IFM User Manual

Release 5.4

Glenn Hutchings

December 24, 2014

CONTENTS

CHAPTER 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 *Making maps* for details 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 *Solving the game* for details of how to do this.

A more involved example of IFM in use throughout a complete game as it's being played is given in *Mapping and solving example*.

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

IFM comes with **ABSOLUTELY NO WARRANTY**. This is free software, and you are welcome to redistribute it under certain conditions; see *License* for details.

1.2 A bit of history

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

1.3 Getting IFM

You can find IFM in the mapping tools directory of the IF archive. See the file `INSTALL` that comes with the IFM source distribution for build and installation details.

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

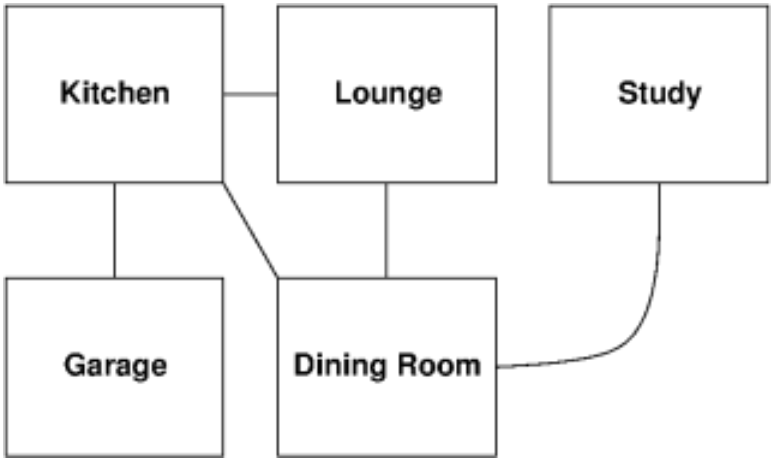
CHAPTER 2

MAKING MAPS

This section gives you a tour of the main commands for making maps. It's not complete; see *Language* 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, like this:



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.

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;
```

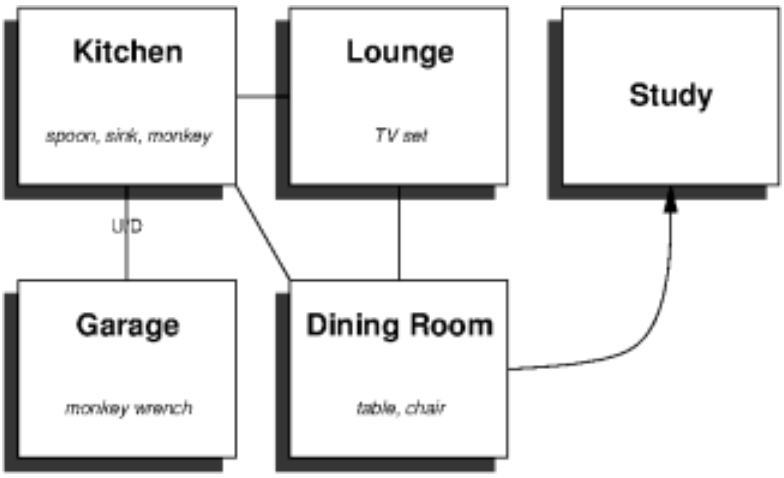
The `in` clause can be omitted, and then the room defaults to the last room mentioned. You can also add an arbitrary note to each item (e.g., to remind you what it's for) using the `note` attribute:

```
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;
```

And this is what it looks like as rendered by IFM:



2.8 Mapping problems

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.

CHAPTER 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 *Language* 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;

# other stuff

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;

# other stuff

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, with only movement commands allowed in between. 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 needs 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 Tweaking the Solution) 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;

# other stuff

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. [1]

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. [2] 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 `false`. In that case, useless items will be dropped as soon as possible.

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 *Recording commands (rec)* for details.

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;
```

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.

[1] However, you can't create an infinite loop since each task can only be done once.
[2] It has a magic crystal ball that can see into the future.

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 Inventory items, 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 [3] 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).

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

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*: 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 *Language*). 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:
   - it's needed for movement,
   - it's needed for a task that hasn't been done yet,
   - it's being kept unconditionally,
   - it's being kept with another item that's carried,
   - 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:
   - have all required items,
   - have done all required previous tasks,
   - 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,
   - it must not have previously been marked unsafe (e.g., because it closes off map connections),

- 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;

# other stuff

# 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. [4]

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 *Recording commands (rec)*.
- 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 too 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.

[4] 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.

- 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.

CHAPTER 4

MAPPING AND SOLVING EXAMPLE

Here's a more detailed example of how to use IFM, using the example game *Ruins* to produce a full IFM map containing items and game-solving tasks.

Ruins originally existed only in snippets throughout Graham Nelson's *Inform Designer's Manual* until Roger Firth produced a playable version of it. It serves as a good demo of IFM capabilities, since it doesn't give away any secrets from a "real" game. The source code for *Ruins* can be found bundled with the Inform 6 sources.

4.1 Mapping the game

OK, let's go. First, let's set the title for the map, which gets printed out (for example) at the top of each PostScript output page:

```
title "Ruins";
```

On starting the game, we see this:

```
Days of searching, days of thirsty hacking through the briars of the
forest, but at last your patience was rewarded. A discovery!

RUINS
An Interactive Worked Example
Copyright (c) 1995 by Graham Nelson.
Revisited 1998 by Roger Firth.
Release 2 / Serial number 981117 / Inform v6.31 Library 6/11 S

Dark Forest
In this tiny clearing, the pine-needle carpet is broken by stone-cut steps
leading down into the darkness. Dark olive trees crowd in on all sides, the
air steams with warm recent rain, midges hang in the air.

A speckled mushroom grows out of the sodden earth, on a long stalk.
```

Ooh, that mushroom looks interesting. Let's map out what we have so far:

```
room "Dark Forest" tag Dark_Forest;
item "speckled mushroom" tag mushroom;
```

Let's have a closer look at that mushroom:

```
>GET SPECKLED MUSHROOM
You stoop to pick the mushroom, neatly cleaving its thin stalk. As you
straighten up, you notice what looks like a snake in the grass.

You hear the distant drone of an aeroplane.

>EXAMINE SNAKE
A menacing, almost cartoon-like statuette of a pygmy spirit with an
enormous snake around its neck.

The plane is much louder, and seems to be flying very low. Terrified birds
rise in screeching flocks.

>GET STATUETTE
Taken.

You involuntarily crouch as the plane flies directly overhead, seemingly
only inches above the treetops. With a loud thump, something drops heavily
to the ground.

>LOOK

Dark Forest
In this tiny clearing, the pine-needle carpet is broken by stone-cut steps
leading down into the darkness. Dark olive trees crowd in on all sides, the
air steams with warm recent rain, midges hang in the air.

Your packing case rests here, ready for you to hold any important cultural
finds you might make, for shipping back to civilisation.

The roar dies away; calm returns to the forest.
```

We've found a statuette that was hidden until we picked the mushroom:

```
item "statuette" tag statue hidden need mushroom;
```

And a packing crate just fell out of the sky! Looks like that was going to happen regardless, so nothing to do, map-wise. Let's continue exploring:

```
>D

Square Chamber
A sunken, gloomy stone chamber, ten yards across, whose floor is covered
with fine sandy dust. A shaft of sunlight cuts in from the steps above,
giving the chamber a diffuse light, but in the shadows low lintelled
doorways to east and south lead into the deeper darkness of the Temple.

Carved inscriptions crowd the walls and ceiling.
```

We went down to get here, and have to decide on a compass direction to represent this on the map; let's choose south. This might clash with other rooms later, but we can always come back and change it if needed.

We can also see some exits to other rooms, and can indicate those too, for later reference:

```
room "Square Chamber" tag Chamber dir s go down exit s e;
item "carvings";
```

OK, let's have a bit of a look around.

```
>EXAMINE INSCRIPTIONS
The carvings cut deep into the rocks from which the chamber is built,
forming swirling sinuous patterns.

>EXAMINE MUSHROOM
The mushroom is capped with blotches, and you aren't at all sure it's not a
toadstool.

>EXAMINE SUNLIGHT
The shaft of sunlight glimmers motes of dust in the air, making it seem
almost solid.
```

Hmm... mushroom, sunlight. Sunlight, mushroom. Hey, let's do it!

```
>PUT MUSHROOM IN SUNLIGHT
You drop the mushroom on the floor, in the glare of the shaft of sunlight.
It bubbles obscenely, distends and then bursts into a hundred tiny insects
which run for the darkness in every direction, leaving strange trails in
the sandy floor. No trace of fungus remain.
```

Well, that was unexpected. Now what?

```
>EXAMINE TRAILS
The insect trails seem to resemble one particular group of carvings on the
wall.

>EXAMINE CARVINGS
You stare at a group of carvings in the middle of one wall, uncannily
resembling the insects' trail left in the sand, and feel your hand reaching
out in that direction.

>PRESS CARVINGS
The carvings slide inwards, revealing a small cavity.

>SEARCH CAVITY
In the cavity among the carvings is a huge iron key.

>GET IRON KEY
Taken.
```

Aha! A key. I predict there'll be at least one door in this game. But I hope we didn't need that mushroom any more...

Recording what we just did:

```
task "Put mushroom in sunlight" need mushroom lose it;
task "Press carvings" after last;
item "iron key" tag key hidden after last;
```

Let's explore a bit more:

```
>S

Stooped Corridor
A low, square-cut corridor, running north to south, stooping over you.
Patches of moss emit a faint green fluorescent glow, just enough that you
can see your surroundings.
The passage is barred by a massive door of yellow stone.

>EXAMINE MOSS
The moss grows in rough clumps, surprisingly thick in places. Is that a
hint of red hidden among all the greenery?

>SEARCH MOSS
Something rolls from the moss onto the ground.

>LOOK

Stooped Corridor
A low, square-cut corridor, running north to south, stooping over you.
Patches of moss emit a faint green fluorescent glow, just enough that you
can see your surroundings.

The passage is barred by a massive door of yellow stone.

You can also see a blood-red ruby here.

>GET BLOOD-RED RUBY
Taken.
```

Haha, more not-very-well-hidden goodies. And look—a door! What a surprise. Let's do some mapping:

```
room "Stooped Corridor" dir s;
item "moss";
task "Search moss";
item "blood-red ruby" tag ruby hidden after last;
```

OK, let's use that key!

```
>UNLOCK DOOR WITH KEY
You unlock the stone door.

>OPEN DOOR
You open the stone door.

>S

Lofty Shrine
For an underground chamber, the Shrine is remarkably high, though its upper
half is hazy and difficult to see clearly. The craggy walls are polished
natural rock, in which tiny flecks of quartz catch the light of the
flickering candle.

The massive yellow stone door is open.
```

```
A great stone slab of a table dominates the Shrine.

On the slab altar is a flickering candle.

Behind the slab, a mummified priest stands waiting, barely alive at best,
impossibly venerable.
```

How atmospheric. Continuing with mapping:

```
task "Unlock door with key" need key;
task "Open door" after last;

room "Lofty Shrine" dir s after last;
item "stone slab";
item "mummified priest";
```

Let's see what we have here:

```
>EXAMINE SLAB
It's big enough to lie on, though in the circumstances that might prove to
be a A Really Bad Idea.

>EXAMINE PRIEST
He is dessicated and hangs together only by will-power. Though his first
language is presumably local Mayan, you have the curious instinct that he
will understand your speech.
```

Hmm... let's press him for information, then:

```
>ASK PRIEST ABOUT RUBY
"That is one of my people's sacred relics."

>ASK PRIEST ABOUT SLAB
"The King (life! propserity! happiness!) is buried deep under this Shrine,
where you will never go."

>ASK PRIEST ABOUT KEY
"That has been lost for generations. My gratitude to anyone who returns it
will be overwhelming."
```

OK, er, let's give it back:

```
>PUT KEY ON SLAB
His eyes dimly agleam with gratitude, the priest takes the key and conceals
it among his rags. Then, lifting the candle, he carefully detaches a large
lump of the mottled wax, and gives it to you.

>EXAMINE WAX
On closer examination, the lump appears to be an old honeycomb.
```

Looks like it might be worth something. Time for a bit of mapping:

```
task "Put key on slab" need key lose it give wax;
item "wax" tag wax hidden;
```

Right, let's go back and try another direction we haven't been yet:

```
>N

Stooped Corridor

The massive yellow stone door is open.

>N

Square Chamber

Carved inscriptions crowd the walls and ceiling.

Strange trails swirl on the sandy floor.

>E
As you move into the eastern shadows, you seem to glimpse the word SENE
scratched on the lintel, but before you can stop to inspect it more
closely, you find yourself in...

Darkness
The darkness of ages presses in on you, and you feel claustrophobic.

Somewhere, tiny claws are scuttling.
```

Crikey. Don't like the sound of that.

```
>E

Darkness

Somewhere, tiny claws are scuttling.

The scuttling draws a little nearer, and your breathing grows load and
hoarse.

>S

Darkness
Somewhere, tiny claws are scuttling.

The perspiration of terror runs off your brow. The creatures are almost
here!
```

OK, this is bad. Think before acting. What did the word *SENE* mean? Hmm, maybe it's *two* words...

```
>SE

Darkness

Somewhere, tiny claws are scuttling.

>NE
Sensing a slight draught you move in that direction, stumbling over
```

```
something lying on the ground in the dark. Almost inadvertently you grab it
before gratefully emerging into the gloomy chamber.

Square Chamber

Carved inscriptions crowd the walls and ceiling.

Strange trails swirl on the sandy floor.
```

Now that was close. But what have we grabbed?

```
>I
You are carrying:
  a silver bangle
  a lump of wax
  a blood-red ruby
  a pygmy statuette
```

Ooh, a silver bangle. Plus, we didn't die!

Let's map what we just did. We moved east from the Square Chamber into the Web of Darkness. There are essentially two of those rooms, corresponding to the two right directions to move in. Let's give the Web rooms a `Dark` style to show they're dark (we can define what `Dark` means, style-wise, later):

```
room "Web of Darkness" style Dark dir e from Chamber oneway;
item "scuttling claws";

room "Web of Darkness" style Dark dir se oneway;
item "scuttling claws";
```

The act of going northeast gave us a bangle (we tripped over it and picked it up). There are two ways of doing this: with an explicit movement task that puts you back in the Square Chamber, or a "dummy" task that hands you the bangle in this room without you doing anything (using `cmd none`). Let's use the first method:

```
item "silver bangle" hidden tag bangle;
task "Stumble over bangle" give bangle cmd "NE" goto Chamber;
```

This second room is a one-way trip back to the Square Chamber. As it stands, that won't be apparent on the map. Let's add the connection in the most direct way:

```
link last to Chamber dir ne n w 2 oneway;
```

Well, there doesn't seem to be anywhere else to go. Let's get all this stuff stashed!

```
>U

Dark Forest

Your packing case rests here, ready for you to hold any important cultural
finds you might make, for shipping back to civilisation.

>PUT STATUETTE IN CASE
Safely packed away.
```

```
[Your score has just gone up by ten points.]

>PUT BANGLE IN CASE
Safely packed away.
[Your score has just gone up by twenty points.]

>PUT RUBY IN CASE
Safely packed away.

[Your score has just gone up by thirty points.]

>PUT WAX IN CASE
Safely packed away.

The case is full, and you're feeling really homesick. Now, how to get out
of here?

A deep throbbing noise can be heard.

[Your score has just gone up by forty points.]
```

OK, we've stashed all the loot we found. But what's that noise?

```
>WAIT
Time passes.

A helicopter appears, hovering overhead.

>WAIT
Time passes.

The helicopter lowers a rope; you tie it around the packing case, which is
hauled up into the air, then a rope ladder comes dangling down. With one
last look around the clearing, you climb the ladder into the waiting craft.
Back to civilisation, and a decent pint of Brakspear's!

    *** You have won ***

In that game you scored 100 out of a possible 100, in 44 turns, earning you
the rank of Master Archaeologist.
```

Huzzah! We found all the "cultural finds" and just need to get them to the museum. [1]

Let's add some tasks that record storing all the treasure. This happens in the starting room, so we'll just go back and put them just after the declaration for that room. We could just have a single task that requires all the treasure together and gets the full 100 points, but let's spell out the points for each one:

```
task "Put statuette in case" tag pack_statue need statue lose it score 10;
task "Put bangle in case" tag pack_bangle need bangle lose it score 20;
task "Put ruby in case" tag pack_ruby need ruby lose it score 30;
task "Put wax in case" tag pack_wax need wax lose it score 40;
```

[1] Yeah, right. You can if you want. I'm off to Ebay.

After the treasure is stashed, we just waited for the helicopter and won the game:

```
task "Wait for helicopter" after pack_statue pack_bangle pack_ruby pack_wax cmd "WAIT" 2 f
```

The only thing remaining to do, map-wise, is to define the display style we've used for the dark rooms. Let's just set the room background colour to be darker. Note that we could also define the `Dark` style by putting it in a file called `Dark.ifm`, and then putting that file in a directory which is searched by IFM. That way, many maps could use the same style definition. Here's the definition:

```
room_colour = "gray70" in style Dark;
```

Putting all these IFM snippets together (and indenting in my preferred style) the completed map looks like this:

```
# IFM map of 'Ruins'

title "Ruins";

room "Dark Forest"                      tag Dark_Forest;
  item "speckled mushroom"              tag mushroom;
  item "statuette"                      tag statue hidden need mushroom;

  task "Put statuette in case"          tag pack_statue need statue lose it score 10;
  task "Put bangle in case"             tag pack_bangle need bangle lose it score 20;
  task "Put ruby in case"               tag pack_ruby need ruby lose it score 30;
  task "Put wax in case"                tag pack_wax need wax lose it score 40;

  task "Wait for helicopter"            after pack_statue pack_bangle pack_ruby pack_wax
                                        cmd "WAIT" 2 finish;

room "Square Chamber"                   tag Chamber dir s go down exit s e;
  item "carvings";

  task "Put mushroom in sunlight"       need mushroom lose it;
  task "Press carvings"                 after last;

  item "iron key"                       tag key hidden after last;

room "Stooped Corridor"                 dir s;
  item "moss";

  task "Search moss";
  item "blood-red ruby"                 tag ruby hidden after last;

  task "Unlock door with key"           need key;
  task "Open door"                      after last tag open_door;

room "Lofty Shrine"                     dir s after open_door;
  item "stone slab";
  item "mummified priest";

  task "Put key on slab"                need key lose it give wax;
  item "wax"                            tag wax hidden;
```

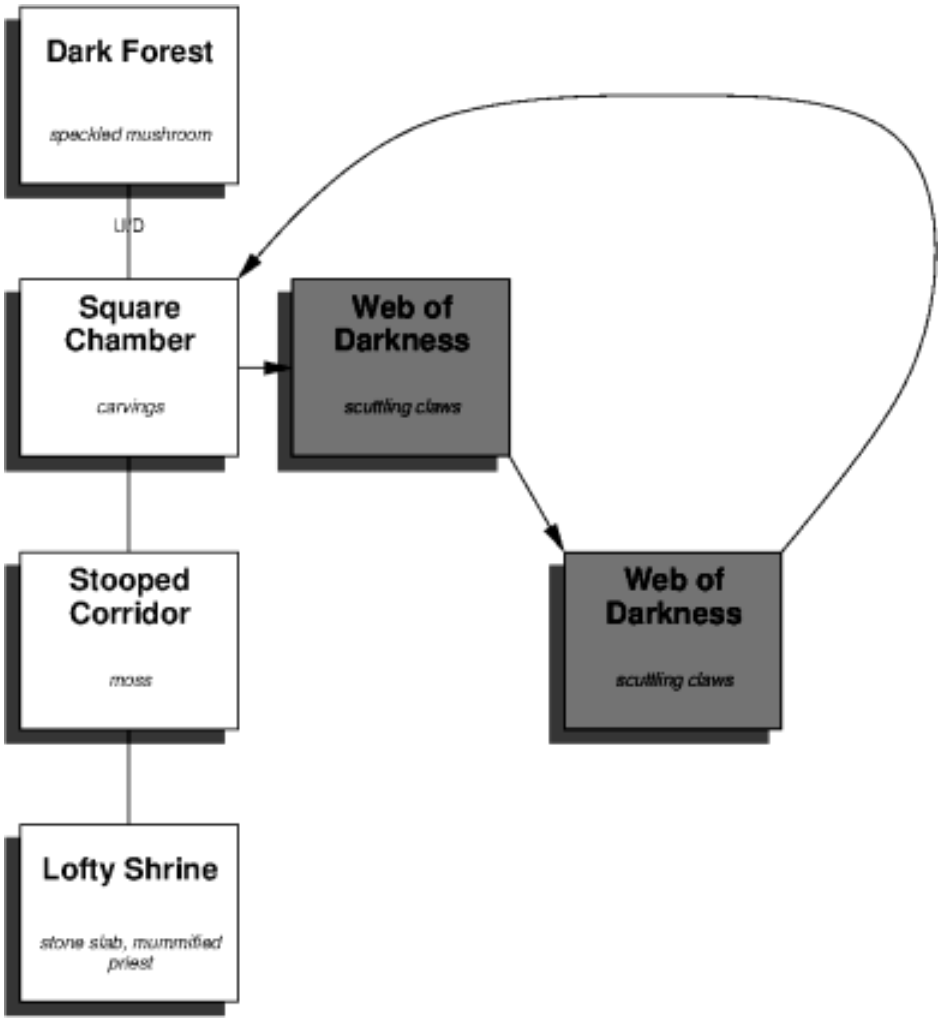
```
room "Web of Darkness"                    style Dark dir e from Chamber oneway;
  item "scuttling claws";

room "Web of Darkness"                    style Dark dir se oneway;
  item "scuttling claws";
  item "silver bangle"                    hidden tag bangle;

  task "Stumble over bangle"              give bangle cmd "NE" goto Chamber;
  link last to Chamber                    dir ne n w 2 oneway;

room_colour = "gray70" in style Dark;
```

4.2 Drawing the map

At any point during the mapping of the game, we could run IFM and get a map of where we've been (and where to explore next, since we marked room exits with `exit`). We can do that, for example, like this:

```
ifm -m -o ruins.ps ruins.ifm
```

and using our favourite PostScript viewer to view the map. Here's what the completed *Ruins* map looks like:



4.3 Listing items

We can see a complete item list using:

```
ifm -i ruins.ifm
```

Here's what gets output with the completed *Ruins* map:

```
Item list for Ruins

blood-red ruby:
   hidden in Stooped Corridor
   obtained after:
      Search moss (Stooped Corridor)
   needed for:
      Put ruby in case (Dark Forest)

carvings:
   seen in Square Chamber
```

```
iron key:
   hidden in Square Chamber
   obtained after:
      Press carvings (Square Chamber)
   needed for:
      Unlock door with key (Stooped Corridor)
      Put key on slab (Lofty Shrine)

moss:
   seen in Stooped Corridor

mummified priest:
   seen in Lofty Shrine

scuttling claws:
   seen in Web of Darkness

scuttling claws:
   seen in Web of Darkness

silver bangle:
   hidden in Web of Darkness
   obtained after:
      Stumble over bangle (Web of Darkness)
   needed for:
      Put bangle in case (Dark Forest)

speckled mushroom:
   seen in Dark Forest
   needed for:
      Get statuette (Dark Forest)
      Put mushroom in sunlight (Square Chamber)

statuette:
   hidden in Dark Forest
   obtained after:
      Get speckled mushroom (Dark Forest)
   needed for:
      Put statuette in case (Dark Forest)

stone slab:
   seen in Lofty Shrine

wax:
   hidden in Lofty Shrine
   obtained after:
      Put key on slab (Lofty Shrine)
   needed for:
      Put wax in case (Dark Forest)
```

4.4 Generating a solution

We can get a basic walkthrough of the game using:

```
ifm -t ruins.ifm
```

and it looks like this:

```
Task list for Ruins

Dark Forest:
   Get speckled mushroom
   Get statuette
   Put statuette in case
      score: 10

Move to Square Chamber (D)

Square Chamber:
   Put mushroom in sunlight
   Press carvings
   Get iron key

Move to Stooped Corridor (S)

Stooped Corridor:
   Search moss
   Unlock door with key
   Open door
   Get blood-red ruby

Move to Lofty Shrine (S)

Lofty Shrine:
   Put key on slab
      note: Gives wax

Move to Stooped Corridor (N)
Move to Square Chamber (N)
Move to Dark Forest (U)

Dark Forest:
   Put ruby in case
      score: 30
   Put wax in case
      score: 40

Move to Square Chamber (D)
Move to Web of Darkness (E)
Move to Web of Darkness (SE)

Web of Darkness:
   Stumble over bangle
      cmd: NE
```

```
      note: Gives silver bangle
      note: Moves you to Square Chamber

Move to Dark Forest (U)

Dark Forest:
   Put bangle in case
      score: 20
   Wait for helicopter
      cmd: WAIT
      cmd: WAIT
      note: Finishes the game

Total distance travelled: 10

Total score: 100
```

As you can see, IFM's solution is slightly different from the way we played it: it stashes the statue straight away. In the actual game, doing that this early wouldn't work—the packing case hasn't been dropped yet. Unless we're trying to make a recording that will play back in Frotz or similar, this doesn't really matter. If we *did* want to, we'd have to fix this and create a recording file using:

```
ifm -t -f rec ruins.ifm
```

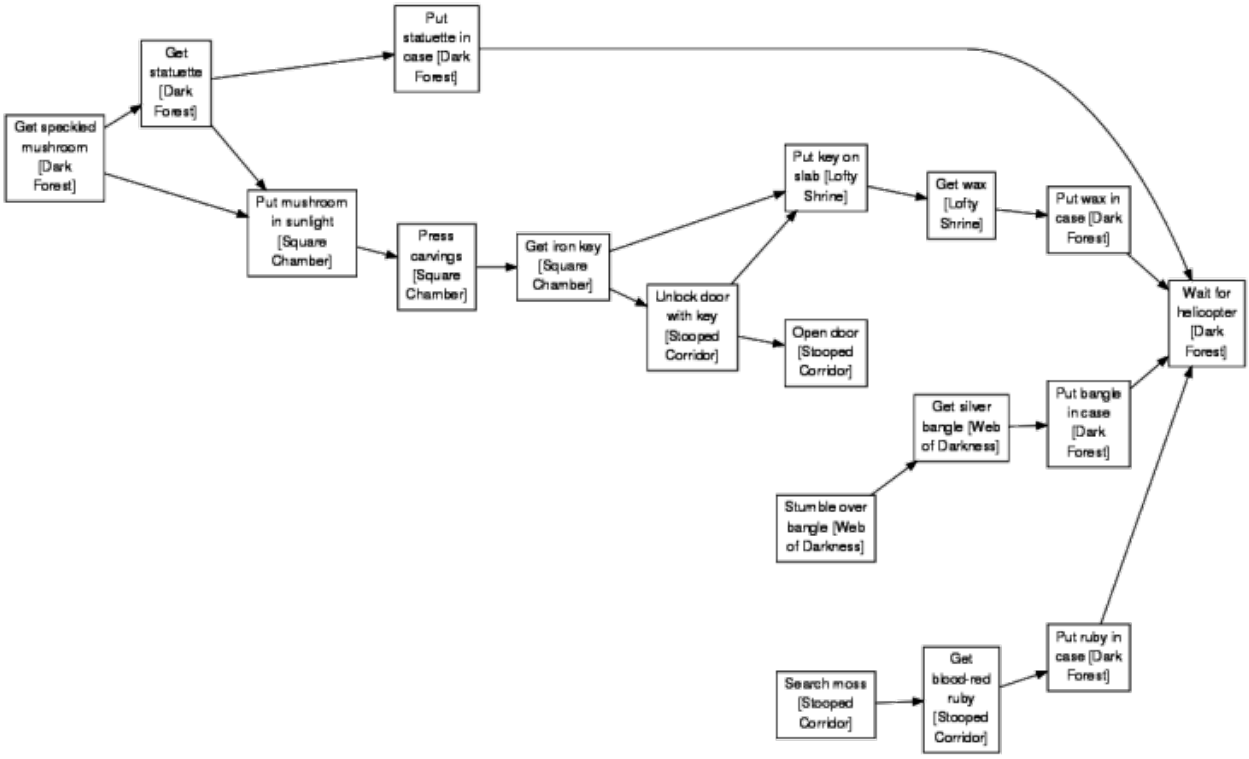
This produces:

```
GET SPECKLED MUSHROOM
GET STATUETTE
PUT STATUETTE IN CASE
D
PUT MUSHROOM IN SUNLIGHT
PRESS CARVINGS
GET IRON KEY
S
SEARCH MOSS
UNLOCK DOOR WITH KEY
OPEN DOOR
GET BLOOD-RED RUBY
S
PUT KEY ON SLAB
N
N
U
PUT RUBY IN CASE
PUT WAX IN CASE
D
E
SE
NE
U
PUT BANGLE IN CASE
WAIT
WAIT
```

Another way to view the solution is using a *task dependency graph*. This uses IFM's *dot* output format, like this:

```
ifm -t -o ruins.dot -f dot ruins.ifm
dot -Tps -o ruins-graph.ps ruins.dot
```

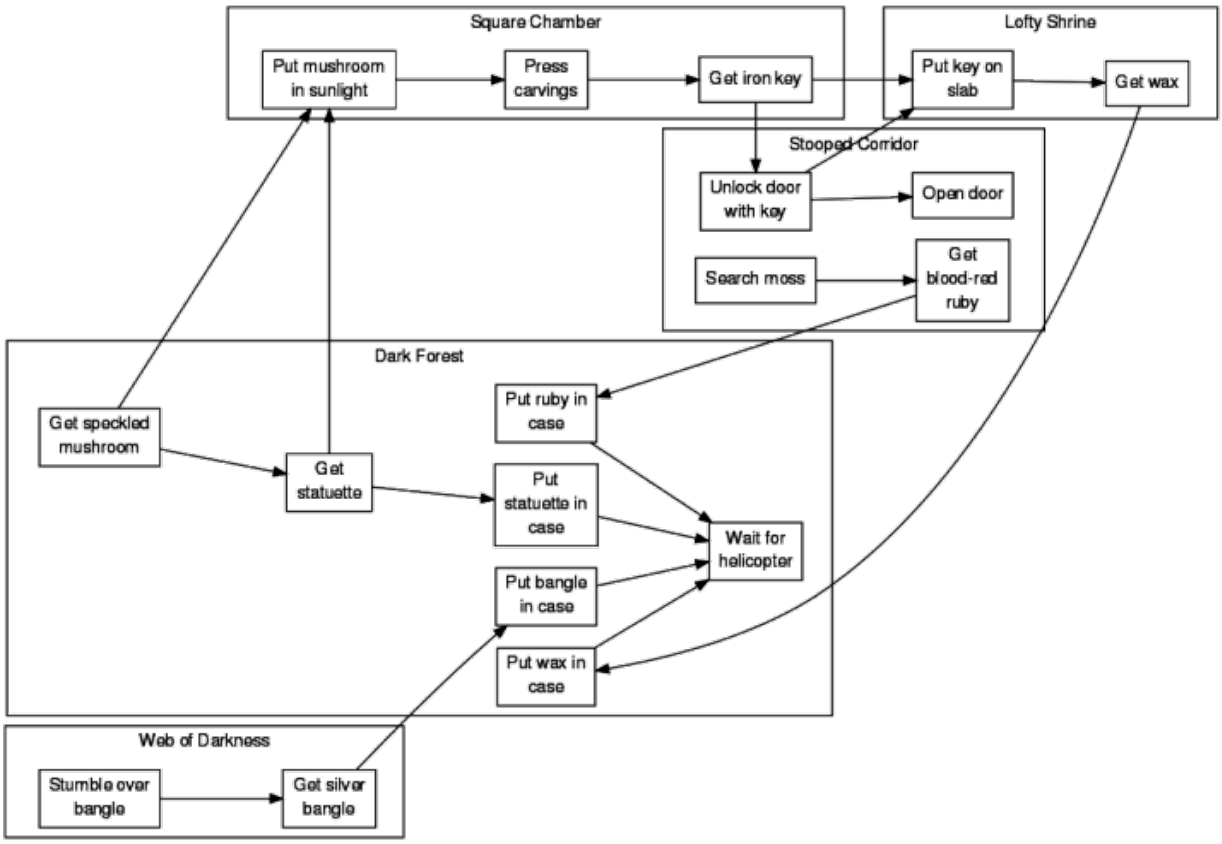
Here's what *Ruins* looks like:



Yet another way to show this is by grouping tasks together according to the rooms they occur in, by setting the `task_graph_rooms` variable:

```
ifm -t -o ruins.dot -f dot -s task_graph_rooms=true ruins.ifm
dot -Tps -o ruins-graph.ps ruins.dot
```

This is what you get:



4.5 Tweaking the solution

If we want to wait for the packing case to appear before trying to put the statue in it, we could do it like this:

```
task "Wait for packing case" in Dark_Forest need statue cmd "WAIT" 2;
```

Also, The solution as it stands is not as short as it could be: the game solver takes a trip back up to the packing case to dump two treasures and only then goes to get the bangle. This is because the stashing-treasure tasks are marginally closer at that point.

What we'd like is for the solver to grab all the underground treasure *before* returning to the surface. This can be done by extending the *length* (i.e., movement cost) of the implicit link between above-ground and below-ground:

```
link Chamber length 10;
```

After adding this line, the solver behaves the way we want.

CHAPTER 5

USING IFM

5.1 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 adjust the search path by setting the environment variable `IFMPATH`.

Then, if you have a personal initialization file, that is read. This file is found in the directory specified by your `HOME` environment variable. It can be called `.ifmrc` or `ifm.ini`; if both exist, they are both read. 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.

  New in version 5.0: Optional list of map sections.

-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 Customization for more details. This option may be repeated.

New in version 5.0.

-s, --set VAR=VALUE
Set a customization variable. This overrides any settings in the input files. This option may be repeated.

New in version 5.0.

--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.

New in version 5.2.

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.

New in version 5.0.

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 *Output variables*.

-v, --version
Print the program version.

-h, --help
Just print some usage information.

5.2 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. The table below shows what's available for each format.

Output	PostScript	Fig	Tk	Text	Rec	Dot	Raw
Map	Y	Y	Y				
Items				Y			Y
Tasks				Y	Y	Y	Y

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. [1]

5.2.1 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.

5.2.2 Fig maps (`fig`)

New in version 5.0.

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.

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.

5.2.3 Tk drawing commands (`tk`)

This produces map commands for input to *tkifm*, a simple graphical interface to IFM. 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.

5.2.4 ASCII text (`text`)

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

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

5.2.5 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.

5.2.6 Task dependencies (`dot`)

New in version 5.0.

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.

5.2.7 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. [2]

5.3 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 *Output variables*.

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:

[2] Programmer-speak for "I couldn't be bothered to document it."

```
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" style Water dir e go in;
```

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).

5.4 Predefined styles

IFM comes with a few predefined style files, as shown in the table below:

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

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. 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.

5.5 Environment variables

IFM uses the following environment variables:

IFMPATH
: A colon-separated list of directories to search for IFM files. These directories are searched before the standard directories.

HOME
: Considered to be your home directory when looking for initialization files.

5.6 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.

5.6.1 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: no items referred to by 'them' You've said `them` to refer to all the items mentioned in the current command, but no items 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.

5.6.2 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 alternative instead. Note that multiple uses of the same obsolete syntax only result in a single warning.

CHAPTER 6

LANGUAGE

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

6.1 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.

6.2 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.

6.3 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.

New in version 5.0: The `require` keyword.

6.4 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 and tags in commands that modify existing objects—these tags must be defined before being used.

6.5 Special names

There are three special names that can refer to IFM objects in certain contexts:

`last` May be used to refer to the last object of a certain type that was defined in a previous statement.

`it` May be used to refer to the most recent object in the current statement.

`them` May be used to refer to all the items mentioned so far in the current statement.

New in version 5.3.

6.6 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 has 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.

6.6.1 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.

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.

`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.

`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*.

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.

New in version 5.0.

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).

6.6.2 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.

New in version 5.0.

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

New in version 5.0.

`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).

New in version 5.0.

`given` Indicate that this item didn't have to be picked up when it entered the inventory (no "get" task should be generated).

> **Warning:** This attribute is obsolete—you should use the task `give` clause instead.

`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. 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.

6.6.3 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*.

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.

6.6.4 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*.

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.

6.6.5 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.

New in version 5.0.

`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.

New in version 5.0.

`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.

6.7 Variables

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

```
[FORMAT.]ID = NUMBER | STRING | true | false | 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.

The values `true` and `false` correspond to the integer values 1 and 0 respectively.

If the `style` clause is present, this means to only set the variable to this value in the specified style.

New in version 5.3: The `true` and `false` keywords.

6.8 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).

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.

CHAPTER 7

OUTPUT VARIABLES

There are many variables available for customizing output, described below. Their initial values are set in the IFM initialization file `ifm-init.ifm`. You can change this file to set global defaults for everybody, or alternatively set your own preferences in your personal init file.

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.

7.1 General variables

Name	Type	Default	Description	Outputs
`colour_file`	string	`ifm-rgb.txt`	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.	*ps* *fig* *tk*
`prolog_file`	string	`ifm-prolog.ps`	Prolog file that gets prepended to all the PostScript output. This defines all the procedures for drawing everything.	*ps*
`font_scale`	float	1	Scale factor which is applied to all fonts. This is a convenience variable to make font adjustments easier.	*ps* *fig*

7.2 Page variables

Name	Type	Default	Description	Outputs
page_size	string	A4	Default page size. Available page sizes are: A3, A4, A, B, C, Legal, Letter.	*ps* *fig*
page_width	float	undef	If both page_width and page_height are defined, these set a custom page size which overrides the page_size variable. Units are in cm.	*ps* *fig*
page_height	float	undef	If both page_width and page_height are defined, these set a custom page size which overrides the page_size variable. Units are in cm.	*ps* *fig*
page_margin	float	2	Margin space to be left on each page, in cm.	*ps* *fig*
page_rotate	bool	undef	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.	*ps*
show_page_title	bool	true	Whether to show the main title on each page.	*ps*
page_title_colour	string	black	Page title colour.	*ps*
page_title_font	string	Times-Bold	Page title font.	*ps*
page_title_fontsize	int	18	Page title fontsize.	*ps*
show_page_border	bool	false	Whether to show a border around each page.	*ps* *fig*
page_border_colour	string	black	Colour of the page border (if drawn).	*ps* *fig*
page_background_colour	string	white	Colour of the page background (if border is drawn).	*ps* *fig*
fit_page	bool	false	Whether to scale Fig layout to fit on a single page. Most of the time, this doesn't make things look good.	*fig*

7.3 Map section variables

Name	Type	Default	Description	Outputs
`show_map_title`	bool	true	Whether to show the map title.	*ps* *fig*
`map_title_colour`	string	`black`	Colour of the title printed above each map section.	*ps* *fig*
`map_title_font`	string	`Times-Bold`	Font of the title printed above each map section.	*ps* *fig*
`map_title_fontsize`	int	14	Font size of the title printed above each map section.	*ps* *fig*
`show_map_border`	bool	false	Whether to show a border around each map section.	*ps* *fig*
`map_border_colour`	string	`black`	Colour of the map border (if drawn).	*ps* *fig*
`map_background_colour`	string	`white` (ps, fig), `wheat` (tk)	Colour of the map background (if border is drawn).	*ps* *fig* *tk*
`map_section_spacing`	int	1	Minimum space, in rooms, between map sections when packed together.	*ps* *fig*
`map_canvas_width`	int	8	Maximum width of the Tk map canvas window, in rooms. Sizes bigger than this will cause scrollbars to appear.	*tk*
`map_canvas_height`	int	6	Maximum height of the Tk map canvas window, in rooms. Sizes bigger than this will cause scrollbars to appear.	*tk*

7.4 Room variables

Name	Type	Default	Description	Outputs
room_size	float	3	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.	*ps* *fig* *tk*
room_width	float	0.8	Proportion of the room space that's taken up by the room width. Should be less than 1 or you'll have no space left for link lines.	*ps* *fig* *tk*
room_height	float	0.65	Proportion of the room space that's taken up by the room height. Should be less than 1 or you'll have no space left for link lines.	*ps* *fig* *tk*
room_colour	string	white	Default background colour of rooms.	*ps* *fig* *tk*
room_shadow_x	float	0.05	X offset of room 'shadows', as a proportion of allocated room space. This is a pseudo-3D effect which makes rooms look raised off the page. Note that you can change the direction of the room shadow by negating one or both of the offsets.	*ps* *fig* *tk*
room_shadow_y	float	0.05	Y offset of room 'shadows', as a proportion of allocated room space. This is a pseudo-3D effect which makes rooms look raised off the page. Note that you can change the direction of the room shadow by negating one or both of the offsets.	*ps* *fig* *tk*
room_shadow	string	grey50	Colour of room 'shadows'.	*ps* *fig* *tk*
room_border_width	float	1	Width of the room box lines.	*ps* *fig* *tk*
room_border	string	black	Colour of the room box lines.	*ps* *fig* *tk*
room_border_dashed	bool	false	Whether to draw dashed room borders.	*ps* *fig*
room_exit_width	float	1	Width of the room exit lines.	*ps* *fig* *tk*
room_exit	string	black	Colour of the room exit lines.	*ps* *fig* *tk*
room_text	string	black	Colour of room description text.	*ps* *fig* *tk*
room_text_font	string	Times-Bold	Font of room description text.	*ps* *fig*
room_text_fontsize	float	10	Font size of room description text.	*ps* *fig*
room_text_font	string	Times 10 bold	Font and fontsize of room description text.	*tk*
show_items	bool	true	Whether to show non-hidden item descriptions in rooms.	*ps*

7.5 Link style variables

Name	Type	Default	Description	Outputs
`link_linewidth`	float	1	Width of link lines.	*ps* *fig* *tk*
`link_colour`	string	`black`	Colour of link lines.	*ps* *fig* *tk*
`link_arrowsize`	float	0.1	Size of oneway link arrows, as a proportion of the allocated room space.	*ps* *tk*
`link_splines`	bool	true	Whether to draw link lines as splines.	*fig* *tk*
`link_dashed`	bool	false	Whether to draw dashed link lines.	*ps* *fig*
`link_textfont`	string	`Times-Roman`	Font of text that's associated with link lines.	*ps* *fig*
`link_textsize`	int	6	Font size of text that's associated with link lines.	*ps* *fig*
`link_textcolour`	string	`black` (ps, fig), `red` (tk)	Colour of text that's associated with link lines.	*ps* *fig* *tk*
`link_textfontdef`	string	`Times 8 bold`	Font and font size size of text that's associated with link lines.	*tk*
`link_updowntext`	string	`U/D`	Text strings indicating up/down 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.	*ps* *fig* *tk*
`link_inouttext`	string	`I/O`	Text strings indicating 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.	*ps* *fig* *tk*

7.6 Join style variables

Name	Type	Default	Description	Outputs
`show_joins`	bool	true	Whether to indicate joins in the room text.	*ps* *fig* *tk*
`join_format`	string	`number`	Join string format (gets put in parentheses in those rooms that join to other rooms). The value should be `number` or `letter`.	*ps* *fig* *tk*

7.7 Game solver variables

Name	Type	Default	Description	Outputs
`keep_unused_items`	bool	true	Whether to keep unused items (i.e., those which were obtained via some task or other, but currently have no use).	*text* *rec*
`all_tasks_safe`	bool	false	Whether to treat all tasks as safe (reckless mode!).	*text* *rec*
`solver_messages`	bool	false	Whether to print game solver info messages (helps with figuring out what it's up to).	*text* *rec*
`finish_room`	string		Comma-separated list of extra room tags which are assigned the `finish` attribute. New in version 5.3.	*text* *rec*
`finish_item`	string		Comma-separated list of extra item tags which are assigned the `finish` attribute. New in version 5.3.	*text* *rec*
`finish_task`	string		Comma-separated list of extra task tags which are assigned the `finish` attribute. New in version 5.3.	*text* *rec*

7.8 Task dependency variables

Name	Type	Default	Description	Outputs
`task_graph_room`	bool	false	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.	*dot*
`task_graph_orphans`	bool	false	Whether to show orphan tasks (those with no previous/next dependencies). Useful for completeness, but it clutters things up a lot.	*dot*
`task_graph_graph`	string		Graph attributes, in Graphviz format.	*dot*
`task_graph_node`	string	`shape=box`	Node attributes, in Graphviz format.	*dot*
`task_graph_link`	string		Link attributes, in Graphviz format.	*dot*
`task_graph_wrap`	int	12	Word wrap length of nodes, in characters. New in version 5.3.	*dot*
`task_graph_font`	string	`Times-Roman`	Font name. New in version 5.3.	*dot*

CHAPTER 8

TOOLS AND UTILITIES

There are several other things bundled with IFM, which you might find useful:

8.1 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 was inspired by Frobot, another transcript-reading mapper. But **scr2ifm** in combination with IFM does a much better job.

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.

8.1.1 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.

8.1.2 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 >).
- 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.

8.1.3 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!

8.1.4 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).

8.1.5 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*`

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

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

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

8.1.6 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).

8.1.7 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.
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:

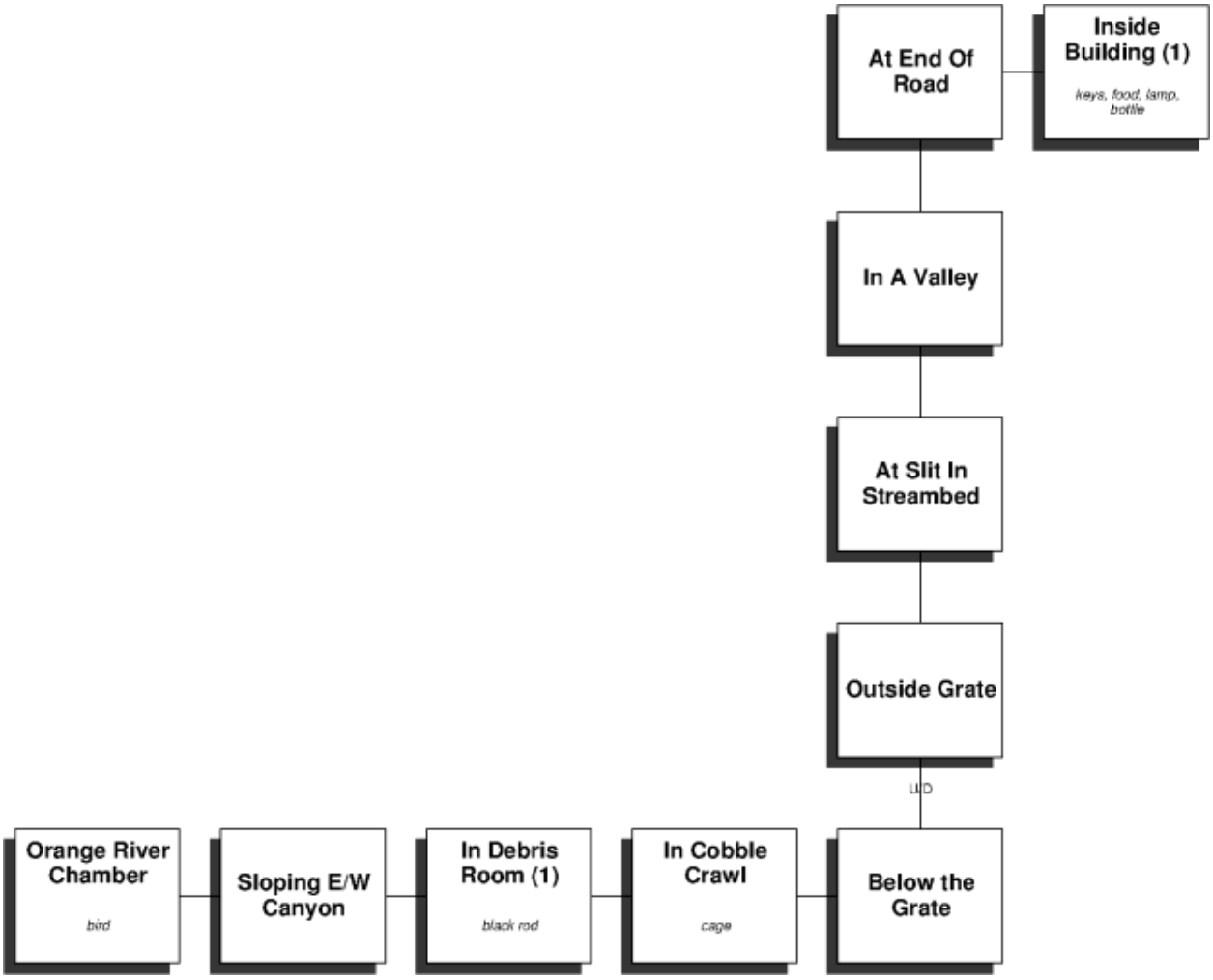
```
scr2ifm -o advent.ifm -i -c advent.cmd advent.scr
```

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 this:



8.1.8 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).

8.2 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 walkthrough 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.

8.2.1 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.

8.2.2 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
```

8.2.3 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).

8.3 ifm2dev: convert IFM maps to various other formats

New in version 5.0.

ifm2dev is a front end to **fig2dev**, which converts diagrams in Xfig 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.

8.4 ifm2tex: convert IFM maps to LaTeX

New in version 5.0.

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.

8.5 ifm2web – convert IFM map to Web image

New in version 5.3.

ifm2web converts an IFM map to an image file suitable for display on the Interweb (i.e., with transparency). It can produce two types of image:

- A map showing one or more map sections, by sending IFM output through **fig2dev** and **convert**.
- A task graph, sending IFM output through **dot** (or **neato**) and **convert**.

These are the options for printing maps:

`-m` `sections`
: Only process the map sections specified. Format is the same as the IFM `-m` option (see *Using IFM*).

`-z` `zoom`
: Set the **fig2dev** magnification (zoom) factor. Default: 1.

`-t`
: Include map section titles.

These are the options for printing task graphs:

-g
: Write task graph instead of a map.

-r
: Group the tasks by rooms.

-n
: Use **neato** instead of **dot**.

Other options:

-S `style`
: Use the specified IFM style.

-s `scale`
: Set the **convert** scale factor, as a percentage. Default: 100.

-o `file`
: Write to the specified file. If not given, the filename is built from the input file prefix and the image format.

-w
: Don't actually run anything; just print what would be done.

-h
: Print a short usage message and exit.

8.6 IFM text editor support

The `contrib` directory of the IFM source distribution has a couple of files which support two popular text editors:

`ifm-mode.el` An IFM editing mode for GNU Emacs, which has:

- Syntax highlighting
- An Index menu showing rooms, items, tasks and tags
- A command to display PostScript maps in an appropriate viewer
- Commands to show items and task lists in another window
- Support for Flymake

`ifm.vim` IFM syntax highlighting for VIM (Vi IMproved).

CHAPTER 9

IFM LINKS

9.1 Mapping programs and tools

- GUEmap is the most popular graphical mapping tool used on Windows.
- IFMapper is a graphical mapping tool for Linux and Windows and has the ability to read and write IFM and GUEmap maps. It also uses (and improves on) the *scr2ifm* algorithm for mapping transcripts.
- Frobot creates maps in a similar manner to *scr2ifm*, by analyzing a game transcript.
- ifm2i7 is a Perl script which converts IFM maps into Inform 7 input, using IFM's *raw* output format.
- asciimapper is a Perl script which takes ASCII representations of maps and converts them to IFM format.

9.2 Existing IFM maps

- There's a web page set up by Dave Chapeskie (dchapes@ddm.wox.org) which has many IFM maps in source and PostScript format, at

  https://bitbucket.org/dchapes/ifmaps

  At the moment it contains maps of a few Infocom and Inform games. Some are complete, others not quite. A few have tasks set up so that a walkthrough can be made.
- Another site containing some IFM maps can be found at:

  http://www.highprogrammer.com/alan/games/video/ifmaps

9.3 Other links

- The game-solving side of IFM touches on an area of research at McGill University called Narrative Flow Graphs.

CHAPTER 10

CREDITS

- A big thanks to Dave Chapeskie (dchapes@ddm.wox.org) for lots of great suggestions for improving IFM, setting up the original IFM *web page*, and contributing the *VIM mode* for editing IFM maps.
- Thanks to Dan Eble (eble@ticalc.org) for patches to improve PostScript output, and a bug fix.
- Thanks to Lee Bigelow (ligelowbee@yahoo.com) for enhancements to the *GNU Emacs IFM mode*.

CHAPTER 11

CHANGE HISTORY

CHAPTER 12

VERSION 5.4 (13 FEB 2009)

- Fixed build problems on various systems.

CHAPTER 13

VERSION 5.3 (14 NOV 2008)

- IFM is now hosted at Bitbucket.
- New `them` keyword, which refers to the list of items mentioned in the current statement.
- New game-solver variables: `finish_room`, `finish_item` and `finish_task`, which can be a comma-separated list of extra rooms, items and tasks which get assigned the `finish` attribute (i.e., cause the solver to exit).
- New task graph variables: `task_graph_wrap`, which controls word-wrapping when printing graph nodes, and `task_graph_font`, which sets the text font.
- The new keywords `true` and `false` can be used in place of 1 and 0, respectively, when setting variables.
- Both versions of the personal initialization file (`.ifmrc` and `ifm.ini`) are now read, if they exist.
- There's a new Perl script `ifm2web`, which converts IFM maps to image files suitable for the Interweb.
- The GNU Emacs `ifm-mode.el` in the `contrib` directory has been improved: it can display task recording and debugging output, now has `imenu` and `flymake` support, and a minor bug fix.
- The documentation is now much prettier, thanks to Sphinx and Pygments. The `contrib` directory contains the Python module used to syntax-highlight IFM code and IF transcripts.
- Since `asciimapper` is now in the IF archive, it is no longer part of the `contrib` directory.
- PostScript maps now conform properly to Adobe DSC. This stops Ghostscript 8.x barfing on them.

CHAPTER 14

VERSION 5.2 (7 SEP 2006)

- There's a new `-errors` option, which controls the maximum number of errors printed before giving up.
- New program in the `contrib` directory: `asciimapper`. This is a tool to create IFM maps from ASCII art maps, contributed by Elijah Griffin.
- The GNU Emacs `ifm-mode.el` in the `contrib` directory has new commands to display maps, items and tasks and various other things. Based on an initial version contributed by Lee Bigelow.
- Several unused features which added needless complexity have been removed. These are: preprocessing by `gpp`, expressions for real/integer values, and variable interpolation in strings.
- An obscure bug has been fixed which involved the `follow` and `need` attributes combined. This caused some valid task orderings to be flagged as cycles.
- A bug with modification of room attributes has been fixed.
- Don't crash when trying to solve game and there are no rooms.
- A crash when trying to print an error message about cyclic task dependencies has been fixed.

CHAPTER 15

VERSION 5.1 (30 NOV 2004)

- The `tkifm` program now supports line numbering and syntax highlighting, thanks to a contributor who wishes to remain anonymous.
- There is now a man page summarizing the command-line options.
- In PostScript output, up/down and in/out labels are now drawn separately, next to their respective rooms. Also, if a link leaves a room and returns to the same room, it's drawn as a small circle. Thanks to Dan Eble.
- Hopefully, the Windows port should work properly now.
- Fixed weird bug with scaled arcs in PostScript output. More thanks to Dan Eble.
- Fixed bug in `ifm2tex` with converting strings to TeX format (specifically, ones with embedded zeros).
- Fixed crash when using `it` in some circumstances.

CHAPTER 16

VERSION 5.0 (23 JAN 2003)

- New map output: Fig format, suitable for viewing or editing with Xfig, or exporting to other formats.
- New utility program: `ifm2dev`, which pipes IFM output in Fig format through `fig2dev` repeatedly to write each map section to its own file.
- New utility program: `ifm2tex`, which converts Fig files to EPS and includes them in a LaTeX document together with tables of items and tasks.
- New task output: graphviz (`dot`) format, which dumps a task dependency graph.
- New `keep with` and `keep until` syntax for items, allowing items to be kept until certain tasks are done or certain other items are dropped.
- New room `nodrop` attribute, indicating that no items should be dropped there.
- New task and item `ignore` attribute, indicating that these things should be ignored when finding a solution.
- New task `do` attribute, indicating that when a task is done, it also does one or more other tasks automatically.
- New option: `-style`, which sets global styles for all IFM objects.
- New option: `-set`, which lets you set variables from the command line. To support this, the variable-setting syntax has been changed: instead of a space separating the driver name from the variable, there's now a period.
- New `-show` option: `maps`, which lists all the map sections and their sizes.
- The `-map` option now accepts an optional list of map sections to print.
- There's a bunch of new variables to control things, including the overall behaviour of the game solver.
- IFM input is now *preprocessed*, in a similar way to C programs. As a result, the old `include` and `alias` features have been replaced with the preprocessor equivalents.
- There's a new `require` keyword, which lets you say that a certain version of IFM (or later) is required for a map.
- Changed implicit-link style behaviour: the `style` clause on a room now applies to both the room and the implicit link, so it's now independent of its position relative to the `dir` clause.
- The command-line option `-debug` has been removed; its functionality has been replaced by a control variable.

- The old `puzzle` and `special` attributes have finally been removed; if you have any old maps which use these, you must replace them with the corresponding styles instead.
- Documentation is now distributed in HTML and PDF formats, instead of GNU Info.
- The GNU Emacs IFM mode in the `contrib` directory now has font lock support. The `ifm.vim` file has also been updated with the latest keywords.
- Fixed bug with room `link` and `join` attributes: it didn't work to use `last` as an argument.
- Fixed glitch in room text on PostScript output: now tries reducing font size before squishing, so that text is filled better.

CHAPTER 17

VERSION 4.1 (10 MAY 1999)

- Fixed minor task bug: in special cases, giving items caused solver to fail to find newly-opened paths.
- Fixed stupid blunder with DOS `\r` characters in one of the source files.
- Fixed core dump problem when reading colour definitions (on FreeBSD, at least).

CHAPTER 18

VERSION 4.0 (21 APR 1999)

- PostScript driver can now print in colour.
- New 3D room shadowing effect on maps.
- Many new output variables to control new map features.
- Variables can now be set via expressions, not just values. Variables can now be used to customize many aspects of the input (e.g., notes, repeat counts).
- Individual parts of the map can now be customized using *display styles*.
- New `include` syntax, which allows files to include other files, either explicitly or via a path search.
- New `-include` (`-I`) option to prepend directories to the search path from the command line.
- New `-show` option to show various things (defined variables, search path), also included in `tkifm` menus.
- More information is now printed in item lists: which tasks require an item, which tasks an item requires, etc.
- Some example maps are now bundled with IFM, in the `demos` subdirectory.
- Names and defaults of many output variables have been changed in order to make them more consistent.
- Extra complexity in variable syntax has been removed, as it wasn't being used. Specifically, variables can't be set according to type of output or per-map-section any more, only by output driver.
- Removed formatted man page documentation: now only Info, HTML and text documentation is distributed. If there are enough complaints, I'll put it back.
- The `puzzle` and `special` attributes are now obsolete; they're replaced by display styles.
- `tkifm` now redisplays old map windows when refreshing the map.
- Win32 version is now compiled with Cygwin b20, and so requires `cygwin1.dll` (not `cygwin.dll`), included in the distribution. See `README.win32` for details.

CHAPTER 19

VERSION 3.0 (1 OCT 1998)

- There's a new program `scr2ifm` in the distribution, which attempts to convert a game transcript into IFM format. It's a perl script, and gets installed on Unix-type systems if perl is detected.
- New `nolink` attribute for rooms, which suppresses generation of an implicit link between this and the previous room.
- New `nopath` attribute for rooms, links and joins, which stops those map connections from being used by the game solver.
- New task `cmd none` syntax, for tasks that don't need any special user commands (e.g., visiting a room for the first time).
- New PostScript variable `link_arrowsize`, which allows you to change the size of arrowheads on one-way links.
- `tkifm` is now installed automatically on Unix-type systems if `wish` (or one of its namesakes) is detected.
- `tkifm` now shows busy-cursor when doing something CPU-intensive.
- Improved solver failure messages, which give more information about what's wrong.
- Removed PostScript documentation from distribution.
- The `times` keyword is now obsolete (but still works, for compatibility); just the repeat count is used instead.
- Fixed bug with `drop all except` clause; it was treating it as just a simple `drop` clause.
- Fixed bug in task path calculation; it didn't always choose the best route between rooms.
- Linking a room to itself with no `dir` clause now results in an error, instead of failing mysteriously at the map-drawing stage.
- Minor fixes to make installation smoother on FreeBSD.

CHAPTER 20

VERSION 2.1 (26 AUG 1998)

- Fixed bug which caused some types of variable not to be set properly.
- Minor fixes to library functions to link properly under Solaris.

CHAPTER 21

VERSION 2.0 (19 AUG 1998)

- Lots of new keywords for use by the game solver, which has been considerably enhanced.
- New commands to modify previous definitions, which allow you to keep all game hacks in one place.
- New `show_tags` variable, which toggles addition of room tag names to room descriptions.
- New `-debug` command-line option, which gives the gory details of what the game solver is up to.
- New recording output, which can generate commands to play back in interpreters.
- Package now has a test suite.
- Links and joins can now be `hidden`, which means they're used only for task purposes.
- Removed `groff` output; it's not worth supporting any more, now that PostScript output works so well.
- Text task output now prints details of the rooms you move through to get places, including directions moved in (or commands typed to move that way).
- PostScript driver now has some standard page sizes. Also there's a variable `page_rotate` which can override the default decisions on whether to print landscape or not.
- Having no rooms defined is not an error any more, in order to placate `tkifm`.
- Changed list attributes to be cumulative, instead of overriding previous value.
- Variables can now be set to `undef`, which allows their default setting to reappear.
- Various `tkifm` enhancements to make it convenient to use your preferred editor.
- Unix version now uses GNU autoconf/automake for installation.
- Got `tkifm` save-as option to set current filename properly.
- Stopped *outside grid* warnings from causing `tkifm` to fail when drawing map.
- Library function now doesn't clobber read-only strings.

CHAPTER 22

VERSION 1.0 (11 JUN 1998)

- First release.

CHAPTER 23

LICENSE

23.1 General Public License

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 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.

                             Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
```

rights.

  We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

  Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

```
  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
```

years, to give any third party, for a charge no more than your
cost of physically performing source distribution, a complete
machine-readable copy of the corresponding source code, to be
distributed under the terms of Sections 1 and 2 above on a medium
customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer
to distribute corresponding source code. (This alternative is
allowed only for noncommercial distribution and only if you
received the program in object code or executable form with such
an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it. For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable. However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License. Any attempt
otherwise to copy, modify, sublicense or distribute the Program 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.

5. You are not required to accept this License, since you have not
signed it. However, nothing else grants you permission to modify or
distribute the Program or its derivative works. These actions are
prohibited by law if you do not accept this License. Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions. You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

```
  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public 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.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.
```

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) 19yy <name of author>
>
> This program is free software; you can redistribute it and/or modify
> it under the terms of the GNU General Public License as published by
> the Free Software Foundation; either version 2 of the License, or
> (at your option) any later version.
>
> This program is distributed in the hope that it will be useful,
> but WITHOUT ANY WARRANTY; without even the implied warranty of

```
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) 19yy name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.
```

23.2 Free Documentation License

```
                GNU Free Documentation License
                   Version 1.1, March 2000

 Copyright (C) 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:

A. 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.
B. 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).
C. State on the Title page the name of the publisher of the
   Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications

adjacent to the other copyright notices.
F. 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.
G. Preserve in that license notice the full lists of Invariant Sections
and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. 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.
J. 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.
K. 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.
L. 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.
M. Delete any section entitled "Endorsements". Such a section
may not be included in the Modified Version.
N. 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 (c) 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.

Symbols

A

J

K

L

M

N

O

P

R

S

T

U

V