choosing notable locale objects activity to find out what ought to be mentioned as part of the locale for a given location (room). That activity assembles a list of items to mention, sorted into priority order. Items with priority 1 go first, then those with priority 2, and so on.

Choosing Notable Locale Objects (for something)

This activity is expected to decide which items ought to be mentioned in a locale description for a given "domain" (room, enterable container or enterable supporter), and to give each item a priority, which is a number ranging upwards from 1 (which is the top priority). The lower the priority number, the earlier the mention, or at least, the earlier the *opportunity* to be mentioned: it's up to other activities whether to actually give the item a paragraph of its own or not.

Something To Keep In Mind
The 'choosing notable locale objects' activity only makes an item (object) a possible candidate for including in a locale description and then decides what order the candidates will attempt to be listed in. The 'choosing' activity does nothing with that information, however.

By default, this activity contains only the standard notable locale objects rule. This rule has logic that chooses exactly those items directly contained by the locale, assigning all of them priority 5. Note that this includes scenery, and other (possibly unwanted) items. Those other items will be excluded later.

To show how this works, add the following to your source text:

SOURCE TEXT

```
Before printing the locale description (this is the dump locale table rule):
  say "Locale Priority List";
  repeat through Table of Locale Priorities:
    if notable-object entry is mentioned:
      let flag be true;
    otherwise:
      let flag be false;
    say "[line break] [notable-object entry]: [locale description priority entry]";
    if flag is false, change notable-object entry to not mentioned;
  say line break.
```

With this in place, you'll get a list showing you the priority of the items. You should see something like this:

STORY TEXT

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

Locale Priority List
 yourself: 5
 table: 5
 wooden chair: 5
 alarm clock: 5
 standing lamp: 5
 plastic box: 5
 high shelf: 5
 service alcove: 5
 coat rack: 5

On the coat rack is a jacket.

You can also see a table, a wooden chair (on which is Jones), the alarm clock, the
standing lamp, a plastic box (closed) (in which are a rubber chicken and a light
bulb), a high shelf (on which is a jug of mayonnaise) and a service alcove (in which
are a waiter, a food tray and a bottle of fresh water) here.
```

Now let's modify the list a bit by putting some items in a specific order. Remember that items with low priority numbers list towards the beginning and items with high priority numbers list towards the end. Conceptually it might help you to think in terms of making a numbered list of the paragraphs to appear in the description, keeping in mind that any item numbered 0 doesn't appear at all.

Add the following to the source text:

SOURCE TEXT

```
After choosing notable locale objects:
  set the locale priority of the plastic box to 1;
  set the locale priority of the standing lamp to 0;
  set the locale priority of the wooden chair to 10.
```

Try It!
Make sure to put the above change in place and try it out. You'll notice that the priorities have changed, but the listing of the objects has not.

This brings up an important cautionary note: priorities are only honored if the items are going to get their own paragraphs, either because of a writing a paragraph about activity for that item or because those items they have initial appearances. (You'll hopefully remember that these are situations I showed you earlier.)

Priorities do not affect the order in which items appear in the final "You can see..." list, except that items with priority 0 or lower are omitted entirely. So you'll notice that the standing lamp no longer appears in that list with the above source text in place because it was given a priority of 0. Other than that, however, nothing has changed.

Try It!
Uncomment the initial appearance statements for the table, wooden chair and the plastic box. Then rerun the story. Now you'll see how the ordering of items changes. Since the wooden chair is set to 10, it will be listed after everything else. (Or, at least, after everything else that is not "nondescript.") Since the standing lamp is given a priority of 0, it won't be listed at all. Since the plastic box is set to 1, it will be listed before everything else.

Now let's remove the after rule that you just put in place for 'choosing notable locale objects.' Also comment out the initial appearance statement for the plastic box but leave everything else as it is. Once that's done, add this source text:

SOURCE TEXT

```
Rule for choosing notable locale objects for the Living Room:
  repeat with item running through large things in the Living Room:
    set the locale priority of the item to 0;
  repeat with item running through small things in the Living Room:
    set the locale priority of the item to 5.
```

The special phrase 'set the locale priority' makes the given items a candidate and sets their priority. Setting a priority to 0 forces an item not to be a candidate and thus not to be included. That's why, with this code in place, you no longer see the table or the chair mentioned. This is the case even though the table and chair have their initial appearance set.

Note that the second repeat phrase above is meant to have the activity do what it would have done otherwise. I had to do this because by putting this rule in place I had effectively replaced what would have normally happened. So that second repeat phrase just puts "what normally would have happened" back in place. With that being said, I could have made life a little easier on myself if I had just done this instead:

SOURCE TEXT

```
Rule for choosing notable locale objects for the Living Room:
  repeat with item running through large things in the Living Room:
    set the locale priority of the item to 0;
  continue the activity.
```

At this point make sure your source text is back to the Reference Configuration 2.

Now let's say that as part of my story, I didn't want the service alcove to be visible unless the lamp was switched on. Add the following story logic:

SOURCE TEXT

```
To decide whether the light level is high:
  if the standing lamp is switched off, no;
  if the player cannot see the standing lamp, no;
  yes.

To decide whether the light level is low:
  if the light level is high, no;
  yes.

After choosing notable locale objects:
  unless the light level is high:
    set locale priority of the service alcove to 0.

Test me with "turn off the lamp / look".
```

Keep in mind that the lamp, as a device, defaults to being switched on. So with the above code in place, you can run the TEST ME command and see what happens.

Look Closely.
When you run the TEST ME command, notice the difference in the "non-descript" item output: when the lamp is off, the service alcove is not mentioned at all.

Printing a Locale Paragraph About

After the choosing notable locale objects activity, the printing a locale paragraph about activity is run for each candidate item.

By this point, the locale description process has identified a number of items as candidates to be described, and worked out a priority order for those items. So now a printing activity is called for each candidate in turn, starting with the highest priority items and working downwards. This activity can either print some text or not, and can either mark the item as "mentioned" or not. If the item is marked as "mentioned," then that item won't appear subsequently in the locale description. (You'll hopefully remember being introduced to this concept earlier.) If this printing activity does nothing, the item becomes "non-descript" and falls through into the final "You can also see..." paragraph, unless another rule mentions the item in the mean time.

By default there are four kinds of "interesting" locale paragraph. The printing activity is run on each notable thing in turn, in priority order.

The basic principle is that, at every stage, Inform should consider an item only if it is not "mentioned" already. This will happen if the item has been named by a previous paragraph, but also if the item has been explicitly marked as such to so as to make sure it isn't listed. In considering an item, Inform has four basic options:

1. **Print a paragraph about the item and mark it as mentioned.** This is good for interesting items deserving a paragraph of their own.

2. **Print a paragraph, but do not mark it as mentioned.** This is only likely to be useful if you want to print information related to the item without mentioning the item itself. (For instance, if the presence of a mysterious parcel resulted in a ticking noise, you could print a paragraph about the ticking noise without mentioning the parcel, with the idea that the parcel would then be mentioned later.)

3. **Print no paragraph, but mark the item as mentioned.** This gets rid of the item, ensuring that it will not appear in the final "You can also see..." sentence, and will not be considered by subsequent rules.

4. **Do nothing at all.** The item then becomes "non-descript" and appears in the final "You can also see..." sentence, unless something else mentions it in the mean time (such as via a text substitution, which I showed in earlier examples).

The default behavior of how all this works is provided by a sequence of seven rules.

1. The don't mention player's supporter in room descriptions rule excludes anything the player is directly or indirectly standing on or, less frequently, in. The logic behind this is that the header of the room description has probably already said something like "Living Room (in the service alcove)", so the player is unlikely to be unaware of this item.

2. The don't mention scenery in room descriptions rule excludes scenery.

3. The don't mention undescribed items in room descriptions rule excludes the protagonist (player character) object. (If this didn't happen, the interactor would see something like "You can also see yourself here.") At present nothing else in Inform is "undescribed" in this sense.

4. The set pronouns from items in room descriptions rule adjusts the meaning of pronouns like IT and HER to refer to items mentioned. Thus if a room description ends "In the service alcove you can see a waiter, a food tray and a bottle of fresh water", then HIM would be adjusted to mean the waiter. Likewise, IT would refer to the bottle of fresh water.

5. The offer items to writing a paragraph about rule gives the writing a paragraph about activity a chance to intervene for any item that has such a rule in the source text. Inform detects whether this activity does intervene or not by looking to see if it has printed any text.

6. The use initial appearance in room descriptions rule uses the initial appearance property of an item which has never been handled (i.e., picked up by the protagonist) as a separate paragraph.

7. The describe what's on scenery supporters in room descriptions rule is used to print text such as "On the coat rack is a jacket." for items which, like the coat rack, are scenery. The logic here is that an assumption is made that scenery will be brought up in the main room description that's presented to the interactor. (An operating assumption of Inform is that scenery supporters make their contents more prominently visible than scenery containers, the latter of which do not have their contents announced automatically.)

So let's say you want to get rid of the "large things" from your room description. Put following code could in your source text:

SOURCE TEXT

```
For printing a locale paragraph about a large thing (called the item):
  set the locale priority of the item to 0;
  continue the activity.
```

Something To Keep In Mind
You've had a couple of chances to see this so far, but I'll call it out here: it's usually a good idea to "continue the activity" at the end of rules for certain activities to make sure that everything that is part of the activity takes place. Here it doesn't really matter, since I'm trying to stop anything from happening regarding the large things. That being said, it doesn't do any harm to include the statement either.

What you'll notice with the above text in place is that the table and the chair are no longer mentioned in the room description. This will be the case even if the initial appearance of the chair and the table are set.

Try It!
Uncomment the initial appearance assertions for the table and chair. You'll notice that both items are still not mentioned.

Suppose you want there to be some high shelves in your game, which the interactor can't get at unless they have the protagonist standing on another item of some kind. In order to set up this sort of situation, you'll want to set up a new kind of thing: a raised supporter. Then, when something is on a raised supporter, you can set up the situation so that the item should be mentioned to the interactor only if the protagonist is in the right position (i.e., standing on something) and otherwise omitted from the description entirely. To see what I mean, add the following to your source text:

SOURCE TEXT

```
For printing a locale paragraph about a raised supporter (called the high place):
  if the player is on a supporter (called the riser):
    say "Up on [the high place] (and only visible because you're on [the riser]) [is-are a list of things on the high place].";
  otherwise:
    say "The [high place] is above you."

Test me with "jones, stand up / stand on the chair / look".
```

Note that you should replace any TEST ME statement you already had in place with the one above. You might also note that with this example, I did *not* put a 'continue the activity' phrase in. The reason I'm not doing that is because I want to completely replace the normal behavior of describing what's on supporters.

Try It!
Make sure to execute the TEST ME command to understand how this is working.

Look Closely.
When you run the TEST ME command, notice the difference between the initial room description and the description you get after the protagonist is standing on the chair.

Here's something to keep in mind: if all that you require is to supply an interesting paragraph about some particular item then it's pretty much always better to use the writing a paragraph about activity and not the activity I've just been showing you. This current activity should only be used when the mechanism itself needs to be adjusted. In fact, in practice, this activity usually hands the job of printing over to the writing a paragraph about activity, which I'll look at next.

Writing a Paragraph About

This activity happens just *before* writing a paragraph about some item in a room description.

The default behavior is for this activity to do nothing. However, if a rule is supplied which prints something up, then this activity replaces the paragraph which would otherwise have been printed. Moreover, any items whose names are said in the course of this rule – for instance, by being listed – are then excluded from the remainder of the room description, because they are considered as having been described sufficiently already. Inform keeps track of which objects have already been named with an either/or property called "mentioned", which it assigns whenever the name of an object has been automatically printed.

At this point make sure your source text is back to the Reference Configuration 2. Then add to the source text:

SOURCE TEXT

```
Rule for writing a paragraph about Jones:
  say "Jones looks at [the wooden chair] and [the table]."

Test me with "jones, stand up / look".
```

There is a lot to notice with this particular example so here is the output you would see with the TEST ME command being executed:

STORY TEXT

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

On the coat rack is a jacket.

You can also see a table, a wooden chair (on which is Jones), the alarm clock, the
standing lamp, a plastic box (closed) (in which are a rubber chicken and a light
bulb), a high shelf (on which is a jug of mayonnaise) and a service alcove (in which
are a waiter, a food tray and a bottle of fresh water) here.

> TEST ME
(Testing.)

> [1] JONES, STAND UP
Jones gets off the wooden chair.

> [2] LOOK
This is the Living Room description.

Jones looks at the wooden chair and the table.

On the coat rack is a jacket.

You can also see the alarm clock, the standing lamp, a plastic box (closed) (in which
are a rubber chicken and a light bulb), a high shelf (on which is a jug of mayonnaise)
and a service alcove (in which are a waiter, a food tray and a bottle of fresh water)
here.
```

What the rule I added does is prevent the appearance of the subsequent text about the chair and the table. Notice the difference in the “You can also see...” paragraphs above. What you’ll see is that the table and wooden chair are not mentioned, since the paragraph about Jones already forced their mention. This is the case even if the table and chair were to have initial appearance text.

Try It!
Uncomment the initial appearance assertions for the table and the chair and then execute the source text again, making sure to run the TEST ME command.

Given how important this is, I want to show the output from the “Try It!” as well:

STORY TEXT

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

This is the initial appearance of the table.

This is the initial appearance of the chair.

On the wooden chair is Jones.

On the coat rack is a jacket.

You can also see the alarm clock, the standing lamp, a plastic box (closed) (in which
are a rubber chicken and a light bulb), a high shelf (on which is a jug of mayonnaise)
and a service alcove (in which are a waiter, a food tray and a bottle of fresh water)
here.

> TEST ME
(Testing.)

> [1] JONES, STAND UP
Jones gets off the wooden chair.

> [2] LOOK
This is the Living Room description.

Jones looks at the wooden chair and the table.

On the coat rack is a jacket.

You can also see the alarm clock, the standing lamp, a plastic box (closed) (in which
are a rubber chicken and a light bulb), a high shelf (on which is a jug of mayonnaise)
and a service alcove (in which are a waiter, a food tray and a bottle of fresh water)
here.
```

There's a subtlety to be aware of. Notice in the original example (before you uncommented the initial appearance of the chair), the first time the room description is printed, Jones is on the chair. So his name is not actually considered to be printed. Put another way, Jones doesn't get a separate paragraph so he is lumped in with the "non-descript" items and only mentioned as part of the chair, i.e., "a wooden chair (on which is Jones)". However, when Jones is not part of another item – i.e., when he is no longer on the chair – he then is mentioned in a separate paragraph.

Now, with the above example, since there was a specific writing a paragraph about regarding Jones, the character was given a specific paragraph. You'll also note that the first time the room description is printed (before Jones is told to stand up), the initial appearances of the chair and table are shown. This is even with the fact that Jones' description mentions the chair and the table. But, again, that description is suppressed since Jones is only initially described as part of the chair.

After Jones has stood up (and is thus no longer "part of" the chair), his own character description kicks in. Since this description does mention the table and chair, by text substitution, this overrides even the initial appearances of the chair and table.

Look Closely.
There's a lot going on with the above example source text and my descriptions of it are trying to cover a lot of detail. I recommend spending some time playing with those specific examples, making sure you understand how and when certain text is printed. Your ability to internalize these concepts of Inform and build up your intuition about what will and won't work will stand you in good stead as you craft your own descriptions.

Listing Nondescript Items (of something)

Let's take a quick second here to re-orient and make sure the thread of discussion keeps on track.

The printing the locale description activity uses the printing a locale paragraph activity for each candidate item in a room. The latter often hands control over to the writing a paragraph about activity for each item.

That latter activity seems like it would be the logical end point every time. And yet, sometimes a paragraph will indeed be written, but not always. Sometimes there's nothing interesting to say about a given item. That means that item is left until a final, single paragraph which gathers up the all such items and places them after a "You can also see ..." header. The printing of that last single paragraph is handled by the listing nondescript items of activity. Remember that as soon as any item gets its either/or property "mentioned" set – usually by having its name printed – that item is removed from the list of items to print paragraphs about so that it will not appear subsequently, regardless of its priority.

This activity prints up the "You can also see..." paragraph at the end of a room description. The items in this list, as you have seen, are referred to as non-descript. These are non-descript items because they don't merit paragraphs of their own. If, as sometimes happens, there are no such non-descript objects in the room, then no such paragraph is printed and this activity does not happen.

At this point make sure your source text is back to the Reference Configuration 2. Then add to the source text:

SOURCE TEXT

```
Before listing nondescript items:
  say "This sentence shows up before listing any non-descript items.";
  change the wooden chair to not marked for listing;
  change the table to not marked for listing.
```

The above text promotes the table and chair out of the non-descript category by unmarking them. This has the effect of both objects not being mentioned at all. However, it's important to note that this wouldn't take effect if the table and chair had an initial appearance. The reason is because the initial appearance check for items is done first before non-descript items are considered.

Remove the above before rule and replace it with this:

SOURCE TEXT

```
Rule for listing nondescript items of the Living Room:
  say "Strewn around ";
  list the contents of the Living Room, as a sentence, tersely, listing marked items only, prefacing with is/are, including contents and giving brief inventory information;
  say "."
```

With that source text in place, you'll see that I'm changing the normal phrasing of the non-descript paragraph. Instead of saying "You can also see...", the paragraph now says "Strewn around ..." Although I'm not going to get into a lot of detail about this right now, you can see from the source text that it's possible to change the listing style as well.

Modifying the Locale Description

A lot has been discussed in the previous pages about not just the **location** but the all-important **locale**. The best approach to learning the above material is really to play around with the concepts a bit, creating locations with lots of items, all with differing levels of implementation in terms of whether the items have initial appearances, are open containers, visible supporters, etc.

So there's nothing that beats actual practice. But here are some general heuristics that might guide your practice. If the effect you want to achieve can be had using writing a paragraph about and listing nondescript items of alone, just use those. That's often the easiest approach. However, if you find that it's necessary to dig in even more, then use choosing notable locale objects and printing a locale paragraph to alter the normal processes of how Inform generates the locale description. You should use the powerful printing the locale description activity only when the entire process needs to be altered, as opposed to the item-by-item manipulations.

I want to provide two (very simple!) examples of why the printing the locale description activity can be a powerful aspect of Inform to work with. Add the following to your source text:

SOURCE TEXT

```
Before printing the locale description of a room (called the setting):
  if the setting encloses the standing lamp and the standing lamp is switched on:
    say "Because the lamp is on, this text is first in the locale description."
```

With this in place, you'll see that the story text prints the text in the 'say' phrase after the author-provided Living Room description but before any locale information is printed. Now add the following to your source text:

SOURCE TEXT

```
Rule for printing the locale description of the Living Room:
  say "This is the (modified) locale description."
```

With the above in place you'll find that the entire locale description has been changed to the bit of simple text in the 'say' phrase. While that sort of "blunt force trauma" effect is probably not going to be

something you are generally seeking, it's good to know that it's possible and it perfectly illustrates why the printing the locale description activity is so powerful and why you might want to be judicious in its use.

Preparing a List

There is one technique I want to bring up that lets you order the items in the non-descript list. This involves using the Complex Listing extension, built in to Inform. Complex Listing is not specifically designed just for handling locale elements for location descriptions but it can be used that way.

At this point make sure your source text is back to the Reference Configuration 2. At the top of the source text, add the following statement:

SOURCE TEXT

```
Include Complex Listing by Emily Short.
```

Now let's arrange the list so that the items can be presented in an order different from their default. Add the following to your source text:

SOURCE TEXT

```
Definition: a thing is other if it is not the player.

Rule for writing a paragraph about something:
  say "Your eye is drawn to the large things: [a list of large things in the location]. ";
  prepare a list of the unmentioned other things in the location;
  say "Along with those you've got [a prepared list]."
```

Read It!
Make sure to look at the differences in output.

Original Output	New Output
On the coat rack is a jacket. You can also see a table, a wooden chair (on which is Jones), the alarm clock, the standing lamp, a plastic box (closed) (in which are a rubber chicken and a light bulb), a high shelf (on which is a jug of mayonnaise) and a service alcove (in which are a waiter, a food tray and a bottle of fresh water) here.	Your eye is drawn to the large things: a table and a wooden chair. Along with those you've got the alarm clock, the standing lamp, a plastic box, a high shelf, a service alcove and a coat rack. On the coat rack is a jacket.

I should note that the "Definition" statement in the source text makes sure that the "other things" used in the rule will not include the protagonist (i.e., player character object). Now let's say that I want to arrange the list so that the items in it are presented in order from shortest name to longest name, and separated by commas with no "and". To achieve this modify the rule you just added accordingly:

SOURCE TEXT

```
Definition: a thing is other if it is not the player.

Rule for writing a paragraph about something:
  say "Your eye is drawn to the large things: [a list of large things in the location]. ";
  prepare a list of the unmentioned other things in the location;
  order list by length;
  say "Along with those you've got [a prepared list]."
```

Read It!
Make sure to look at the differences in output of the list.

Original Output	Length-Ordered Output
Your eye is drawn to the large things: a table and a wooden chair. Along with those you've got the alarm clock, the standing lamp, a plastic box, a high shelf, a service alcove and a coat rack.	Your eye is drawn to the large things: a table and a wooden chair. Along with those you've got a coat rack, a high shelf, the alarm clock, a plastic box, the standing lamp and a service alcove.

What the phrases did in both examples is use the “prepare a list {of some description}” statement. This sets up a table, called the Table of Scored Listing, which contains all the items that are going to described in your list, in two columns. The first column is the output (the item that is going to be named) and the second column is the assigned “score” (meaning, the value you’ve given this item to order it with respect to everything else in the list). The “order list by {some phrase}” statement does just what it sounds like: orders the list. In this case, the goal is to arrange the list according to the number of letters in the name of the item.

Those statements I just mentioned are provided courtesy of the Complex Listing extension. However, you’ll notice here that the work was only being done because it was placed within a rule about the writing a paragraph about activity.

Room Description Control

Make sure your source text is back to the Reference Configuration 2. At the top of the source text, add the following statement:

SOURCE TEXT

```
Include Room Description Control by Emily Short.
```

Before I start getting into what this extension is doing, it's really important to see what it's basic inclusion does.

Try It!
Execute the story. You'll notice that beyond the author-provided Living Room description, no locale information is displayed.

With your story running, let's consider some commands:

STORY TEXT

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

> PARAGRAPHS
Paragraph debugging is now on.

> LOOK
This is the Living Room description.

table: rank 0
wooden chair: rank 0
Jones: rank 0
alarm clock: rank 0
standing lamp: rank 0
plastic box: rank 0
rubber chicken: rank 0
light bulb: rank 0
high shelf: rank 0
jug of mayonnaise: rank 0
service alcove: rank 0
waiter: rank 0
food tray: rank 0
bottle of fresh water: rank 0
jacket: rank 0
```

In a nutshell, what this extension provides is a "table of seen things" and each of those "seen things" is given a ranking number. So why don't you get any output? The answer is contained in the fact that in the list above every item in the Living Room has a rank of 0. An item with such a rank means that item is effectively excluded from the locale.

What happens is that the Room Description Control goes through the 'Table of Seen Things' and executes the a writing a paragraph about activity for each item it finds in that table that is not flagged as "mentioned". Any items whose names have been printed earlier during the room description are given the property "mentioned" and will not have any specific writing a paragraph about activity called for them. Any item is considered to be "mentionable" when that item is currently not "mentioned" and when it is "marked for listing".

If this all sounds strangely familiar, it's probably because you remember reading about this exact same behavior earlier in this document. What I've just described is the default behavior of Inform and that default behavior is what Room Description Control emulates. That being said, the Room Description Control extension replaces the room description paragraphs about objects rule from the carry out looking rulebook. It's that rule that gets the locale display going. Specifically, the Room Description Control extension replaces that rule with one of its own: the new object description rule.

As to why you're getting nothing for your locale display, you have to understand the operating principle of the extension. The Room Description Control is intended to completely replace Inform's management of room description output (for locale display) with some other mechanism that has built-in management of priority (meaning, the order in which paragraphs are written) and omission (meaning, skipping some objects that would ordinarily have their own paragraphs).

What's the "other mechanism"? That's what you, as the author, have to provide. You have to provide a set of writing a paragraph about activities that will produce the kind of output you want. As you might guess, this is not necessarily something you'll want to do on a whim.

The previous sections of this document were, in fact, showing you how to work with Inform's default system and modify it to suit your needs. What I'm describing here is a way to entirely replace that default system.

If there's one operating assumption behind the Room Description Control extension, it's that the author desires to produce their own description output framework, using the writing a paragraph about activity pretty much exclusively.

So let's talk about how to do this. There's an extension out there called 'Ordinary Room Description' and the purpose of this extension to use the Room Description Control framework (which totally overrides the default Inform functionality) and then to replicate the default behavior. What's nice is that this extension logic serves as a great example as to how to use the Room Description Control framework in the first place.

Getting an Ordinary Room Description

In order to get back to the "ordinary" (default) room description, you need to understand a little about how Room Description Control is working. As mentioned before, the main thing the extension does is replace the room description paragraphs about objects rule (in the carry out looking rulebook) with its own rule, called the new object description rule.

What that new rule does is fairly simple: it references (follows) another rulebook called the description-priority rulebook. Essentially what happens is that every item is set as "not marked for listing." That's essentially why, with the exception of the author-provided room description, you see nothing at all displayed when the Room Description Control framework is included as part of your source.

Those description-priority rules are important for a couple of reasons. One of those reasons is that they deal with the ordering (ranking) of items and that ordering determines the sequence in which any writing a paragraph about activities occur. This is partly handled by the use of a ranking rulebook. The general idea is that the higher an item's rank, the higher it will be sorted in the Table of Seen Things and thus the earlier it will appear in the room description. Another reason the description-priority rules are important is important is that they reference the description-concealing rulebook. These rules allow you to determine when (or even if) certain items are listed at all as part of the room description.

For now, let's at least get our non-descript items listed. Add the following source text:

SOURCE TEXT

```
A last description-priority rule:
  repeat through the Table of Seen things:
    if the output entry is unmentioned, change the output entry to marked for listing;
  if a marked for listing thing is in the location, list the contents of the location, as a sentence, listing marked items only.
```

What I'm doing here is inserting a rule at the end of the description-priority rulebook and saying that any unmentioned items in the Table of Seen Things (a table which, remember, the Room Description Control builds) should be set to "marked for listing." Then for each such item that's in the location of the protagonist, I'm making sure those items are listed.

While this will give you some output, it certainly doesn't look like how Inform normally presents this list. Here's how you can modify the above rule to get a little closer:

SOURCE TEXT

```
A last description-priority rule:
  repeat through the Table of Seen things:
    if the output entry is unmentioned, change the output entry to marked for listing;
  if a marked for listing thing is in the location:
    say "You can see ";
    list the contents of the location, as a sentence, listing marked items only;
    say " here."
```

Try It!
Make sure you try both of the variations above and execute the story with each in place. Notice how the output changes.

Now let's see how items with an initial appearance are handled. Keep the above source text you just added in place and then uncomment the statements that set the initial appearance for the chair and the table. What happens when you execute the story?

Not a whole lot, at least in terms of the initial appearance. You'll notice that the table and the chair are listed in the general list of items. In normal Inform operation this list would be called the "nondescript" list but remember that the inclusion of Room Description Control has totally overridden the 'normal Inform operation' and the above source text (for the description-priority rule) did not re-instate that. The point is that all you have is a list of items. Further, any initial appearances are not considered.

First keep in mind what Inform means by "initial appearance": it means the item in question has a property by that name and that the item in question has not been handled (i.e., picked up and dropped) by the protagonist. So those concepts have to be defined again. Add the following source text:

SOURCE TEXT

```
Definition: a thing is initially-described if it provides the property initial appearance.
Definition: a thing is initially-describable if it is initially-described and it is not handled..
```

Having the definitions is great and all, but you'll need to add a rule that actually does something with them in order to recreate Inform's default functionality for handling the initial appearance of items. Add the following to your source text:

SOURCE TEXT

```
Rule for writing a paragraph about an initially-describable thing (called this_item):
  now this_item is mentioned;
  say "[initial appearance of this_item][paragraph break]".
```

Here's your output:

STORY TEXT

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

This is the initial appearance of the table.

This is the initial appearance of the chair.

You can see a table, a wooden chair, the alarm clock, the standing lamp, a plastic
box, a high shelf and a service alcove here.
```

You certainly have your initial appearances back for the table and chair. Although you'll note that the table and chair are still listed in the second list as well. A bit more subtle is the fact that the "You can see..." start to the secondary list of items has not changed to "You can also see ...", which is a standard Inform practice when other items have been mentioned. Both situations can be handled by changes to the description-priority rule. Change your source text for that rule as such:

SOURCE TEXT

```
A last description-priority rule:
  now every thing is not marked for listing;
  repeat through the Table of Seen things:
    if the output entry is unmentioned, change the output entry to marked for listing;
  if a marked for listing thing is in the location:
    say "You can [if something is mentioned]also [end if]see ";
    list the contents of the location, as a sentence, listing marked items only;
    say " here."
```

What the first line does is make sure that not all items are marked for listing, which they otherwise are. The text substitution is presumably obvious in terms of its effect.

Now let's say you wanted the plastic box to be unlisted, even after everything you just did. You could insert your own rules into the description-concealing rulebook, which is provided by the Room Description Control extension. Here's an example:

SOURCE TEXT

```
A description-concealing rule when the player is in the Living Room:
  now the plastic box is not marked for listing.
```

With this in place, you'll notice that the plastic box no longer appears in your list of items.

You can also change the 'ranking rule' for items, indicating where they do and do not appear in the list. As a simple example of this, add the following to your source text:

SOURCE TEXT

```
A ranking rule for the chair:
  increase the description-rank of the chair by 10.
```

Read It!
Make sure to look at the differences in output of the list.

Without Ranking Rule	With Ranking Rule
Living Room This is the Living Room description. This is the initial appearance of the table. This is the initial appearance of the chair. You can also see the alarm clock, the standing lamp, a high shelf and a service alcove here.	**Living Room** This is the Living Room description. This is the initial appearance of the chair. This is the initial appearance of the table. You can also see the alarm clock, the standing lamp, a high shelf and a service alcove here.