IFM

Version 5.2

Interactive → **Fiction** → **Mapper**

by Glenn Hutchings

Contents

List of Figures

List of Tables

1 Introduction

1.1 About IFM

IFM is a language for keeping track of your progress through an Interactive Fiction game, and a program for producing various different sorts of output using it. You can record each room you visit and its relation to other rooms, the initial locations of useful items you find, and the tasks you need to perform in order to solve the game.

The IFM mapping commands are designed so that you can easily add to the map as you explore the game. You type in the rooms you visit and the directions you move in to reach other rooms, and IFM calculates the position of each room in relation to the others. A map can consist of several independent sections, allowing you to divide up the map however you like. See Section 2 for an example of how to make a map.

The IFM task commands, if used, allow you to specify the order in which game-solving tasks must be done. The IFM program can then calculate and print a high-level "walkthrough" of the game. See Section 3 for examples of how to do this.

Several output formats are available, including PostScript, Fig and ASCII text—these are described in Section4. Some of the output formats have a set of variables which control the appearance of the output— see Section 4.4.

1.2 History of IFM

When I started playing IF games, years ago, I did what many other people did—made a rough map on paper of where I'd explored, so I could see which directions I hadn't tried yet. Inevitably, the maps always fell off the page and turned into a complete mess.

My first attempt at solving this problem was to make use of the tools I had available to draw the map for me. So I started typing the map network into a file, and wrote a small Perl script to convert the data into something that groff could read and draw a map from. I also added a way of recording which items I'd found, and whether I'd found a use for them yet.

As it stood, this worked quite well—it produced nice maps and lists of items so I could see where I'd got to. The only problem was that it was just as tedious making maps this way as on paper, since I had to work out the offset of each room relative to a fixed origin.

Eventually I decided that enough was enough. I wanted to be able to add stuff to the map as I played the game, without having to worry about offsets or anything irrelevant like that. Hence the main design criteria for IFM. The other criteria were easy—an input language, for flexibility and extendability, and a program for converting files in this language into different types of output.

Partway through writing the initial version of IFM, I had the idea that it would be nice to be able to somehow record not only the map and items found so far, but what's been done to solve the game. The concept of tasks appeared, which immediately tripled the size of the program and caused all sorts of headaches.

But eventually it got to the stage where it might be worth releasing, so in June 1998 the first version appeared in the IF archive. Very soon after that, I got a lot of feedback from one Dave Chapeskie (dchapes@ddm.wox.org), who made a lot of great suggestions for improvements and new features. He also very kindly started up a Web page for IFM maps; you can find it at http://www.sentex.net/~dchapes/ifm.

And now, finally, it's finished and I can go back to that IF game I was playing.[1]

---

[1] Oh all right. Maybe *one* more feature. . .

2   Making Maps

This section gives you a tour of the main commands for making maps. It's not complete; see Section 5 for a full list of commands.

2.1   Introduction to Maps

The traditional Infocom-style way of drawing Interactive Fiction maps is the "boxes-and-lines" method; an example is shown in Figure 1. This is the style of map that IFM produces. Rooms are represented as boxes on a square grid, and links between the rooms are drawn as lines connecting them. Links emanate from rooms in any of the eight standard compass directions, and also follow the grid. In the following sections, we'll introduce the IFM commands that can be used to draw this example map.
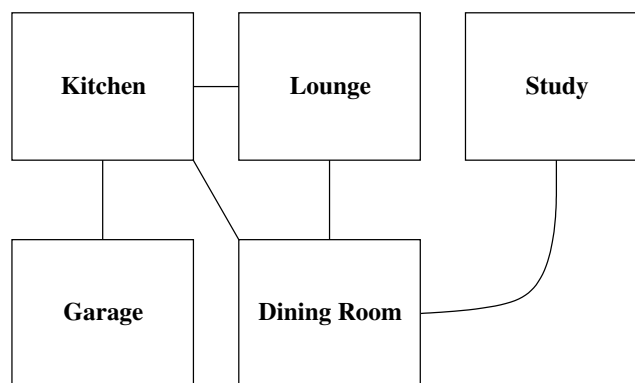
Figure 1: Infocom-style map

2.2   Adding Rooms

To draw the example map from the previous section, you first choose an arbitrary start location: the kitchen (when mapping a real game, you'd choose your actual start location). To add the kitchen, just type this:

```
room "Kitchen";
```

Now you're in the kitchen. Suppose, if this were a real game, that you first went south to explore the garage. That can be added to the map like this:

```
room "Garage" dir south;
```

Now you've said which way you went to get to the garage, and since you were in the kitchen, IFM knows that the garage is south of the kitchen. By the way, `south` can be abbreviated `s` (and similarly for all other directions), just like in the games.

Ok, you're in the garage. Unfortunately, that's a dead end and you have to retrace your steps to the kitchen. You've already mapped that, so there's no need to do anything. Now you head off east to the lounge. Now, you're moving from the kitchen again but IFM thinks you're in the garage (IFM's idea of "where you are" is always the last room mentioned). You need a way of referring to the kitchen again—to do that, you add a "tag" to it by changing the "kitchen" line like this:

```
room "Kitchen" tag Kitchen;
```

The tag name can be any name you like. You might think that you could refer to the kitchen by using the name in quotes, but that would mean you could never have two distinct rooms with the same name. Another advantage of tags is that they can be much shorter than the real room names. The tag `K` would be just as valid in the example above (though not as readable).

Now you can refer to the kitchen by its tag, so you can move east from it into the lounge like this:

```
room "Lounge" dir e from Kitchen;
```

The `from` clause tells IFM where you're moving from. If it's omitted, it assumes you're moving from the last room mentioned.

Continuing your exploration, you move south into the dining room:

```
room "Dining Room" dir s;
```

You exit the dining room to the east, and turn a corner north before entering the study. How can you represent the corner faithfully on the map? Like this:

```
room "Study" dir e n;
```

This says that you move east, then north, to get to the study. Now, what if someone locked the study door behind you and the only way out was through the window? That's a one-way trip into the study, which you can indicate using the `oneway` attribute like this:

```
room "Study" dir e n oneway;
```

This is indicated on the map by an arrow.

2.3   Adding Links

The map as it stands is not complete—the diagonal link between the kitchen and the dining room is missing (because you didn't go that way in visiting the kitchen or the dining room). To add it, you need to modify the dining room command like this:

```
room "Dining Room" dir s link Kitchen;
```

The `link` clause creates a straight-line link between the current room and the room with the specified tag name (in this case, the kitchen).

Note that if this link needed to turn corners, as in the study example above, then that method of linking the rooms wouldn't have worked. In that case, you'd have to use the stand-alone `link` command. For example:

```
link Diner to Kitchen dir n nw;
```

This assumes you've given the dining room the tag name `Diner`. The link starts off going north, then turns northwest, and finally goes toward the kitchen. Note that in a `link` command, if you omit the final direction which leads to the linked room, it is added automatically.

2.4   Other Directions

Suppose that there were steps down from the kitchen into the garage, and that you wanted to indicate that you could up or down as well. You could do that using the `go` clause, like this:

```
room "Garage" dir s go down;
```

This indicates that the actual direction travelled is downwards, but it is still represented as south on the map. The `go` clause accepts `up`, `down`, `in` and `out`. As with compass directions, `up` and `down` may be abbreviated as `u` and `d`.

2.5   Room Exits

At various points in a game, you arrive in a room with many directions to explore. It is useful to be able to mark some of these directions as unexplored, so that you can come back and explore them later. You could mark these by creating dummy rooms in those directions, but this is tedious. Alternatively, you can use the `exit` clause, like this:

```
room "Dining Room" dir s exit nw e;
```

This says that there are two unexplored exits from this room, in the northwest and east directions. When a map is drawn, this fact will be displayed by a small line poking out of the room in those directions.

When you come to actually explore those directions, and add links to new rooms, the corresponding room exit markers will no longer be drawn. So you can leave the exit clauses in if you want.

2.6   Map Sections

In IFM, rooms are divided into groups called "map sections". Each room in a map section has an explicit spatial relationship to all the other rooms in that section. A room which is obtained by moving via a `dir` clause from a previous room is on the same map section as the previous room, since its co-ordinates can be calculated relative to it.

There are several reasons why it might be a good idea to split a game map into different sections:

- Some maps can be very large, and may not look good on a single piece of paper.

- It might be awkward to put rooms in relation to each other because of, say, a lot of up/down connections which have to be "flattened out".

- The game might naturally divide into sections—a prologue, middle-game and end-game, for example.

IFM manages independent map sections automatically, by deciding which rooms are on which section. No special command is needed to start a new map section—simply define a room which has no connection to any previous room, by leaving out the `dir` clause (note that that's how the kitchen starts out, in the example).

Rooms on different map sections are completely separate, and you may not link them via the `link` command. However, you can indicate where a room on one section is connected to a room on another, using the `join` command:

```
join Room1 to Room2;
```

As usual, `Room1` and `Room2` are tag names. You can also use `join` as a clause in a room command (usually done with the room starting in a new section):

```
room "Basement" join Ground_Floor;
```

The "joined" status of the two rooms is indicated after their description text; the default is to use an increasing number.

Each map section can be given a title using the `map` command, like this:

```
map "Kensington Gardens";
```

This names the next map section that hasn't been named. Note that you should have as many `map` commands as you have map sections, although this isn't enforced—any names that are missing will be assigned default names, and extra names will be ignored. It's conventional to give a `map` command just before starting a new map section.

In rare circumstances (e.g., a three-dimensional maze) you may need to have rooms on the same map section which are not connected to each other. The room `dir` clause creates an implicit link from the previous room by default, but you can stop this from happening by using the `nolink` attribute. As a trivial example:

```
room "One Side of Wall" tag this_side;
room "Other Side of Wall" dir e nolink tag other_side;
room "Underground Passage" tag passage_1;
room "Underground Passage" tag passage_2 dir e;
join this_side to passage_1 go down;
join passage_2 to other_side go up;
```

In this example, there are two map sections: above ground, and below ground. But the two above-ground rooms are not connected directly.

2.7 Adding Items

As well as rooms, IFM can indicate the initial rooms of various items found in the game. To add an item, use the `item` command like this:

```
item "Spoon" in Kitchen;
```

The `in` clause can be omitted, and then the room defaults to the last room mentioned. You can also add an arbitrary note to each item (e.g., to remind you what it's for) using the `note` attribute:

```
item "Spoon" in Kitchen note "Stirs tea";
```

Here's the completed map description for the above example, with a few other items thrown in:

```
title "Example Map";

room "Kitchen" tag Kitchen;
  item "spoon" note "Stirs tea";
  item "sink";
  item "monkey";

room "Garage" dir s go down;
  item "monkey wrench" note "For wrenching monkey out of sink";

room "Lounge" dir e from Kitchen;
  item "TV set";

room "Dining Room" dir s link Kitchen;
  item "table";
  item "chair";

room "Study" dir e n oneway;
```

See Section 4 for how to display it.

2.8 Conflicts

After creating a map from a real game and sending the results through IFM, you may get warnings which talk about things overlapping. This is due to two rooms, or a room and a link, wanting to occupy the same space on the map. There are several ways that this could occur:

- The game designer made some room links longer than others, and you haven't taken that into account. To extend the length of a link, just add a length indicator after the direction in the `dir` clause (e.g., `dir e 2` instead of `dir e`).

- One of the links turned a corner, so that the direction you use to go back isn't the opposite of the one you used to get here. In that case, you need to add the corner-turn in the link (e.g., `dir e s` instead of `dir e`).

- The map is multi-level, in which case it's probably best to split it into different map sections.

- The map is just weird. Colossal Cave is a good example, particularly the above-ground section and the mazes. There seems to be no logic tying the rooms together. You're on your own.

3 Solving the Game

As well as making a map of your game, you can record the steps you took to solve it. IFM can then calculate a (fairly) optimal solution. This section is a guide to how to do it. Again, it's not a complete specification—see Section 5 for that.

3.1 Introduction to Tasks

The basic game-solving action is called a "task" in IFM-speak. To introduce a task, you use the `task` command, like this:

```
task "Put the fluffy bunny in the incinerator";
```

Most tasks need to be done in a certain room. The default is that a task must be done in the last room mentioned. You can change that by using the `in` clause, just as for items. Some tasks can be done anywhere—you can say `in any` to indicate that. As usual, you can add tags to tasks in order to refer to them later.

The game solver in IFM divides tasks into two fundamental types: "safe" and "unsafe". An unsafe task is one that, if done, might get you into a position where solving the game is impossible. The game solver avoids doing unsafe tasks until it really has to.

3.2 Task Dependencies

3.2.1 Requiring Tasks

A lot of tasks require a previous task to be done first. To say this, you use the `after` clause:

```
task "Press incinerator start switch" tag press_switch;
...
task "Put the fluffy bunny in the incinerator" after press_switch;
```

As a shortcut, to avoid having to tag many tasks, you can say `after last` to indicate the last task mentioned. For example, if the two tasks above could be done in the same room, you could say

```
task "Press incinerator start switch" tag press_switch;
task "Put the fluffy bunny in the incinerator" after last;
```

Alternatively, you could merge the two into a single task—the simplicity or complexity of tasks is up to you.

The `after` clause only says that a task will come after another—it doesn't specify how soon after. But in some situations it is essential that a task immediately follows a specific previous task, without deviation. You can use the task `follow` clause to specify this. For example:

```
room "Mission Control" ...;
task "Activate launch sequence" tag activate;
...
room "Rocket Cabin" ...;
task "Fasten seat belt" follow activate;
```

The `follow` clause creates a chain of tasks that must be done one after the other. The game solver will not attempt the first task in the chain until it knows that all the tasks are possible (i.e., all the prerequisites for each task in the chain are satisfied). Also, if one of the tasks in the chain is unsafe, then all previous tasks in the chain are also marked unsafe.

Of course, you can only have a single task in a `follow` clause—the immediately preceding task. It is an error for two or more tasks to try to immediately follow the same task.

3.2.2   Requiring Items

For a lot of tasks, you need to have one or more items in your possession. You can indicate this by using the `need` clause, like this:

```
task "Put the fluffy bunny in the incinerator" need bunny;
```

Here, `bunny` is the tag name of the corresponding item. You can list more than one item tag—e.g., `need bunny asbestos_gloves`.

   Note that you don't need to add tasks to get required items yourself—the game solver does that automatically. It knows it has to get all the items which appear in `need` clauses.

3.2.3   Obtaining Items

Sometimes a task need to be done before you can get an item. One way to indicate this is with the task `get` clause:

```
task "Put money in drinks machine" need coin get lemonade;
```

This naturally implies that all tasks which supply an item (via the `get` clause) must be done before any task which needs that item.

   An alternate way to phrase this is with the item `after` clause, which says that the item can't be picked up until a specified task is done. This is a common combination in IFM:

```
task "Put money in drinks machine" need coin;
item "lemonade" hidden after last;
```

Some items are only available *before* doing a certain task. You can use the `before` clause to say that:

```
item "precious diamond" before trigger_alarm;
```

Some items can only be picked up if you're already carrying another—use the `need` clause for that:

```
item "hot coal" need tongs;
```

Sometimes doing a task not only allows you to get an item, but also puts it in your inventory. You can say that using the `give` clause:

```
task "Buy beer" need money give beer_mug;
```

The `give` clause overrides all other restrictions placed on getting items; the item is just teleported into your possession.

3.2.4   Ignoring Tasks

In some circumstances, all the effects of doing a task occur before the task is done. If this happens, the task will be ignored. For example, if a task A gives an item, but that item is first given by task B, then task A will be ignored (provided it doesn't do anything else of importance).

   If a task has no effect, it is *not* ignored—it's assumed that you've recorded that you did something but don't know why yet. Also, tasks which finish the game or score points are never ignored.

   You can explicitly ignore a task using the `ignore` attribute. This is useful while you're solving the game (see Section 3.7) and when the task can be done by other tasks (see the next section).

3.2.5   Doing Other Tasks

You can arrange for a task to automatically do other tasks after it is done, using the `do` clause. For example:

```
room "Control Room";
task "Press airlock button" do open_airlock;
...
room "Outer Airlock";
task "Open airlock" tag open_airlock;
room "Inner Airlock" dir s after last;
```

In this example, the airlock can be opened in one of two ways: manually, when in the Outer Airlock, or via the button in the Control Room. Pressing the button will cause the "open airlock" task to be done immediately afterwards. Note that if the manual method is used first, the press-button task will be ignored.

The `do` clause causes *any* task to be done—even tasks that have prerequisites, and explicitly ignored ones. This is useful in that you can create special ignored tasks that can be done by any of a set of other tasks, whichever gets there first. The `do` clause is also recursive: a task can do another task which in turn does another, and so on.[2]

Note that any task which does an unsafe task in this way is itself marked unsafe.

3.3   Handling Items

3.3.1   Inventory Items

Items can be split into two types: useful and useless. A useful item one that is needed by at least one task, or is required in order to be able to move somewhere; all other items are useless. The game solver will try to go and get all useful items, and will ignore the useless ones. It keeps track of the items it's carrying, and knows when a useful item is no longer needed.[3] At that point, it will be dropped.

If the solver obtains a useless item (via a task `get` or `give` clause, or an item `need` clause) it will never drop it. This is just a default; you can change it by setting the variable `keep_unused_items` to zero. In that case, useless items will be dropped as soon as possible.[4]

If you want to make sure that an item is never dropped in any circumstance, you can mark it with the `keep` attribute. This is handy for items that act as general containers for other items.

Sometimes a useful item needs to be kept for longer than usual. In the hot coal example above, the tongs would be dropped as soon as the coal was picked up, leaving you with a burnt hand. What's needed here is to carry the tongs for as long as you have the coal. You can use the `keep with` clause to say that:

```
item "hot coal" tag coal need tongs;
item "tongs" tag tongs keep with coal;
```

Now the tongs won't be dropped until after the coal is, even if they have no other use. Similarly, there's also a `keep until` clause, which keeps an item until a specific task is done.

Finally, if a room has the attribute `nodrop` set, no items will be voluntarily dropped in that room. Any items which need to be dropped will then be dropped after the next task that happens in a room where dropping is allowed.

3.3.2   Losing Items

Sometimes, doing a task causes items to be destroyed. You can say that with the `lose` clause:

```
task "Light bonfire" need match lose match;
```

---

[2]However, you can't create an infinite loop since each task can only be done once.

[3]It has a magic crystal ball that can see into the future.

[4]The reason for the default is that useless items obtained in this way probably *do* have a purpose—you just don't know what it is yet. This is relevant when you're creating a recording from a partially-played game; see Section 4.2.5 for details.

This naturally implies that all other tasks which need the item must be done before the task that destroys it. A "drop" task is never generated for items that are lost in this way.

Incidentally, you can use the special tag `it` to refer to the last room, item or task tag name within a command. So the previous example could also have been written

```
task "Light bonfire" need match lose it;
```

3.3.3 Dropping Items

As mentioned in Section 3.3.1, IFM knows when a useful item is no longer needed, and drops it automatically. But sometimes items need to be dropped temporarily, even though they're needed later. You can do that using the `drop` clause:

```
task "Throw spear at tree stump" need spear drop it;
```

In this example, the spear is dropped in the same room that the task was done in. If you ever need the spear for anything else, it will be picked up again by the game solver. Note that an item will only be dropped if it is being carried—mentioning an item in a `drop` clause does not imply that it's needed to do the task.

Sometimes items must be dropped in a different room to the one you're in. You can use the `in` clause to modify things:

```
room "Top of Chute";
task "Put laundry in chute" need laundry drop it in Bottom_of_Chute;
```

In other cases, you need to drop all the items you're carrying, or all except certain items. You can use `drop all` and `drop all except` to say that.

Normally, if an item is dropped but is needed again for some other task, there is nothing to stop the game solver picking it up again (provided there's a path to the room the item was dropped in). But sometimes you need to drop an item and not pick it up again until you've done something else. You can use the `until` clause to say that:

```
task "Put coin in slot" give chocolate drop coin until open_machine;
```

A task which drops items will be marked unsafe if there is no path back to the dropped items.

3.3.4 Leaving Items

There are some situations where your movement is blocked if you are carrying particular items. You can use the `leave` attribute of rooms, links and joins to specify a list of items that must be left behind before using them. For example:

```
room "Bottom of Canyon";
item "heavy boulder" tag boulder;
room "Top of Canyon" dir n go up leave boulder;
```

If the `leave` clause appears before the `dir` clause, that means the items must be dropped before entering the room (from any direction). It is generally the case that, if an attribute could apply to a room or its implicit link with the previous one, its position relative to the `dir` clause is what decides it.

You can also say `leave all`, which means that you must leave all the items you're currently carrying, and `leave all except`, which omits certain items from being left behind.

When finding a solution, the game solver will carry items until it is forced to drop them. If the dropped items are needed later, the game solver will try to come back and get them. If it is trying to do a task which requires items, it will choose a route to get to the task room which doesn't involve dropping any of the needed items.

Note that the `leave` clause overrides the room `nodrop` attribute; items will be dropped even in those rooms.

3.4   Moving Around

3.4.1   Limiting Movement

Sometimes an item is required, or a task needs to be done, before movement in a certain direction is possible. To represent this, you can give `need` and `after` clauses to rooms, links and joins. For example:

```
room "Cemetery" dir s from winding_path;
task "Unlock the iron door" need rusty_key;
room "Crypt" dir s go down after last;
```

Here's another example:

```
room "Dimly-lit Passage" dir e;
room "Dark Passage" dir e need candle;
```

In this case it is the link between the two rooms that is blocked off until the candle is obtained. If the `need` clause had appeared before the `dir` clause, the restriction would apply to the room itself (i.e. no entering the room from *any* direction without the candle).

In some cases, doing a task closes off a room, link or join so that it can't be used any more. You can use the `before` clause to indicate this:

```
room "Bank Vault" tag Vault;
room "Bank Entrance" tag Entrance dir e before trigger_alarm;
```

All tasks which close things off like this are marked unsafe, since they could block off a crucial path through the game.

Sometimes in a game there is the situation where a path is closed off and, later on in the game, reopened again. A single link or join can't represent this. However, there's nothing to stop you from using two or more joins between the same rooms. If you mark them with the `hidden` attribute, they won't appear on the map either. For example, this line could be added to the previous example to provide an escape route:

```
join Vault to Entrance go e after disable_alarm hidden;
```

3.4.2   Movement Tasks

There are several different ways of moving around in a game. The usual way is to say the direction you want to go in. Another way is to do something else which results in movement. A good example is the magic word XYZZY from Colossal Cave. It acts exactly like a movement command, in that you can use it again and again and it moves you somewhere predictable. The best way to represent this in IFM is to use a join to connect the two rooms, and specify the command used to do the movement via the `cmd` clause, like this:

```
join Debris_Room to Building after examine_wall cmd "XYZZY";
```

Yet another way of moving around is a one-off event that "teleports" you to a different room. You can indicate that this happens using the task `goto` clause,[5] and supplying the tag name of the destination room. For example:

```
task "Get captured by goblins" goto Dungeon;
```

As soon as the task is done, you teleport to the new location—no intervening rooms are visited. Note that because each task is only done once, this method of travel can only be used once. Note also that the `drop` and `leave` actions are done *before* teleporting you to the new location (so if you drop items in the "current room", you will be teleported away from the dropped items).

---

[5]All the best languages have a `goto` statement, you know.

3.5 Other Game Features

3.5.1 Scoring Points

Many games have some sort of scoring system, whereby you get points for doing various things. In IFM you can record this using the `score` clause, which can apply to rooms, items or tasks. It takes one integer argument—a score value. For rooms, it's the score you get when visiting it for the first time. For items, it's the score for first picking it up. For tasks, it's the score for doing that task. If an item has a score, but is given to the player via a task `give` clause, then its score is added to the score for that task instead.

3.5.2 Finishing the Game

Usually a game finishes when you complete some final task. You can indicate which task this is using the `finish` attribute. This attribute can attach to rooms, items or tasks, giving three different types of finish condition: entering a room, picking up an object or doing a task. If the game solver ever manages to do something which is flagged with the `finish` attribute, it considers the game solved and stops. Any extra things left to do will not be done, even if they score points.

3.6 Finding a Solution

Here's what the game solver does in order to come up with a solution to the game. First, extra internal tasks are generated. These are tasks to:

- get items which are required for explicitly-mentioned tasks to be done,

- get items which are required to get other items,

- get items which are needed to go in certain directions,

- get items which are scored,

- go to rooms which are scored.

Next, all the rooms are connected using their links and joins. This means that for each room, a list is made of all other rooms reachable in one move. Note that it is possible for some rooms to be unreachable—for example, all rooms in a section where there is no "join" to rooms on other sections.

Then the task dependencies are calculated. A dependency is where one task must be done before another. The task dependencies are examined to see if there are any cycles—that is, chains of tasks where each one must be done before the next, and the last must be done before the first. If there are any, then the game is unsolvable, since none of the tasks in a cycle can be done.

If there are no cyclic dependencies, the task list is "traversed" to find a sequence which solves the game while satisfying the task dependencies. The start room is the room which was first mentioned in the input (but this can be changed—see Section 5). While there are tasks left in the task list, the following steps are performed:

1. The inventory is examined to see if there are any unwanted items; if so, and dropping items in the current room is allowed, they are dropped. An item is wanted if at least one of the following is true:

   (a) it's needed for movement,

   (b) it's needed for a task that hasn't been done yet,

   (c) it's being kept unconditionally,

   (d) it's being kept with another item that's carried,

   (e) it's being kept until a certain task is done.

2. The map is traversed to find the distances of all rooms from the current room. Then the task list is sorted in order of ascending distance of the rooms they must be done in. Tasks which can be done in any room count as having distance zero.

3. The sorted task list is scanned to find the nearest possible task. For a task to be possible, the player must:

   (a) have all required items,

   (b) have done all required previous tasks,

   (c) be able to get from the current room to the task room via a path which doesn't require items not yet collected, or tasks not yet done, or which involves dropping needed items on the way.

   Priority is given to safe tasks. For a task to be safe,

   (a) it must not have previously been marked unsafe (e.g., because it closes off map connections),

   (b) there must be a return path from the task room back to the current one. This is to avoid taking a one-way trip before preparing properly.

   If there are any safe tasks, the nearest one will be done next regardless of how close an unsafe task is. If there are no safe task, the nearest unsafe task will be chosen.

4. If there was a possible task, do it and remove it from the list. Move to the room the task was done in (if any). If not, then the game is unsolvable. Give up.

5. Finally, examine the list of remaining tasks to see if any are now redundant and can be removed. A redundant task is one that only does something that's already been done (e.g. go and get an item that you've already been given).

3.7 Tweaking the Solution

There will be some situations (quite a few, actually) where the game solver doesn't do things the way you want it to. This section gives a few tips, and some new keywords, for modifying things.

3.7.1 Making things safe

Firstly, the game solver is completely paranoid. It has to be, because it doesn't do any lookahead past the current task. It won't do anything unsafe (e.g., go to a room to do a task when there's no immediate return journey) unless there's nothing safe left to do. It will quite happily plod halfway across the map to pick something up rather than do something a bit scary in the next room.

However, you can reassure it with the task `safe` attribute. Adding this to a task tells the solver that this task is safe, regardless of what it thinks. So if you *know* that a one-way trip can eventually be returned from, by doing other tasks, you can stop the solver from avoiding it. But bear in mind that by doing this you are taking full responsibility if the solver gets stuck.

If you want to be seriously reckless, you can set the variable `all_tasks_safe` to a nonzero value. Then, *all* tasks will be considered safe.

3.7.2 Changing path lengths

Another thing the solver doesn't know about is how easy or difficult it is to get from place to place on the map. Suppose you're in a game which is on two levels separated by a tiresome set of access doors with ID cards. The connection between the levels may only be two rooms on the map, but it's a lot more in terms of typing. You can avoid unnecessary trips through these doors by artificially changing the "length" of the connection between levels, by using the `length` attribute of links and joins:

```
room "Level A" tag LA ...;
room "Level B" tag LB dir e length 50;
```

In this way, by choosing an appropriate number for the length, you make it appear to the solver that all the rooms in level A are closer to each other than any of the rooms in level B. This means that priority will be given to tasks in rooms in the same level as you are now, (hopefully) minimizing the number of level changes. Note that the `length` attribute doesn't affect map drawing at all.

3.7.3 Closing off paths

There may be times when you want a map connection to appear on the map, but not be used in solving the game—for example, it may be certain death to go that way. You can use the `nopath` attribute of rooms, links and joins to indicate this. It doesn't affect map output in any way.

    Another use for this attribute is to force the game solver to do things in a different order. This might be preferable to adding fake task dependencies.

3.7.4 Ignoring parts of the solution

Sometimes it's useful to be able to ignore certain parts of the solution—for example, if you want to generate a sequence of game commands that get you to a particular position as quickly as possible. To that end, you can mark tasks and items with the `ignore` attribute. An ignored task will never be attempted, and an ignored item will never be picked up. This means that anything dependent on those tasks or items will not be done either. The game will very probably be unsolvable as a result, unless you've ignored an unused item, or ignored a task that's done elsewhere via a `do` clause.

3.7.5 Keeping fixes together

It's probably best to keep all your "game tweaks" together, separate from the "pure" game, and commented appropriately. You can do this by using commands which just modify existing objects, instead of creating new ones, by referring to their tags. As an example, suppose you have the following situation:

```
room "Top of Chute" ...;
room "Bottom of Chute" dir s go down oneway;
task "Do something weird" tag weird_task;
...
```

Suppose you're at the top of the chute, and that there's some stuff to be done at the bottom, but no immediate way back up. As usual, the game solver balks at taking a one-way trip and will do anything to avoid it. But suppose you know that, as long as you have your giant inflatable cheeseburger, you can get back out again. You can say:

```
# Bottom of chute isn't that scary.
task weird_task need burger safe;
```

which modifies the task at the bottom of the chute to (a) require the burger (so that you won't go down there without it), and (b) be considered safe by the game solver. So it will happily slide down the chute without getting stuck at the bottom.

    This way of modifying previous objects applies all types of object, even links and joins—these can be tagged too, in the normal way. The single exception is the implicit link created by the room `dir` clause. These links automatically get tagged when the room does, and with the same name. So the two-level example above could be split into:

```
room "Level A" tag LA ...;
room "Level B" tag LB dir e;
...
# Stop gratuitous travel between levels.
link LB length 50;
```

3.7.6 Displaying solver messages

Finally, you can gain an insight into what the game solver's up to by setting the `solver_messages` variable (either in one of the input files, or via the `-set` command-line option). This produces reams of output giving details of the game solver's thoughts before it does anything. It's supposed to be self-explanatory, but my view is slightly biased. Detailed documentation may follow (a) if enough people ask for it, and (b) if I ever get around to it.

3.8   Limitations

Given the wild imaginations of today's IF authors, there are bound to be some game solving situations that can't easily be dealt with using IFM. Some of the things that IFM ignores are:

- Random events. For example, the Carousel room in Zork, and all the NPCs in Colossal Cave. There's no way to address this problem, but then again, hand-written walkthroughs have the same difficulty. However, if you're trying to tailor recording output so that it will play back properly in an interpreter, there is a workaround—see Section 4.2.5.

- Carrying capacity. A solution may require you to carry more than you're allowed. This might be addressed in a future version, but inventory-juggling puzzles are out of fashion these days (if they were ever in) so this is not much of a problem. Some games provide you with an item that can carry stuff for you—if so, a workaround is to add some special tasks that periodically put everything you're carrying into it.

There are some other limitations that are the result of certain keyword combinations in the current implementation of IFM. These are fixable, and hopefully will be in a later version. They are:

- If you have more than one link or join which connects the same two rooms, then if any of them set the `length` attribute, they must *all* set it—and to the same value. Otherwise IFM will give an error.

- Unsafe tasks in a "follow" task chain normally cause all the previous tasks in the chain to be marked unsafe too (so the solver will avoid trying the first, knowing it'll be forced to do something distasteful later). However, some tasks are not known to be unsafe until just before they might be done—specifically, those for which there is no return path. This is because whether there's a return path depends on where you are now. So a "follow" chain could possibly lead to a game-solving dead end.

- There's a problem with the `leave/need` attribute combination. The game solver could select a path from one room to another in which an item must be left behind at one point, but is needed for movement later on in the path. This sort of path should be invalid, but isn't.

4 Using IFM

4.1 Running the Program

IFM is run from the command line. The general form of the command is:

```
ifm [options] [file...]
```

On startup, IFM does the following. Firstly, the system initialization file is read. This sets defaults used by everyone. This file is called `ifm-init.ifm`, and is found by searching a standard set of directories. You can prepend to this search path by setting the environment variable `IFMPATH` to be a colon-separated list of directories.

Then, if you have a personal initialization file, that is read. This file is found in the directory specified by your `HOME` environment variable. On Unix-like systems it is called `.ifmrc`, and on Win32 systems it is called `ifm.ini`. You can use this file to customize the default variable settings for different types of output.

Then input from the file(s) on the command-line is read. If no files were specified, `stdin` is read. A filename equal to `-` also indicates that `stdin` should be read at that point.

If any of the `-map`, `-items`, `-tasks` or `-show` options was specified, the appropriate output is produced. If not, only a syntax check of the input is done.

When producing output, the output format specified by the `-format` option is used. If this was not specified, the first format in the list which supports this type of output is chosen.

Some of the output formats use additional library files to do their work. For example, the PostScript output format prepends a standard "prologue" file to all output. These files are found using the same search path as the system initialization file (see above).

Here's a summary of the command-line options (which can be abbreviated), starting with the output options:

`-m, -map[=SECTIONS]`

> Draw a map of the game. You can optionally specify a list of the map sections to print. The list is a comma-separated set of map section numbers (starting from 1) and can include ranges. For example, the argument `1,3-5` would print map sections 1, 3, 4 and 5. If the list isn't specified, all sections are printed.

`-i, -items` Print a list of items which appear in the game.

`-t, -tasks` Print a list of tasks required to solve the game.

`-f, -format FORMAT`

> Specify the output format.

`-o, -output FILE`

> Write to the specified file, instead of stdout.

Next comes the auxiliary options:

`-I, -include DIR`

> Prepend the specified directory to the library and include file search path. This option may be repeated.

`-S, -style STYLE`

> Set a global style. See Section 4.3 for more details. This option may be repeated.

`-s, -set VAR=VALUE`

> Set a customization variable. This overrides any settings in the input files. This option may be repeated.

`-noinit` Don't read your personal init file.

`-w, -nowarn` Don't print warnings.

```
-e, -errors NUM
```
Print this many errors before aborting (default: 10). If set to zero, print all errors.

Finally, here are the information options:

`-show TYPE`   Show one of several types of information, and exit. The `TYPE` argument can be one of:

> `maps`   Show a list of all the map sections defined in the input. This is useful for finding the numbers of the map sections you want to print.
>
> `path`   Show the directories that are searched for library and include files.
>
> `vars`   Show a complete list of defined variables, in a format suitable for feeding back into IFM. See Section 4.4.

`-version`   Print the program version.

`-help`   Just print some usage information.

4.2   Types of Output

IFM has three different types of output (a map, a list of items, and a list of tasks) and several different output formats, which are described in the following sections. Not all types of output are produced by each output format. Table 1 shows what's available for each format.

	PostScript	Fig	Tk	Text	Rec	Dot	Raw
Map	Yes	Yes	Yes				Yes
Items				Yes			Yes
Tasks				Yes	Yes	Yes	Yes

Table 1: IFM output formats

All the map output formats display map sections in the same way, so that what you get with one format looks much the same as another.[6]

4.2.1   PostScript Maps (`ps`)

This produces a PostScript map suitable for printing. Several map sections may be printed per page, and the maps are printed over as many pages as it takes. Automatic packing is done to try to get a good fit on the page. Also, portrait or landscape is chosen depending on whichever gives the best fit. Fonts of room and item text are scaled to fit them in the room boxes, if required.

4.2.2   Fig Maps (`fig`)

This produces a map which can be read (and edited) by Xfig, and any other programs which understand Fig format. The map sections are packed to get a best fit automatically, in a similar manner to PostScript, but since Fig has no concept of pages, it is most useful when you're printing each map section individually. There's a utility program called `ifm2dev` which automatically does this—see Section 7.1.

Fig format is also useful if you want to print poster-sized maps over several pages. The `-M` option of `fig2dev` (part of the `transfig` package) will automatically do this.

4.2.3   Tk Drawing Commands (`tk`)

This produces map commands for input to `tkifm`, a simple graphical interface to IFM (see Section 7.3). It isn't very useful to produce this output yourself—`tkifm` does that internally to build its map pictures. But you can control its display by setting variables in the usual way.

---

[6]Well, that's the goal anyway. But there are still some very minor differences.

4.2.4   ASCII Text (`text`)

This produces human-readable output for items and tasks. The output should be fairly self-explanatory.

4.2.5   Recording Commands (`rec`)

This output produces a list of commands suitable for feeding to IF interpreters in playback mode. All the commands in the output are converted to uppercase.

In order for this to work properly, you have to give commands that the game will understand. The `cmd` attribute of rooms, links, joins and tasks can help with this. Currently there's no item `cmd` attribute, so you have to make sure that the item description is recognized by the game (for `get` and `drop` commands). Also, if a task is implicitly done in the game without you having to type any commands (e.g., visiting a room), you can indicate this by using `cmd none`.

Of course, a recording will only play back properly in an interpreter if it provides correct game commands. Random events can't be dealt with by IFM, and will probably cause playback to fail. But you can work around this with an interpreter that is able to fix the random seed at startup (e.g., `frotz`). This should eliminate most (but not all) of the problems of randomness.

4.2.6   Task Dependencies (`dot`)

This produces a graph of the dependencies of tasks on each other, in Graphviz (dot) format. You'll need to have Graphviz installed in order to display the graph; you can get it from `www.graphviz.org`.

4.2.7   Raw Data (`raw`)

This produces raw data for all output formats, intended for use by other programs (and the IFM regression test suite). Each entry consists of a number of data lines, and is separated from other entries by a blank line. Each data line consists of an attribute, a colon, and its value. The attributes should be self-explanatory.

4.3   Customization

You can change the appearance of many output features according to your taste. You do this by setting the values of the variables that control those features. This section tells you how to use variables—for a complete list of the customization variables available, see Section 4.4.

As a first example, the background colour of rooms is determined by the variable `room_colour`. Its default value is "white". It can be changed like this:

```
room_colour = "beige";
```

Setting a variable like this will affect all output formats. But in some cases you don't want to do that. A good example is the one above—if you don't have a colour printer, you may not want to have beige rooms printed (they'll come out greyish). To get around that, you can set variables that are specific to a particular output format:

```
tk.room_colour = "beige";
```

This says to set the variable to "beige" only if producing Tk output. The default for all other formats is still "white".

You can also customize the appearance of individual rooms and links on the map, by using different display styles. A display style is just a group of variable settings with a given name. For example, suppose you're making a map of Colossal Cave and want to mark rooms where you can refill your water bottle. You can define a style called, say, Water, like this:

```
style Water;
room_colour = "light blue";
endstyle;
```

The values of variables that are set between the `style` and `endstyle` clauses only apply to things drawn in that style. Now, if you declare rooms like this:

```
room "At End Of Road";
room "Inside Building" dir e go in style Water;
```

then the room "Inside Building" will be drawn with a light blue background. You can customize individual links in a similar manner.

An alternative way to define a variable in a particular style is to use the `in style` clause, like this:

```
room_colour = "light blue" in style Water;
```

If a style only changes a single variable, this may be more convenient.

If you assign a style (say, called "newstyle") to an object, but don't define it anywhere in your input, then IFM will look for a file called `newstyle.ifm` using the standard search path. If the file exists, it is expected to define style "newstyle". For example, you could put the "Water" style definition above into a file called `Water.ifm` somewhere on the IFM search path, and it would be read automatically. This is useful if, for example, you want to use the same style in several different maps.

You can define global styles using the `-style` command-line option; these apply to all IFM objects. Global styles are most useful when setting variables that affect the overall appearance of the output, in conjunction with the file search method described above (e.g. a file containing general colour and font definitions).

4.4   Output Variables

There are many variables available for customizing output. Most of them are for customizing map output. Their names, descriptions and default values are all contained in the IFM initialization file. You can change this file to set global defaults for everybody, or alternatively set your own preferences in your personal init file.

Here's the original initialization file that came bundled with IFM.

```
############################################################ -*- Perl -*-###
# System IFM init file
##########################################################################

# This file contains the default definitions for all variables used by the
# output formats.  Changing these values will affect all users.

# Each variable listed below has a comment next to it indicating which
# type(s) of output are controlled by it.

# The default values are designed to give nice results in all formats.  The
# default for PostScript is to produce maps for monochrome printing on one
# or more pages of A4 paper.  The default for Fig is to produce one big
# page (which can be split into multiple pages later).

# The map output formats differ in their treatment of fonts.  In PostScript
# and Fig, the font and font size are specified separately, via the *_font
# and *_fontsize variables.  In Tk, they are both specified together, via
# the *_fontdef variables.

# In PostScript and Fig output, a font size represents the maximum desired
# size -- the actual size may be scaled down in order to fit the text into
# an enclosing space (e.g. a room box).

# In Tk output, all line width values are interpreted as integers.

##########################################################################
# General variables
```

```
##########################################################################

# File of colour definitions, which contains the RGB values of each colour
# referred to below (and a good many more).  It's just the rgb.txt file
# found on Unix/X11 systems.  See that file for a list of available
# colours.  You can use the same colour names for the Tk output, since it
# uses the standard X colours.
colour_file = "ifm-rgb.txt"; # ps fig tk

# Prolog file that gets prepended to all the PostScript output.  This
# defines all the procedures for drawing everything.
prolog_file = "ifm-pro.ps"; # ps

# Scale factor which is applied to all fonts.  This is a convenience
# variable to make font adjustments easier.
font_scale = 1; # ps fig

##########################################################################
# Page variables
##########################################################################

# Default page size.  Available page sizes are: A3, A4, A, B, C, Legal,
# Letter.
page_size = "A4"; # ps fig

# If defined, these set a custom page size which overrides the 'page_size'
# variable.  Units are in cm.
page_width = undef; # ps fig
page_height = undef; # ps fig

# Margin space to be left on each page, in cm.
page_margin = 2; # ps fig

# Whether to rotate each page to landscape.  If not defined, then rotation
# is decided on a per-page basis in order to get the best fit.
page_rotate = undef; # ps

# Whether to show the main title on each page.
show_page_title = 1; # ps

# Appearance of the main title (if shown).
page_title_font = "Times-Bold"; # ps
page_title_fontsize = 18; # ps
page_title_colour = "black"; # ps

# Whether to show a border around each page.
show_page_border = 0; # ps fig

# Colour of the page border (if drawn).
page_border_colour = "black"; # ps fig

# Colour of the page background (if border is drawn).
page_background_colour = "white"; # ps fig

# Whether to scale Fig layout to fit on a single page.  Most of the time,
# this doesn't make things look good.
fit_page = 0; # fig

##########################################################################
# Map section variables
```

20

```
#########################################################################

# Whether to show the map title.
show_map_title = 1; # ps fig

# Appearance of the title printed above each map section.
map_title_font = "Times-Bold"; # ps fig
map_title_fontsize = 14; # ps fig
map_title_colour = "black"; # ps fig

# Whether to show a border around each map section.
show_map_border = 0; # ps fig

# Colour of the map border (if drawn).
map_border_colour = "black"; # ps fig

# Colour of the map background (if border is drawn).
map_background_colour = "white"; # ps fig
tk.map_background_colour = "wheat"; # tk

# Minimum space, in rooms, between map sections when packed together.
map_section_spacing = 1; # ps fig

# Maximum width and height of the Tk map canvas window, in rooms.  Sizes
# bigger than this will cause scrollbars to appear.
map_canvas_width = 8; # tk
map_canvas_height = 6; # tk

#########################################################################
# Room variables
#########################################################################

# Space allocated for each room, in cm.  In PostScript and Fig, this is the
# maximum size -- the actual size may be reduced in order to fit things on
# the page.
room_size = 3; # ps fig tk

# Proportion of the room space that's taken up by the room dimensions.
# These values should be less than 1 or you'll have no space left for link
# lines.
room_width = 0.8; # ps fig tk
room_height = 0.65; # ps fig tk

# Default background colour of rooms.
room_colour = "white"; # ps fig tk

# Offset and colour of room 'shadows'.  This is a pseudo-3D effect which
# makes rooms look raised off the page.  The offsets are a proportion of
# the allocated room space.  Note that you can change the direction of the
# room shadow by negating one or both of the offsets.
room_shadow_xoff = 0.05; # ps fig tk
room_shadow_yoff = 0.05; # ps fig tk
room_shadow_colour = "grey50"; # ps fig tk

# Width and colour of the room box lines.
room_border_width = 1; # ps fig tk
room_border_colour = "black"; # ps fig tk

# Whether to draw dashed room borders.
room_border_dashed = 0; # ps fig
```

21

```
# Width and colour of the room exit lines.
room_exit_width = 1; # ps fig tk
room_exit_colour = "black"; # ps fig tk

# Colour, font and fontsize of room description text.
room_text_colour = "black"; # ps fig tk
room_text_font = "Times-Bold"; # ps fig
room_text_fontsize = 10; # ps fig
room_text_fontdef = "Times 10 bold"; # tk

# Whether to show non-hidden item descriptions in rooms.
show_items = 1; # ps fig tk

# Colour, font and fontsize of item description text (if shown).
item_text_colour = "black"; # ps fig tk
item_text_font = "Times-Italic";# ps fig
item_text_fontsize = 6; # ps fig
item_text_fontdef = "Times 8 italic"; # tk

# Whether to append room tag names to room descriptions.  If so, they are
# appended in square brackets, like [this].
show_tags = 0; # ps fig tk

#############################################################################
# Link style variables
#############################################################################

# Width and colour of link lines.
link_line_width = 1; # ps fig tk
link_colour = "black"; # ps fig tk

# Size of oneway link arrows, as a proportion of the allocated room space.
link_arrow_size = 0.1; # ps tk

# Whether to draw link lines as splines.
link_spline = 1; # fig tk

# Whether to draw dashed link lines.
link_dashed = 0; # ps fig

# Appearance of text that's associated with link lines.  The Tk default is
# to colour them differently for readability.
link_text_font = "Times-Roman"; # ps fig
link_text_fontsize = 6; # ps fig
link_text_fontdef = "Times 8 bold"; # tk
link_text_colour = "black"; # ps fig
tk.link_text_colour = "red"; # tk

# Text strings indicating up/down/in/out on links.  PostScript is currently
# a special case: the strings either side of the '/' are extracted and
# printed at either ends of the link, near the room they come from.
link_updown_string = "U/D"; # ps fig tk
link_inout_string = "I/O"; # ps fig tk

#############################################################################
# Join style variables
#############################################################################

# Whether to indicate joins in the room text.
```

22

```
        show_joins = 1; # ps fig tk

        # Join string format (gets put in parentheses in those rooms that join to
        # other rooms).  The value should be "number" or "letter".
        join_format = "number"; # ps fig tk

        #########################################################################
        # Game solver variables
        #########################################################################

        # These variables modify the overall behaviour of the game solver.

        # Whether to keep unused items (i.e. those which were obtained via some
        # task or other, but currently have no use).
        keep_unused_items = 1; # text rec

        # Whether to treat all tasks as safe (reckless mode!).
        all_tasks_safe = 0; # text rec

        # Whether to print game solver info messages (helps with figuring out what
        # it's up to).
        solver_messages = 0; # text rec

        #########################################################################
        # Task dependency variables
        #########################################################################

        # Whether to group tasks by the room they're done in.  This can either
        # enhance the task structure or make it look a complete mess.
        task_graph_rooms = 0; # dot

        # Whether to show orphan tasks (those with no previous/next dependencies).
        # Useful for completeness, but it clutters things up a lot.
        task_graph_orphans = 0; # dot

        # Graph attributes, in Graphviz format.
        task_graph_attr = ""; # dot

        # Node attributes, in Graphviz format.
        task_graph_node = "shape=box";  # dot

        # Link attributes, in Graphviz format.
        task_graph_link = ""; # dot

        #########################################################################
        # End of initialization file
        #########################################################################
```

4.5   Predefined Styles

IFM comes with a few predefined style files, as shown in Table 2.[7] The "scope" field indicates which type of IFM object it applies to. Styles that have global scope can meaningfully be used by the -style command-line option.

---

[7]If you create any generally useful or nice-looking styles, you might want to send me a copy so I can include them with the next version of IFM.

Style	Scope	Description
helvetica	Global	Use Helvetica fonts everywhere in maps
puzzle	Room	Mark room as containing a puzzle
special	Link	Mark link as being special in some way
reckless	Global	Treat all tasks as safe when solving the game
verbose	Global	Print verbose solver messages

Table 2: Predefined styles

5 Language

This section gives a complete detailed description of the IFM language.

5.1 Symbols

In the following sections, these symbols are used:

ID   A name starting with a letter, followed by any combination of upper- and lowercase letters or numbers or an underscore. For example, Dense_Forest. IDs are not allowed to clash with reserved words (e.g., room, or tag). One way to avoid this is to capitalize all tags (reserved words are all in lowercase).

STRING   Any sequence of characters, in double-quotes. For example, "Black Rod". To get a double-quote inside a string, quote it using a backslash, like this:

      "Ground Floor, \"A\" Block"

You can continue strings on several lines—a newline-and-whitespace sequence is translated into a single space, just like in TADS and Inform.

NUMBER   A number. If the context requires an integer, the number is silently rounded to the larges integer not greater than this value.

COMPASS   A compass direction, which can be abbreviated or in full (e.g. n, se, northwest, etc).

OTHERDIR   One of up, down, in or out.

[...]   An optional part.

A | B   Either A or B.

5.2 Format

IFM generally has a free-format layout—i.e., whitespace may be placed anywhere to increase readability. The only exception is inside quoted strings, where spaces are significant. Comments may be added, starting with a hash (#) and continuing to the end of the line. All commands are terminated with a semicolon.

5.3 Control

The overall title of the map may be set using the command

      title STRING;

If a map has several sections, you can set the title of each section using the command

      map STRING;

This sets the title of the next map section. If you use this command at all, then the number of uses should be the same as the actual number of map sections. It's conventional (but not required) to put the `map` command just before the room that starts a new map section.

If your map uses features that are only present in later versions of IFM, you can indicate that up front by using the command

```
require NUMBER;
```

Then, if the IFM version number is less than this number, parsing will abort immediately, avoiding lots of potentially confusing syntax errors.[8]

5.4 Tags

All IFM objects may be given tag names, so that you can refer to them in other commands. Tags for different types of object are independent—for example, you could have a room and an item with the same tag. However, tags for similar objects must be unique.

In many cases, you are allowed to refer to a tag name anywhere, even earlier in the file that you defined it (as long as the tag is defined somewhere!). Exceptions are the room `from ID` clause (see Section 5.5.1) and tags in commands that modify existing objects (see Section 5.5)—these tags must be defined before being used.

The special tag `last` may be used to refer to the last object that was defined (depending on the context). Also, within an individual command, the special tag `it` may be used to refer to the most recent object tag.

5.5 Commands

There are five different types of object in IFM: rooms, items, links, joins and tasks. Each is created using its own command, the general format of which is:

```
<type> <body> [attribute-list];
```

For rooms, items and tasks, `<body>` is just a string description. For links and joins, it specifies two room tags to link or join together.

Many of the attributes or objects accept a list of tags as arguments. All of these, if specified more than once in the same object, concatenate the lists together.

Once an object as been declared with a tag name, its attributes can be modified by later commands referring to that tag, like this:

```
<type> ID [attribute-list];
```

where `ID` is the tag name of the object. Note that the tag must be defined earlier in the file than it is used.

5.5.1 Rooms

A new room is added using the command

```
room STRING [attribute-list];
```

where `STRING` is a description of the room. Room attributes can be:

`tag ID`   Give the room a tag name, so that you can refer to it elsewhere.

`dir COMPASS [NUMBER] [COMPASS [NUMBER]...] [from ID]`
    Specify the position of the room relative to the room with the given tag ID (which must be defined before it is used). If no `from` clause is specified, the last defined room is used instead. There can be more than one direction given—the new room is placed relative to the previous one using them. Following a direction with a number means to repeat it that many times.

---

[8]There's no point in requiring any version less than 5.0, since that's when the `require` syntax was added.

The `dir` clause creates an implicit link between this room and the previous one. Some of the room attributes below behave differently depending on whether they appear before or after the `dir` clause in the attribute list.

If the room is given a tag name, then the implicit link will be given the same tag.

link ID [ID...]
Specify other rooms that this room links to. Note that this creates a link with no special attributes—use the standalone `link` command for that (see Section 5.5.3).

join ID [ID...]
Specify rooms on other map sections that this room joins to. Note that this creates a join with no special attributes—use the standalone `join` command for that (see Section 5.5.4).

exit COMPASS [COMPASS...]
Indicate which other directions the room has exits in. Room exits in a particular direction are marked on the map only if there is no link going to or from the room in that direction.

note STRING
Append a note to the room's note list.

score NUMBER
Indicate that you score the specified number of points when visiting this room for the first time.

need ID [ID...]
If this appears before a `dir` clause, indicate that you can only enter this room after getting the specified items. If it appears afterwards, it applies to the implicit link instead.

after ID [ID...]
If this appears before a `dir` clause, indicate that you can only enter this room after doing the specified tasks. If it appears afterwards, it applies to the implicit link instead.

before ID [ID...]
If this appears before a `dir` clause, indicate that you can only enter this room before doing the specified tasks. If it appears afterwards, it applies to the implicit link instead. Those tasks are marked unsafe.

leave ID [ID...]
If this appears before a `dir` clause, indicate that the specified items, if carried, must be left behind when entering the room. If it appears afterwards, it applies to the implicit link instead.

leave all [except ID [ID...]]
As above, except indicate that *all* items must be left behind. The `except` clause can be used to omit specific items.

go OTHERDIR
Indicate that the link to this room is in the specified direction.

cmd STRING
Specify the command you type to move to this room from the previous one. If no `cmd` clause is given, the command is deduced from the `go` clause. If that isn't specified, the command will be deduced from the `dir` clause.

cmd from STRING
As above, but this specifies the command to go in the other direction. This defaults to the `cmd to` command, if specified.

cmd to STRING
This is identical to `cmd` on its own, and only exists for symmetry.

oneway   Indicate that the return journey from this room to the previous one is not possible.

length NUMBER
> Indicate that the direction link to this room has the specified length (default 1). This only affects the calculation of the nearest task when solving the game—see Section 5.5.5.

start
> Indicate that this is the room the player starts in. Default is for the first room mentioned to be the start room. If more than one room has this attribute, the last one declared takes precedence.

finish
> Indicate that entering this room finishes the game.

nodrop
> Indicate that no items should be voluntarily dropped in this room.

nolink
> Indicate that this room does not have an implicit link with the previous one via the `dir` clause.

nopath
> Indicate that the implicit link from this room should not be used by the game solver.

style ID [ID...]
> Add a list of display styles to the room (and also the implicit link, if any). See Section 5.7.

5.5.2   Items

An item is introduced using the command

```
item STRING [attribute-list];
```

where STRING is the item description. Item attributes can be:

tag ID
> Give the item a tag name, so you can refer to it elsewhere.

in ID
> Set the initial location of this item. Default is the last defined room. If there is no last room (i.e. an item was declared before any room was declared), then this item is initially carried by the player.

note STRING
> Append a note to the item's note list.

score NUMBER
> Indicate that you get points the first time you pick this item up.

hidden
> Indicate that this item is not immediately obvious when entering the room.

keep
> Indicate that this item shouldn't ever be dropped (no "drop" task should be generated).

keep with ID [ID...]
> Indicate that the item shouldn't be dropped until all the other specified items have left the inventory.

keep until ID [ID...]
> Indicate that the item shouldn't be dropped until all the other specified tasks have been done.

ignore
> Indicate that this item should be ignored when trying to find a solution (i.e., never go out of your way to pick it up).

given
> Indicate that this item didn't have to be picked up when it entered the inventory (no "get" task should be generated). This attribute is obsolete—you should use the task `give` clause instead (see Section 5.5.5).

lost
> Indicate that this item wasn't dropped when it left the inventory (no "drop" task should be generated). Normally you should use the task `drop` or `lose` clauses instead (see Section 5.5.5). The only use for this attribute is for items that are left behind due to a `leave` clause.

need ID [ID...]
> Indicate that you can only pick this item up after getting the specified items.

after ID [ID...]
>   Indicate you can only pick this item up after the specified tasks are done.

before ID [ID...]
>   Indicate you can only pick this item up before the specified tasks are done.

finish   Indicate that getting this item finishes the game.

style ID [ID...]
>   Add a list of display styles to the item. See Section 5.7.

5.5.3 Links

You can create extra room links using the command

>   link ID to ID [attribute-list];

and the following attributes may be specified:

tag ID   Give the link a tag name, so you can refer to it elsewhere.

dir COMPASS [COMPASS...]
>   Set the intermediate directions that this link travels in, in the same manner as for rooms. Note that if you omit the final direction to the linked room, it is added automatically.

go OTHERDIR
>   Indicate that the link is in the specified direction.

cmd STRING
>   Specify the command you type to go in this direction. If no cmd clause is given, the command is deduced from the go clause. If that isn't specified, the command will be deduced from the dir clause.

cmd from STRING
>   As above, but this specifies the command to go in the other direction. This defaults to the cmd to command, if specified.

cmd to STRING
>   This is identical to cmd on its own, and only exists for symmetry.

oneway   Indicate that this is a one-way link, in a similar manner to the room attribute of the same name.

hidden   Indicate that this link should not be drawn on the map. Hidden links are still used when solving the game.

nopath   Indicate that this link should not be used by the game solver.

length NUMBER
>   Indicate that this link has the specified length (default 1). This only affects the calculation of the nearest task when solving the game—see Section 5.5.5.

need ID [ID...]
>   Indicate that you can only go in this direction after getting the specified items.

after ID [ID...]
>   Indicate that you can only go in this direction after doing the specified tasks.

before ID [ID...]
>   Indicate that you can only go in this direction before doing the specified tasks. These tasks are marked unsafe.

leave ID [ID...]
> Indicate that the specified items, if carried, must be left behind when using this connection.

leave all [except ID [ID...]]
> As above, except indicate that *all* items must be left behind. The except clause can be used to omit specific items.

style ID [ID...]
> Add a list of display styles to the link. See Section 5.7.

5.5.4   Joins

There is a standalone join command which joins two rooms on different map sections:

> join ID to ID [attribute-list];

The following attributes may be specified:

tag ID
Give the join a tag name, so you can refer to it elsewhere.

go COMPASS | OTHERDIR
> Indicate that the join to this room is in the specified direction.

cmd STRING
> Specify the command you type to go in this direction. If no cmd clause is given, the command is deduced from the go clause. If that isn't specified, the command will be undefined.

cmd from STRING
> As above, but this specifies the command to go in the other direction. This defaults to the cmd to command, if specified.

cmd to STRING
> This is identical to cmd on its own, and only exists for symmetry.

oneway
Indicate that this is a one-way join, in a similar manner to the room attribute of the same name.

hidden
Indicate that this join should not be drawn on the map. Hidden joins are still used when solving the game.

nopath
Indicate that this join should not be used by the game solver.

length NUMBER
> Indicate that this join has the specified length (default 1). This only affects the calculation of the nearest task when solving the game—see Section 5.5.5.

need ID [ID...]
> Indicate that you can only go in this direction after getting the specified items.

after ID [ID...]
> Indicate that you can only go in this direction after doing the specified tasks.

before ID [ID...]
> Indicate that you can only go in this direction before doing the specified tasks. These tasks are marked unsafe.

leave ID [ID...]
> Indicate that the specified items, if carried, must be left behind when using this connection.

leave all [except ID [ID...]]
> As above, except indicate that *all* items must be left behind. The except clause can be used to omit specific items.

style ID [ID...]
> Add a list of display styles to the join. See Section 5.7.

5.5.5 Tasks

You can indicate tasks which need to be done in order to solve the game using the command

```
task STRING [attribute-list];
```

and these are the available attributes:

tag ID   Give the task a tag name, so you can refer to it elsewhere.

in ID   Specify the room the task must be done in. If this clause is omitted, it defaults to the last defined room. You can use the special word any to indicate that the task may be performed anywhere. A task declared before any room is equivalent to saying in any.

need ID [ID...]
                Indicate that the specified items are required before you can do this task.

after ID [ID...]
                Indicate that this task can only be done after all the specified tasks have been done.

follow ID
                Indicate that this task must be done immediately after the specified one. Not even a "drop item" task is allowed in between.

do ID [ID...]
                Indicate that doing this task also causes the specified other tasks to be done (if they aren't done already). These other tasks are done immediately, without regard for any prerequisite items or tasks they might need, and their effects are carried out—including any "do" clauses they might have, recursively.

get ID [ID...]
                Indicate that doing this task enables you to get the specified items, and must be done before you can get them.

give ID [ID...]
                Indicate that doing this task puts the specified items straight into your inventory, wherever they happen to be.

lose ID [ID...]
                Indicate that doing this task causes the specified items to be lost. This implies that all tasks which need these items must be done before this one.

drop ID [ID...] [in ID] [until ID [ID...]]
                Indicate that doing this task drops the specified items in the current room (or the room indicated by the in clause) if you're carrying them. No "drop" message is generated. If there's an until clause, you can't retrieve the items until the specified tasks have been done.

drop all [except ID [ID...]] [in ID] [until ID [ID...]]
                As above, but drop everything you're carrying. The except clause can be used to omit specific items.

goto ID   Indicate that you get "teleported" to the specified room when this task is done. This happens after "give" and "drop" actions.

safe   Mark this task as safe—i.e. one that can't cause the game solver to get stuck.

ignore   Don't ever do this task explicitly when solving the game. The task may still be done via a "do" action.

finish   Indicate that doing this task finishes the game.

```
score NUMBER
```
        Indicate that you get the specified score for doing this task.

```
note STRING
```
        Append a note to the task's note list.

```
cmd STRING [NUMBER]
```
        Specify the exact command you type to do the task. If a number follows the command, do the command that many times. Multiple `cmd` clauses concatenate into a list of commands.

```
cmd none
```
        Indicate that no command is required to do this task.

```
style ID [ID...]
```
        Add a list of display styles to the task. See Section 5.7.

5.6 Variables

Various aspects of output are controlled by variables. These are set using the following syntax:

```
[FORMAT.]ID = NUMBER | STRING | undef [in style ID];
```

`FORMAT`, if specified, is the name of a specific output format—the variable then applies only to that output format. `ID` is the name of the variable, and it can take a numeric or string value. Note that setting a variable to the value `undef` effectively removes it. If the style clause is present, this means to only set the variable to this value in the specified style (see Section 5.7).

5.7 Styles

A "style" defines a set of variables with particular values, so that those values can be referred to together. IFM keeps track of the currently active list of styles, and there are two commands which change this list. The command

```
style ID;
```

pushes the specified style onto the style list. This style becomes the current style. Any IFM objects declared while a style list is in force will by default be output in those styles. Any variable setting is by default in the current style (though you can specify a particular style using the `in style` clause—see Section 5.6).

    The command

```
endstyle [ID];
```

pops the current style from the style list. The previous style on the list (if any) becomes the current style. The ID, if specified, should match the ID in the corresponding `style` command, or a warning is given.

    Each display style has its own set of values for customization variables. On output, when the value of a variable is needed for displaying an object, the style list for that object is searched in reverse order of declaration. The value used is from the first style to define this variable. If no style defines it, then the default value is used.

    If a style is referenced by an object but not defined anywhere in the input, then its definition is assumed to be in a separate file, which is searched for using the standard search path. The name of this file is formed by adding a `.ifm` suffix to the style name. If the file is not found, or it does not define the required style, a warning is given.

6 Diagnostics

This section describes the possible error and warning messages which might be produced by IFM, and what they mean. Note that individual output formats may print their own errors and/or warnings. These lists only cover the standard ones.

6.1 Error Messages

Here's the list of error messages. If any errors occur, no output is produced.

error: invalid repeat count
> You've given a repeat count of zero or less for a string or direction, which doesn't make much sense.

error: no last room
> You've given the very first room a `dir` clause.

error: no [type] referred to by 'last'
> You've said `last` to refer to the last room, item or task that was defined, but none of that type of object have been defined yet.

error: no [type] referred to by 'it'
> You've said `it` to refer to the last room, item or task tag that was mentioned in the current command, but no tags of that type of object have been mentioned.

error: [type] tag [name] already defined
> You've given two similar objects the same tag name.

error: [type] tag [name] not defined
> You've referred to a tag name that hasn't been defined anywhere in the input.

error: [type] tag [name] not yet defined
> You're referring to a tag at a point where it hasn't yet been defined, in a situation where it must be (e.g. the room `from` clause, or a command that modifies attributes of a previously-defined object).

error: can't modify [name] attribute
> You're attempting to modify an attribute of an object which can't be changed once it's set (e.g. a tag name). This is because it would create inconsistencies between objects.

error: can't link [name1] and [name2] -- different map sections
> The rooms you're trying to link are on different sections of the map, and have no spatial relation to one another. You might have forgotten to link a previous room in the list. Or you meant to use a `join`.

error: can't link [name] to itself without at least one direction
> To link a room to itself, you need to specify at least one direction. Otherwise, the link goes nowhere.

error: links between [name1] and [name2] have differing lengths
> You've defined more than one link or join between the given rooms, but given them different values for the `length` attribute. This isn't allowed.

error: more than one task needs to follow [task] immediately
> You've given two or more tasks an identical `follow` tag. Only one task can "follow" a given task.

error: [num] cyclic task dependencies
> The game isn't solvable because there's one or more chains of tasks where each must be done before the next, but the last must be done before the first.

`error: variable [name] is not defined`
>   A customization variable needed by an output format is not defined. You should only see these
>   errors if you have modified or overridden the system initialization file. The remedy is to define
>   the variable somewhere.

6.2   Warning Messages

Here's the list of warning messages. If only warnings occur, then output is still produced.

`warning: attribute [attr] ignored -- no implicit link`
>   You've given a room with no `dir` clause an attribute that is associated with that link (e.g. `oneway`).
>   Most likely you're putting the attribute in the wrong place—if you want, say, a join to have one
>   of these attributes, you must define it using the standalone `join` command instead.

`warning: link from [name1] to [name2] outside grid`
>   The destination room for a link is not in a compass direction from the last specified position.

`warning: rooms [name1] and [name2] overlap`
>   The coordinates of the specified rooms are the same.

`warning: room [name] crossed by link line between [name] and [name]`
>   A link line passes through the coordinates of a room.

`warning: room [name] has multiple [dir] links`
>   More than one link connects to the specified room in a particular direction.

`warning: can't solve game ([num] tasks not done)`
>   The game is unsolvable according to the current set of tasks. This can be due to part of the map
>   being inaccessible, or IFM stupidly choosing the wrong order of doing things. Hopefully the
>   latter shouldn't happen very often.

`warning: can't solve game ([num] tasks ignored)`
>   The game is unsolvable because you're explicitly ignoring tasks and/or items, using the `ignore`
>   attribute.

`warning: no matching style command`
>   You've used `endstyle` without a matching `style`. You probably have too many `endstyle`
>   commands.

`warning: unexpected style: [name] (expected [name])`
>   You've used `endstyle` with an argument that doesn't match the argument of the corresponding
>   `style`. You might have missed out another `endstyle` somewhere, or have too many.

`warning: style [name] referenced but not defined`
>   An object in your input uses the specified style, but it isn't defined anywhere and the style
>   definition file `name.ifm` doesn't exist in the search path (or if it does, it doesn't define the
>   required style).

`warning: [syntax] is obsolete -- use [phrase] instead`
>   You've used an obsolete syntax. Consult the documentation and then try the suggested al-
>   ternative instead. Note that multiple uses of the same obsolete syntax only result in a single
>   warning.

7   Other IFM Programs

There are several other programs bundled with IFM, which you might find useful.

7.1   `ifm2dev`—convert IFM maps to various other formats

`ifm2dev` is a front end to `fig2dev`, which converts diagrams in Fig format to various other formats. `ifm2dev` converts each map section and writes them to separate files. The command format is:

```
ifm2dev [-o template] [fig2dev-options] [-- ifm-options] file
```

The `-o` option sets the file template for the output files. It must contain an integer format indicator (e.g. `%d`); this is replaced with the map section number in filenames. If not set, it defaults to `prefix-%02d.suffix`, where `prefix` is the file prefix of the input file (with `.ifm` removed), and `suffix` is the suffix appropriate for the type of output.

All `fig2dev` output options are passed through without change. You can use the `-L` option of `fig2dev` to set the output format. See the `fig2dev` manpage for details.

You can supply options to IFM by first giving the end-of-options indicator (`--`) and then the options.

7.2   `ifm2tex`—convert IFM maps to LATEX

`ifm2tex` is a program for creating a summary file in LATEX format, suitable for converting to PostScript and printing. It uses `ifm2dev` to create the maps. The command format is:

```
ifm2tex [-m] [-i] [-t] [-- ifm-options] file
```

The options indicate which combinations of things you want output (maps, items or tasks). Several files are written, whose names depend on the prefix of the input file (i.e., minus its `.ifm` suffix):

`prefix.tex`	The main LATEX input file, which includes all the others.
`prefix-maps.tex`	The maps, which include the EPS figures.
`prefix-items.tex`	The table of items.
`prefix-tasks.tex`	The table of tasks.
`prefix-map-N.eps`	An EPS figure for each map section.

At the moment, the program is very basic and doesn't have any options to control anything. But you can sort of customize things by using your own main LATEX file and doing your own including of the other files.

7.3   `tkifm`—create maps interactively

`tkifm` is a graphical front end to IFM. It provides a text editing window in which you can map out your game using IFM commands, and a set of menus to view things. The various features are as follows:

Text editing window	This is the main window, where you type IFM commands. It provides all the usual text editing command bindings that come with the Tcl/Tk text widget, as well as syntax highlighting.
File menu	The standard set of file commands: New, Open, Save, Save-As, Export (to PostScript or Fig), Quit. There's also a command called Redraw, which invokes IFM on the current file again. Normally you don't have to use this (it's done whenever you open a new file or save the current one), but if you change your initialization file (see below) the changes won't be noticed unless you do a Redraw.
Map menu	For each map section in your map, a menu entry appears here. Selecting it will draw the appropriate map in another window.

Item menu	This contains a single menu item, which displays a list of items in another window.
Task menu	This contains two menu items: "Task list (brief)", which displays a high-level walk-through of the game in another window, and "Task list (verbose)" which does the same but gives detailed information about what the game solver is up to.
Show menu	This contains two menu items: "Variables", which shows all the currently defined variables and their values, and "Paths", which shows the file search path.
Help menu	This displays a small info panel about the current IFM version, including copying restrictions.

7.3.1   Using Your Own Editor

If you'd like to use your own editor to edit IFM files, but still view results with `tkifm`, you can. `tkifm` recognizes when the file it is visiting gets modified, and rereads it if so. If you like, you can also disable all `tkifm` file modification commands by setting the `ifm(edit)` variable to zero (see below). This is probably a good idea if using another editor, or else you might accidentally save from `tkifm` and lose all your changes.

7.3.2   Customization

On startup, `tkifm` reads an initialization file in your home directory (the one pointed at by the `HOME` environment variable). On Unix systems it is called `.tkifmrc`, and on Win32 systems it is called `tkifm.ini`. From there, using Tcl commands, you can set various things that affect the appearance of the program. Here's an example file showing the valid variables, their format and defaults.

```
# Example tkifm init file.

# Whether to allow editing.
set ifm(edit) 1

# Edit window dimensions.
set ifm(editwidth)  80
set ifm(editheight) 24

# Editing font.
set ifm(editfont) {Courier 12 bold}

# Edit window colours.
set ifm(editforeground) black
set ifm(editbackground) wheat

# Item list window dimensions.
set ifm(itemwidth)  50
set ifm(itemheight) 30

# Task list (brief) window dimensions.
set ifm(taskwidth)  50
set ifm(taskheight) 30

# Task list (verbose) window dimensions.
set ifm(verbosewidth)  80
set ifm(verboseheight) 30

# Variable window dimensions.
set ifm(varswidth)  50
set ifm(varsheight) 30

# Text window font.
set ifm(textfont) {Times 12 bold}
```

```
# Text window colours.
set ifm(textforeground) black
set ifm(textbackground) wheat

# Whether to allow tearoff menus.
set ifm(tearoff) 1

# Syntax highlighting variables.
set ifm(syntaxcomments)     firebrick
set ifm(syntaxstrings)      grey40
set ifm(syntaxstructure)    blue
set ifm(syntaxdirections)   darkgoldenrod
set ifm(syntaxspecial)      cadetblue
set ifm(syntaxbuiltin)      forestgreen
set ifm(syntaxkeyword)      royalblue
set ifm(syntaxpreprocessor) purple
```

7.3.3   Errors and Warnings

Any errors or warnings that occur when invoking IFM will be displayed in a dialog. The current line of the text window will be changed to point at the error or warning line (if appropriate).

7.4   `scr2ifm`—convert transcripts to IFM map

scr2ifm reads one or more transcripts of an Interactive Fiction game and produces a map of it in IFM format. It works on Infocom-style transcripts—those produced by Inform- and TADS-generated games (and of course Infocom games themselves). The idea is that you play your game in a carefree manner, without worrying about mapping anything, and after the session use scr2ifm to get a map. Well, that's the theory anyway.

scr2ifm offers two basic ways of mapping: you can create an initial (incomplete) map from your first transcript and then add the rest "by hand", or you can create a set of transcripts, each exploring different parts of the game, and then merge them all into a final map. Which you choose is up to you.

7.4.1   Options

These are the command-line options for scr2ifm:

-c file   Append the given file of IFM and parameter commands to the end of the map. This allows you to fix various problems in the resulting map.

-o file   Write output to the given file instead of stdout.

-i   Indent the output nicely (according to *my* definition, anyway). This makes things look better if you've added item and task commands. Default is no indentation at all.

-l   Add a comment to each IFM command indicating the file and line number of the transcript command that generated it.

-w   Don't print warnings.

-h   Print a short usage message and exit.

7.4.2   Operation

scr2ifm works by recognizing several key things in the transcript:

- The commands you type. These are easy to spot, because they are preceded by a prompt (usually >).

36

- When the current room changes. This is not quite so easy, but still fairly simple. When you enter a new room, a short room title is printed (for example, "West of House"). The program looks for these short titles in the text that follows a command. First, a line is checked for an invalid format—things that never appear in a title (e.g., "?" or "!"). Then the word count is checked—titles are hardly ever more than 7 or 8 words long. Finally the maximum length of an uncapitalized word is examined—this is almost always 3 or less in titles (the length of "the").

- When a previously-visited room is visited again. This is the most difficult thing to determine, since anything might have changed in the room since the last visit. If there is a description, an exact description match is tried. If that fails, a substring match is attempted. If *that* fails, the first few words are examined to see if they match. If there's no description, an exact room name match is tried. This isn't as reliable, which is why you should always create a transcript in "verbose" mode.

- Special IFM commands that are (hopefully) ignored completely by the game.

Some of these checks can be tailored to a particular transcript.

7.4.3   Making Transcripts

For best results with `scr2ifm`, you should follow these guidelines when making a transcript:

- Always use "verbose" mode. If you don't, then when you revisit a room you've been in before, no description will be printed. In that case, `scr2ifm` will have to rely on the room name to tell where it is. If there's more than one room with that name, it'll get confused (and so will you when you look at the map!).

- After starting the script, look around. Otherwise your starting room may not be included on the map, since the room description may not get printed.

- If there's a bend in a link (e.g., if you leave a room by going east, and enter the next room from the south) make sure you do the return journey. Otherwise, `scr2ifm` won't know about the bend.

- Avoid places where different rooms have the same room description (i.e., mazes). `scr2ifm` can't resolve cases like this, and will assume there's just a single room. Note that this doesn't exclude *all* mazes—the "twisty passages, all different" maze from Colossal Cave would still work, since the descriptions are different.

- If you play the game over several sessions, and create several separate transcripts, pay attention to the starting rooms. If a transcript starts in a room already visited in the previous transcript, then its rooms will be added to the same map section. If not, a new map section will be created (and you should use the `map` command to give it a name).

Some games will be more amenable to `scr2ifm` than others—in general the ones with reasonable map connections, where each connection makes sense in relation to the others. For these games, the order in which rooms are visited will probably make no difference to the resulting map. But for perverse games (e.g. Colossal Cave, above ground) the order of visiting makes a big difference. In these cases, it's probably best to avoid trying to get all the connections on the map—you'll spew forth lots of IFM warnings otherwise, and the map will look awful. Sometimes it's worth having a second attempt at a transcript, choosing different directions to go in, to see if it improves things.

Another problem with mapping is that some games have these inconvenient things called puzzles, which sometimes block you from exploring everywhere so you can map it properly. Come on, IF authors, let's get our priorities right!

7.4.4   IFM Commands

While you're making your transcript, you can use a subset of IFM commands to add extra things that `scr2ifm` will recognize. (Of course, the game almost certainly won't understand these commands, and will print a suitable indignant reply. But that shouldn't affect the state of things.) The commands recognized are:

title <name>	Give the map a title. The name must be in double-quotes. This command can appear anywhere, and has the same effect each time.
map <name>	Indicate that the current room starts a new map section with the given name. The name must be in double-quotes. This is useful if it seems like the map divides into different sections (e.g., the floors of a house).
	If you use the map command, you'll end up with two or more map sections. The first map section is the one that contains the very first room, and will have a default name unless you give it one using another map command.
item <name> [attrs]	Declare an item in the current room. The name must be in double-quotes. If you don't give it a tag attribute, one is generated automatically by changing all invalid tag characters in the name to underscores.
item [attrs]	Add the given attributes to the last declared item, if any.
item delete	Delete the last declared item, if any.
task <name> [attrs]	Declare a task in the current room in a similar manner to items. The name must be in double-quotes.
task [attrs]	Add the given attributes to the last declared task, if any.
task delete	Delete the last declared task, if any.

Note that scr2ifm knows almost nothing about IFM command syntax. If there's a syntax error in a command, you'll have to manually edit the resulting transcript (or delete the item/task and do it again).

7.4.5   Fixing Problems

The map produced by scr2ifm will not be perfect in a lot of cases. When running it through IFM you might get warnings about overlapping rooms, or crossed link lines. These problems are usually fixed by stretching the link lines of some of the rooms. Some overlapping-room problems are caused by the program making a bad choice of direction to put an up/down/in/out connection in. Another problem might be not recognizing when a room is the same as a previously-visited room, because the room description changed too much between visits.

   You can fix most of these problems by creating a command file and specifying it with the -c option. There are two types of entry: scr2ifm commands and IFM commands. In a command file, blank lines and comment lines (those starting with a #) are ignored. If a command isn't recognized as a scr2ifm command, it's assumed to be an IFM command, and is passed straight into the output verbatim. You can use these to resolve conflicts in the generated map, by stretching certain links or hiding them altogether.

   Here's a list of the commands available:

is_room TEXT	Declare a line of text that is actually the name of a room. Normally room names get recognized properly, but there may be some special cases that aren't.
not_room TEXT	Declare a line of text that definitely isn't the name of a room. This is for opposite cases to the above.
use_name TEXT	Use only the room name to decide whether a room has been previously visited. This is for cases where a room description changes too much between visits to be called the same room. Note that this implies that there can only ever be one room with this name.
set_option LETTER	This is a convenience that lets you specify scr2ifm command-line options in the command file instead. E.g. set_option i will indent output nicely.

set_param VAR = VALUE   This evaluates the given perl expression. You can use this to set various parameters that control how the transcript is parsed (see below). Setting a value of `undef` will remove that parameter (so that it won't be used).

Here's a list of the parameters that control how the transcript is parsed, and their defaults. You can use the scr2ifm `set_param` command to set them in the command file.

$name_remove   Matched text to remove before seeing if a line of text is a room name. This is for getting rid of stuff like " (in the dodgem car)". Default: `\s+\(.+\)`

$name_maxwords   Maximum no. of words in a room name. Default: 8

$name_maxuncap   Maximum length of an uncapitalized word in a room name. Default: 3

$name_invalid   Regexp which, if matched, signals an invalid room name. Default: `[.!?"]`

$desc_minwords   Minimum no. of matching words required to match a room description. Default: 20

$cmd_prompt   Regexp matching a command prompt. Default: `^>\s*`

$cmd_look   Regexp matching a 'look' command (case-insensitive). Default: `^l(ook)?$`

$cmd_undo   Regexp matching an UNDO command (case-insensitive). It's assumed that only a single level of UNDO is available. Default: `^undo$`

$cmd_teleport   Regexp matching commands that are known to cause a teleport to an arbitrary room (case-insensitive). Default: `^(restart|restore)$`

7.4.6 Writing the Map

Output from `scr2ifm` is as follows. First, if the title of the map was specified with the `title` command, it is printed. Then, if there's more than one map section, a list of map section titles is printed.

Next, there's a list of IFM commands derived from the transcript. Each time a new room is seen, a room command is output for it. Each room is given a tag name formed by the initials of each capitalized word in its name. To make tags unique, the second and subsequent occurrences of a tag name have that number appended. For example, room "West of House" would get tag `WH`.

If a movement command was up, down, in or out, then a compass direction is chosen for it that doesn't clash with other rooms (if possible) and the `go` attribute is used to mark its real direction. If a movement command isn't recognized, the same is done except this time the `cmd` attribute is used to record the command instead.

If movement is seen between two already-visited rooms, a link command is output (or a join, if the rooms appear to be on different map sections). Link tags are built by concatenating the room tags of the linked rooms with an underscore in between. Link movement is dealt with in the same manner as for rooms.

If the special `item` or `task` commands are seen, then an item or task command is output following the room that it appears in. If an item or task hasn't been given a tag, it is assigned one. The tag is made by translating all invalid tag characters in the name to underscores, capitalizing (to avoid clashes with IFM commands) and, if the tag clashes with a previously-defined tag, adding a numerical suffix to make it unique. For example, "black rod" might get a tag `Black_rod_2`.

Finally, the IFM commands from the file indicated by the `-c` option are echoed (if any).

7.4.7 Example Session

Here's a simple example of `scr2ifm` in action on Colossal Cave. First, the transcript in file `advent.scr` (with annotation):

```
Start of a transcript of
ADVENTURE
The Interactive Original
By Willie Crowther and Don Woods (1977)
David M. Baggett's 1993 reconstruction, ported by Graham Nelson
Release 3 / Serial number 951220 / Inform v1502 Library 5/12
Interpreter 1 Version B / Library serial number 951024

>verbose
ADVENTURE is now in its "verbose" mode, which always gives
long descriptions of locations (even if you've been there
before).

>l

At End Of Road
You are standing at the end of a road before a small brick
building. Around you is a forest. A small stream flows out
of the building and down a gully.
```

The previous two commands are recommended after starting the transcript.

```
>e

Inside Building
You are inside a building, a well house for a large spring.

There are some keys on the ground here.

There is tasty food here.

There is a shiny brass lamp nearby.

There is an empty bottle here.

>item "keys"
That's not a verb I recognise.

>item "food"
That's not a verb I recognise.

>item "lamp"
That's not a verb I recognise.

>item "bottle"
That's not a verb I recognise.
```

The previous four commands declare the given items to the program. You don't have to bother with this if you don't want item locations on your map.

```
>get all
stream: You have nothing in which to carry the water.
well house: That's hardly portable.
spring: That's hardly portable.
pair of 1 foot diameter sewer pipes: That's hardly portable.
set of keys: Taken.
```

40

```
tasty food: Taken.
brass lantern: Taken.
small bottle: Taken.

>w

At End Of Road
You are standing at the end of a road before a small brick
building. Around you is a forest. A small stream flows out
of the building and down a gully.

>s

In A Valley
You are in a valley in the forest beside a stream tumbling
along a rocky bed.

>s

At Slit In Streambed
At your feet all the water of the stream splashes into a
2-inch slit in the rock. Downstream the streambed is bare
rock.

>s

Outside Grate
You are in a 20-foot depression floored with bare dirt. Set
into the dirt is a strong steel grate mounted in concrete. A
dry streambed leads into the depression.
```

Note that in this game we don't go in all possible directions—there's no logic to the connections, and it'd just make a complete mess on the map.

```
>unlock grate
What do you want to unlock the steel grate with?

>keys
You unlock the steel grate.

>task "Unlock grate" need Keys
That's not a verb I recognise.
```

That last command declared a task—again, you don't have to bother with this if you don't want to.

```
>open grate
You open the steel grate.

>task "Open grate" after last
That's not a verb I recognise.

>d

Below the Grate
You are in a small chamber beneath a 3x3 steel grate to the
surface. A low crawl over cobbles leads inward to the west.
```

```
The grate stands open.

>w

In Cobble Crawl
You are crawling over cobbles in a low passage. There is a
dim light at the east end of the passage.

There is a small wicker cage discarded nearby.

>item "cage"
That's not a verb I recognise.

>get cage
Taken.

>w

Darkness
It is pitch dark, and you can't see a thing.

>undo
In Cobble Crawl
[Previous turn undone.]
```

Oops! Stumbled into the dark by mistake. Without the UNDO, that would have meant 'Darkness' appeared on the map as a room.

```
>turn lamp on
You switch the brass lantern on.

>w
In Debris Room
You are in a debris room filled with stuff washed in from
the surface. A low wide passage with cobbles becomes plugged
with mud and debris here, but an awkward canyon leads upward
and west.

A note on the wall says, "Magic word XYZZY."

A three foot black rod with a rusty star on one end lies
nearby.

>item "black rod" tag Rod
That's not a verb I recognise.

>xyzzy

Inside Building
You are inside a building, a well house for a large spring.
```

Goodness me! That was unexpected—we've been teleported. The link almost certainly won't look good on the map. We have the option of UNDOing, right now, or fixing the map later. Let's do the latter.

```
>xyzzy
```

```
In Debris Room
You are in a debris room filled with stuff washed in from
the surface. A low wide passage with cobbles becomes plugged
with mud and debris here, but an awkward canyon leads upward
and west.

A note on the wall says, "Magic word XYZZY."

A three foot black rod with a rusty star on one end lies
nearby.

>get rod
Taken.

>w

Sloping E/W Canyon
You are in an awkward sloping east/west canyon.

>w

Orange River Chamber
You are in a splendid chamber thirty feet high. The walls
are frozen rivers of orange stone. An awkward canyon and a
good passage exit from east and west sides of the chamber.

A cheerful little bird is sitting here singing.

>item "bird"
That's not a verb I recognise.

>unscript

End of transcript.
```

Ok, now to create a map. To do that, use this command:

```
scr2ifm -o advent.ifm advent.scr
```

To check for problems, just type:

```
ifm advent.ifm
```

On the first run through, there are complaints about an overlapping room, due to a bad choice of direction for the "down" link from Outside Grate, and the expected problems with the link between the Debris Room and the Building. A command file, advent.cmd, will fix these problems:

```
# Fix bad choice of direction.
link BG dir s;

# Hide "xyzzy" link.
link IDR_IB hidden;

# Replace it with join.
join IDR to IB cmd "xyzzy";
```

Now, if we invoke

```
scr2ifm -o advent.ifm -i -c advent.cmd advent.scr
```

we get the following map:

```
## IFM map created by scr2ifm.pl

## Commands generated by transcript.
room "At End Of Road" tag AEOR;

room "Inside Building" dir e from AEOR tag IB;
  item "keys" tag Keys;
  item "food" tag Food;
  item "lamp" tag Lamp;
  item "bottle" tag Bottle;

room "In A Valley" dir s from AEOR tag IAV;

room "At Slit In Streambed" dir s from IAV tag ASIS;

room "Outside Grate" dir s from ASIS tag OG;
  task "Unlock grate" need Keys tag Unlock_grate;
  task "Open grate" after last tag Open_grate;

room "Below the Grate" dir n from OG go down tag BG;

room "In Cobble Crawl" dir w from BG tag ICC;
  item "cage" tag Cage;

room "In Debris Room" dir w from ICC tag IDR;
  item "black rod" tag Rod;

link IDR to IB dir n s cmd "xyzzy" tag IDR_IB;

room "Sloping E/W Canyon" dir w from IDR tag SEC;

room "Orange River Chamber" dir w from SEC tag ORC;
  item "bird" tag Bird;

## Customization commands.
link BG dir s;
link IDR_IB hidden;
join IDR to IB cmd "xyzzy";
```

Printed, it looks like Figure 2.

7.4.8   Limitations

Here's a list of things that `scr2ifm` doesn't currently deal with:

- Mazes. Different maze rooms that look identical (i.e., have identical names *and* descriptions) will simply appear as a single room.

- One-way links. These aren't detected, so you'll have to add the appropriate IFM attribute yourself (in a command file, or by fixing the map directly).
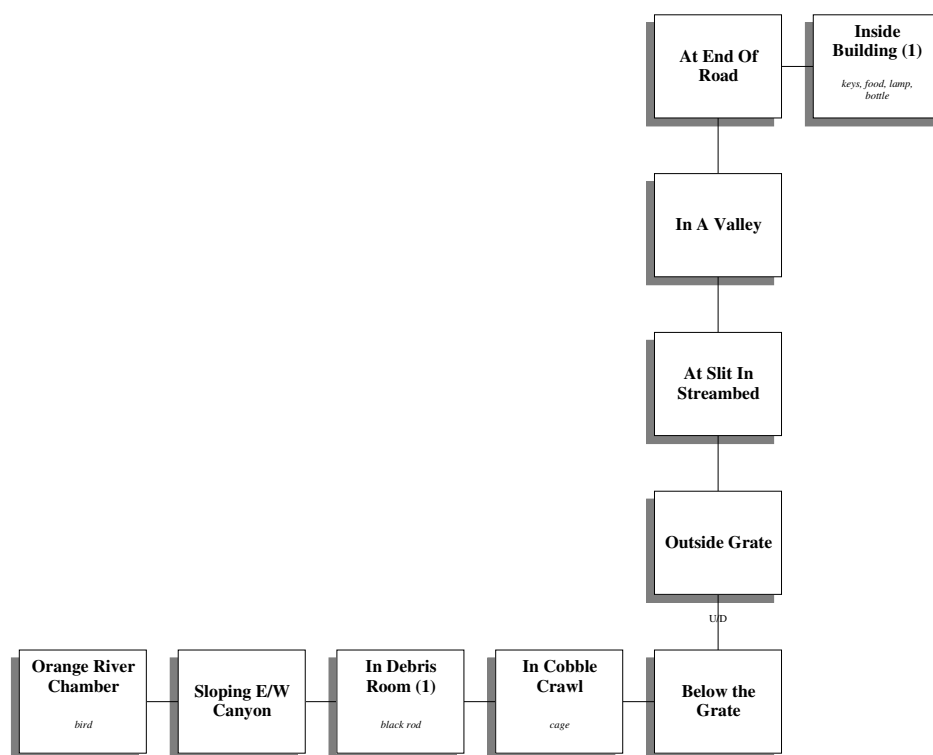
Figure 2: Map produced by scr2ifm

A GNU Free Documentation License

Version 1.1, March 2000

Copyright Â© 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you."

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque."

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the

DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

3. State on the Title page the name of the publisher of the Modified Version, as the publisher.

4. Preserve all the copyright notices of the Document.

5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

8. Include an unaltered copy of this License.

9. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

11. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

13. Delete any section entitled "Endorsements." Such a section may not be included in the Modified Version.

14. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications." You must delete all sections entitled "Endorsements."

6. Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Addendum: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright Â© [year] [your name]

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being [*list their titles*], with the Front-Cover Texts being [*list*], and with the Back-Cover Texts being [*list*]. A copy of the license is included in the section entitled "GNU Free Documentation License."

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being [*list*]"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.