

Copyright © 1989, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

**FLIP: A GRAPHIC USER INTERFACE  
FOR MANAGEMENT AND UTILIZATION  
OF FACILITIES**

by

Alex C. West

Memorandum No. UCB/ERL M89/39

18 April 1989

COVER PAGE

**FLIP: A GRAPHIC USER INTERFACE  
FOR MANAGEMENT AND UTILIZATION  
OF FACILITIES**

by

Alex C. West

Memorandum No. UCB/ERL M89/39

18 April 1989

**ELECTRONICS RESEARCH LABORATORY**

College of Engineering  
University of California, Berkeley  
94720

TITLE PAGE

**FLIP: A GRAPHIC USER INTERFACE  
FOR MANAGEMENT AND UTILIZATION  
OF FACILITIES**

by

Alex C. West

Memorandum No. UCB/ERL M89/39

18 April 1989

**ELECTRONICS RESEARCH LABORATORY**

College of Engineering  
University of California, Berkeley  
94720

# FLIP : A Graphic User Interface for Management and Utilization of Facilities<sup>†</sup>

Alex C. West

Computer Science Division – EECS  
University of California  
Berkeley, CA 94720  
(awest@wilson.berkeley.edu)

## Abstract

A program called FLIP (Facility Layout Information Program), *version 2.7*, is first described in general, then a more detailed user's guide is presented, and finally a programmer's guide discusses some implementation details.

## Table of Contents

1. Introduction .....	3
2. System Description .....	3
2.1. What Has Been Used in FLIP's Implementation ? .....	3
2.2. Overview .....	4
2.3. Limitations, Bugs, Need for Future Work .....	5
3. User's Guide .....	7
3.1. Getting Started .....	7
3.2. Initial Configuration and Basic Functions .....	8
3.3. Where Should the Mouse be Placed ? .....	8
3.4. Menus vs. Typed Input .....	9
3.5. On-Line Help Information .....	10
3.6. The Command Menus .....	10
3.6.1. General Menu .....	10
3.6.2. Geometric Menu .....	11
3.6.3. Utility Menu .....	12
3.6.4. Equipment Menu .....	12
3.7. Coordinate Systems and Units .....	14
3.8. How to Use FLIP to Alter Information in the Database .....	14
3.9. Mode and Stacking of Operations .....	15
3.10. FLIP as an Intermediary .....	16
4. Programmer's Guide .....	16
4.1. Database Schema and Coordinate Systems .....	17
4.2. General Graphic Tools Supporting FLIP .....	18
4.2.1. The Graphic Tool xcom .....	20
4.2.2. The Graphic Tool xevents .....	20
4.2.3. The Graphic Tool xmenuenv .....	21
4.2.4. The Graphic Tool xtext .....	22
4.3. General-Purpose Modules in FLIP .....	23
4.4. Flow Control in FLIP .....	25

---

<sup>†</sup> This research was supported by Semiconductor Research Corp. and National Science Foundation under grant number MIP-8715557

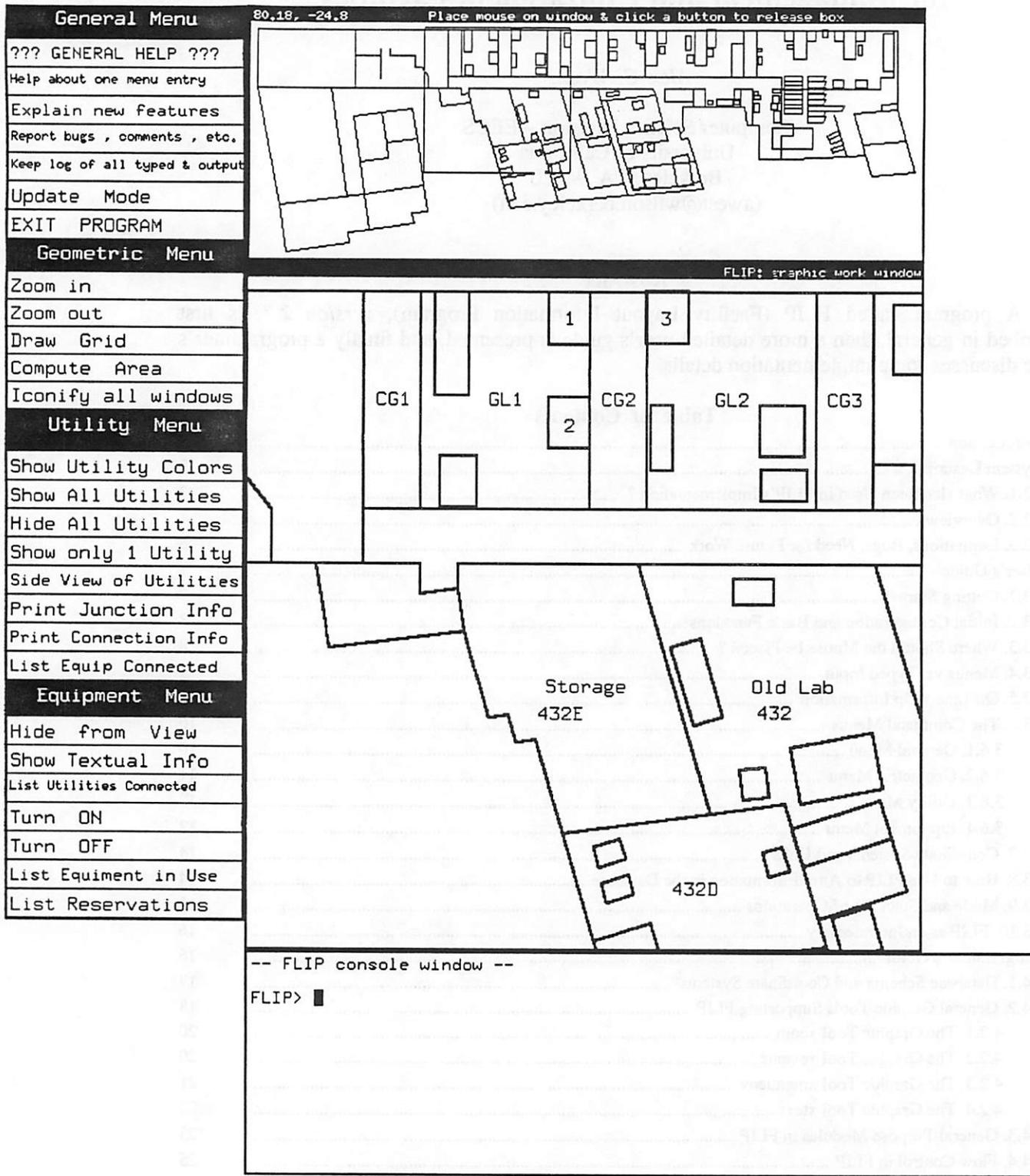


Figure 1. FLIP soon after start-up.

# 1. Introduction

A laboratory or factory contains a collection of objects (walls, partitions, doors, equipment, outlets, utility networks, etc.) of various complexity that have associated pieces of information (location, name, shape, parts, maintenance history, user instructions, etc.). Storing a suitable subset of this information in computer databases opens up new possibilities of retrieval, manipulation and display. Different computer programs may then be employed to make use of part or all of the stored information in order to support technical staff in maintaining and upgrading the complex physical facilities.

## 2. System Description

The program FLIP (Facility Layout Information Program) is a graphic user interface that permits entry, viewing and modification of a group of objects that together can be used to represent to some approximation an actual laboratory or factory floor. Each object belongs to a pre-defined set of types and contains geometric and/or textual information. In addition, other utility programs that take equipment names as input may be directly invoked from FLIP. In other words, FLIP can be used either for its own capabilities or as an intermediary between the user and some other programs.

All geometric information used by FLIP is three-dimensional, even though the user is presented a two-dimensional view most of the time. Utility networks and some other objects are color-coded. FLIP is equally menu-driven and text-driven, i.e. commands may be either selected from menus or typed in at the user's choice. Various selections also give the choice of either using the mouse or, equivalently, typing in a name or value. With few exceptions, FLIP manages to completely isolate its users from the details of the underlying database representation. Both a general help function and help information on most individual commands are available on-line.

FLIP has been installed and is in current use on SUN 3/110 color workstations in the Microfabrication Laboratory (or Microlab) at Berkeley. Early implementation development was done on a VaxStation II/GPX. All the sample FLIP images used in this paper (for example, figure 1) are based on actual Microlab data entered as of present.

### 2.1. What Has Been Used in FLIP's Implementation ?

FLIP is written in C and runs on UNIX or ULTRIX. The underlying database is Relation Technology's INGRES. Communication with INGRES takes place by means of the INGRES/EQUEL/C embedded query language, which allows database constructs interspersed with C code. Graphic operations make use of the X Window System, version 10, which is accessed by means of function calls to the Xlib library.

Several graphic modules were written with FLIP in mind; however, their function is general enough (e.g., menus) for them to be eventually replaced by better or more sophisticated ones developed outside the FLIP project. For example, various "toolkits" coming with X11 may one day be used to upgrade various parts of FLIP's graphic performance. General graphic modules and the rest of FLIP's source code amount to just over 10,000 lines of C code (including comments.)

## 2.2. Overview

Figure 2 shows the most general block diagram of FLIP's function. The emphasis here is the fact that keeping a single database helps avoid problems of duplication and inconsistency. In this context, FLIP is merely one of the many programs that can be employed to enter, alter and display information on the facility. By maintaining a modular approach, it is also easier to upgrade, modify and add new programs as the need arises. Notice that FLIP can also be used as an intermediary to invoke some other independent programs.

Figure 3 shows an approximate block diagram representation of FLIP's internal structure. Additional details are discussed in section 4, the Programmer's Guide. Shown in figure 4 is a sample situation showing a typical mode of operation in FLIP; all utility networks are displayed and a detail is zoomed in.

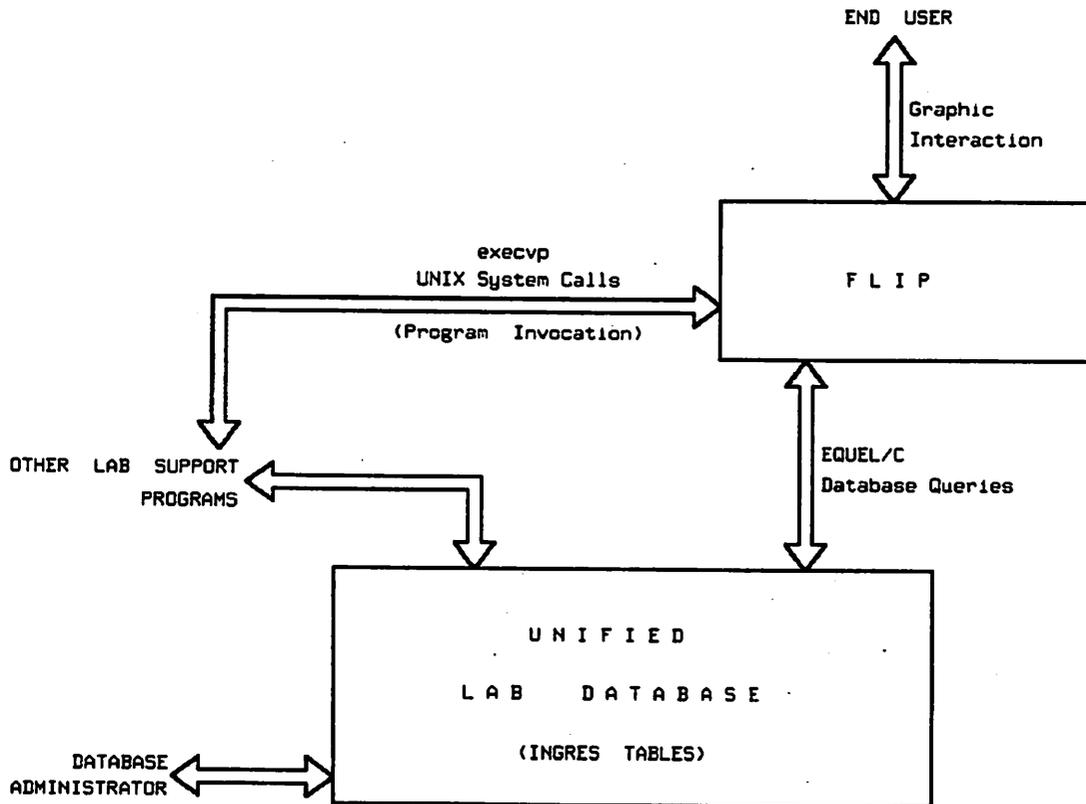


Figure 2. Highest-level block diagram.

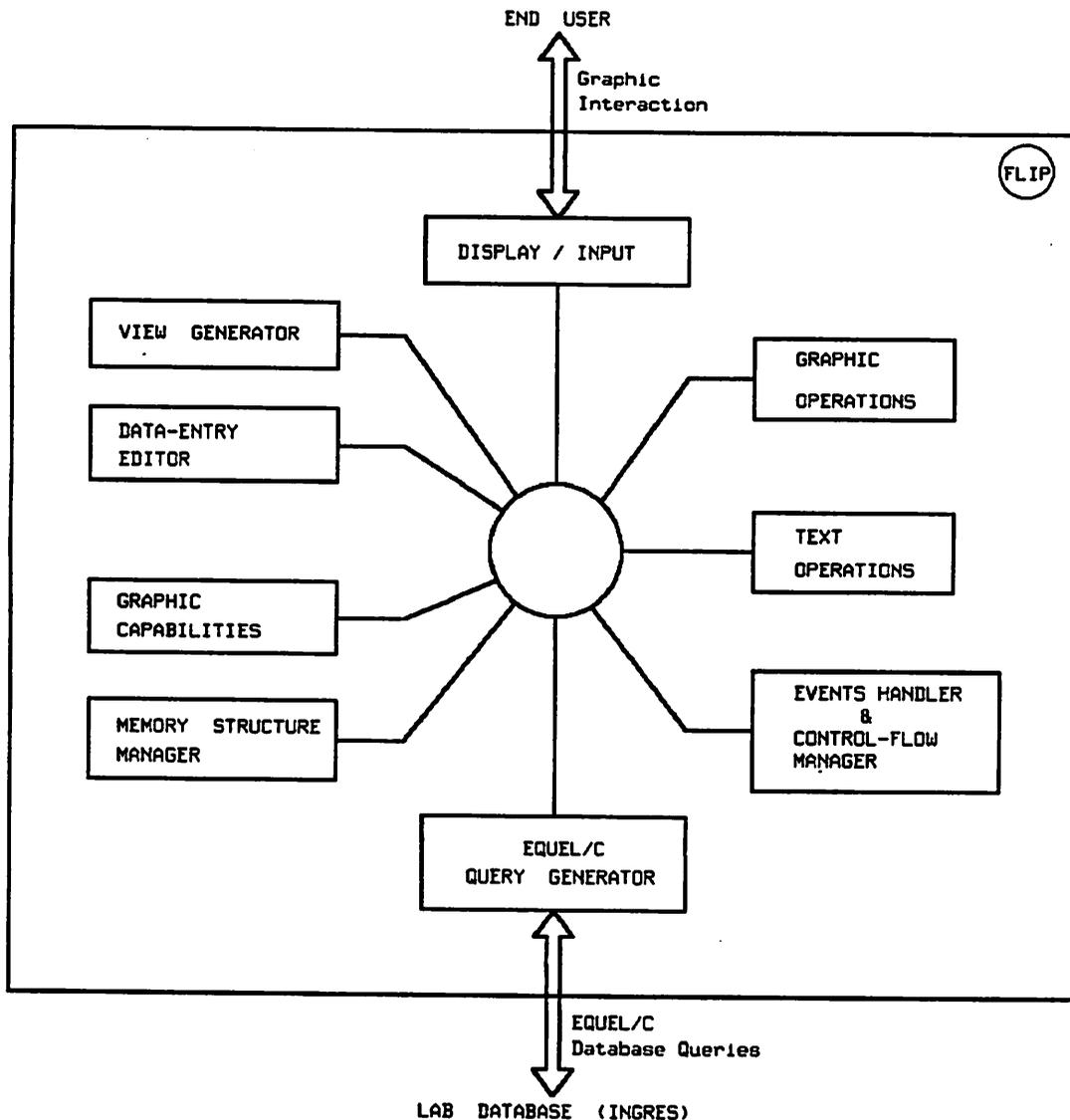


Figure 3. Block diagram of FLIP's internal structure.

### 2.3. Limitations, Bugs, Need for Future Work

- It should be possible to use FLIP to invoke a much larger variety of other programs than just those that take equipment names as input.
- A few additional functions should be added to avoid having to ever modify the database directly. (For example, FLIP currently offers no operations to store or modify labels in the database, nor operations to delete walls from the database.)
- The database schema (see section 4.1) should be revised and expanded. It would also be very desirable to have some general way of letting users describe new types of objects and operations on them.

Figure 4. Typical FLIP scene. (Approximate B/W rendition)

Utility Networks		General Menu		75.78, -6.88 Place mouse on window & click a button to release box	
Y	sprinkler	???	GENERAL HELP ???		
Y	110	Help about one menu entry			
Y	208	Explain new features			
Y	208p3	Report bugs, comments, etc.			
Y	220	Keep log of all typed & output			
Y	440p3	Update Mode			
Y	480	EXIT PROGRAM			
Y	air	Geometric Menu			
Y	gases	Zoom in			
Y	n2	Zoom out			
Y	o2	Draw Grid			
Y	chws	Compute Area			
Y	cws	Iconify all windows			
Y	d1	Utility Menu			
Y	icw	- Remove palette -			
Y	ihw	Show All Utilities			
Y	sewer	Hide All Utilities			
Y	storm	Show only 1 Utility			
Y	vdrain	Side View of Utilities			
Y	cleanvac	Print Junction Info			
Y	pumps	Print Connection Info			
	< Add New Name >	List Equip Connected			
		Equipment Menu			
		Hide from View			
		Show Textual Info			
		List Utilities Connected			
		Turn ON			
		Turn OFF			
		List Equipment in Use			
		List Reservations			
		Pumps Database			
		Gases Database			
		<pre> -- FLIP console window -- FLIP&gt; z in [Command understood as : 'Zoom in']  FLIP&gt; z in [Command understood as : 'Zoom in']  FLIP&gt; in z [Command understood as : 'Zoom in']  FLIP&gt; █ </pre>			

Update Menu
Add Wall
Extend Wall
Add Door
Add Junction
Add Connection
Extend Connection
Extract Coordinate
Add Equipment
Modify Equipment Text
Connect Equip to Utility
Delete Element
Leave Update

- Unlike some other programs using the X Window System, FLIP does not allow the user to store a personalized start-up configuration in a file.
- A tentative plan to add digitized photographs (possibly with added captions) to the type of objects that FLIP handles, was never carried out.
- No feedback to the user is given when operations are temporarily suspended and placed on a stack of stopped operations, nor when they are later resumed (See section 3.9.)
- It would be nice to make greater use of the 3-dimensionality of the information.
- Units of measure should be shown whenever a numerical quantity is presented to the user.
- FLIP expects the database to be initialized, i.e. to have all the INGRES tables set up as required. There should be an option to have FLIP do the initialization.
- The need is felt for a "Kill Operation" function that could abort any update operation while in the middle of it. Currently, the operation needs to be carried to the end and then denied confirmation (which FLIP always prompts for.)
- Adding the capability of showing a "You are here" mark on the layout view, has been suggested by some people.
- The choice between "one-shot" and "repeated" modes of operation (see section 3.9) is a difficult one for some operations and it is not clear whether the choices made by FLIP are the best ones.
- It would be convenient if screen redrawing could be sped up (maybe by setting up some hash table for the data in memory.)
- Data inconsistency may be produced if many users run FLIP concurrently and perform update operations (see section 4.1.)

### 3. User's Guide

The previous sections provide a general idea of what FLIP is used for, of what it looks like and of what parts it is composed of. We will now turn our attention to some of the most important details of actually using FLIP.

#### 3.1. Getting Started

Like many other graphic programs using the X Window system, FLIP expects the shell environment variable `DISPLAY` to be correctly set to the name of the graphic server that you want to send the picture to. The reason for this need is that X is a *network* system, i.e. you might run your program on one computer and have the display appear elsewhere. The shell variable `DISPLAY` can be set by issuing the command

```
setenv DISPLAY computer:number
```

where *computer* is the name of the computer that the graphic monitor is connected to, and *number* specifies which of the graphic monitors to use, in case there is more than one. (If there is only one, use `:0`) Many people set the variable `DISPLAY` in their ".login" file.

FLIP is started by simply typing `flip`. It is possible to provide one or both of the following two arguments :

- v            Verbose mode. Some details of the database operations are printed out, together with some other information.

**-d *dbase\_name*** Use a different database than the default one; *dbase\_name* must be the name of a valid INGRES database, with all the necessary tables already set up.

During start-up, a small window may appear, and soon disappear, in the upper left-hand corner of the screen. The small window is the *xtext* window, whose function will be explained in section 3.2. The somewhat mysterious occasional appearance and prompt disappearance is perfectly normal and has to do with the way the graphic tool *xtext* works, explained in section 4.2.4.

### 3.2. Initial Configuration and Basic Functions

When FLIP is first started, it has three windows and four command menus. The three windows are the *Global window*, with purple border, the *Detail window*, with green border, and the *Console window*, a light blue color.

Inside the Global window, FLIP generates an overall view of the facility as seen from above, i.e. its parallel projection onto the floor. Black lines represent walls and partitions; grey line represent the rough outlines (*bounding boxes*) of pieces of equipment. Notice that moving the mouse above this window "drags along" a little black box, called *magnifying glass*. A magnified view of whatever is under it appears in the Detail window, which is where all graphic selections and operations are performed (hence its header of "FLIP: graphic work window".) The magnifying glass can be "frozen" in position in the Global window by clicking any of the buttons on the mouse. A second click will restore the magnifying glass' mobility. The window header provides a constant reminder of the status of this feature, as well as a position feedback in the coordinate system and units used for the facility.

The Console window is where FLIP prints out prompts, instructions and feedback for the user. Users' typed input appears in it, too, just like on a regular terminal. A second text window, called *xtext window*, pops up as needed. (See figure 5) Its function is to allow display and editing of information in textual form. Notice that the functions of the two windows, even though both involve text, are kept separate so that the basic monitoring of what is going on in FLIP, done through the Console window, may still go on unencumbered while files are being viewed or edited in the *xtext* window.

The various operations that FLIP can do are grouped by category in various menus. Each menu has a distinguishing name that indicates the general category of operations that its entries select. Other menus that are specific to a given operation pop up as needed while the operation is being carried out. The commands available through the menus will be discussed in detail in section 3.6.

### 3.3. Where Should the Mouse be Placed ?

Generally speaking, place the mouse on the window or menu that you are selecting from or operating on. However, it is *not* necessary to place it on the Console window in order to type on it; you can type into the Console window with the mouse on any of FLIP's windows, including the menus, but with the *exception* of the *xtext* window, which is the only other window besides the Console that expects typed input. This method allows the user to specify which of the two windows will get the typed input and also reduces the amount of mouse motion necessary to get work done.

It is important to realize that, even though both handle textual information, the Console window and the *xtext* window not only have different roles but are also implemented in a totally

```
xtext
--- Invoking 'more' ---
[To advance slowly, keep pressing RETURN with mouse *inside* this window]

1. How to view this help information

    The text shown in this window is filtered through the "more" program.
    If unfamiliar with it, please type h after placing the mouse inside this
    window. In order to give any other "more" commands to scan through this
    text, the mouse must be inside this text window.

2. General overview

    FLIP (Facility Layout Information Program) is a 3-dimensional user-
    interface for display and manipulation of graphic and textual information
    about the layout of the Microlab and its utility networks and equipment.
    Some pre-existing programs that act on equipment can also be invoked through
    FLIP (look for them in the Equipment Menu.)

3. Configuration and basic functions

    When FLIP is first started it has three windows and four menus. The
    three windows are the Global Window, with purple border, the Detail Window,
    with green border, and the Console Window, of a light blue color.

    Inside the Global Window, FLIP generates an overall view of the lab as
    seen from above. Black lines represent walls and partitions; grey line
    --More--(24%)
```

Figure 5. The xtext window presenting general help information

different way. In some ways, the xtext window acts as a more separate entity from FLIP than the Console window does. This fact is due both to design choice and to implementation details (sections 4.2.1 and 4.2.4.)

### 3.4. Menus vs. Typed Input

If typed input is expected by a routine (i.e., if you are prompted for it in the Console window), then anything typed on any FLIP window other than xtext (as explained in section 3.3) will be passed to the routine that requested it; otherwise, any typed input that you provide unsolicited by FLIP will be matched against the entries of the menus currently displayed and if any entry matches it, the corresponding operation will be executed. So, to select an entry from a menu, you have the choice of placing the mouse on the desired selection and pressing one of its buttons, or of typing its name instead.

Names need not be typed accurately or in full. In fact, the input-recognizing routine is insensitive to case and to additional blanks in between words. Furthermore, it is tolerant of omission of words, extraneous words, change of word order, and it will match individual words even if one is a leading substring of the other or if they differ by a "single misspelling" (omission or addition of a single letter, substitution of a single letter, or inversion of two consecutive letters.)

### 3.5. On-Line Help Information

To find out what major changes have taken place in the last version or two, select the "Explain new features" entry.

To obtain help on a specific menu entry, select the entry "Help about a menu entry"; once you do that, the next one (*only one*) menu entry you select will not perform its ordinary function but instead it will give you help information about it. (This special mode lasts for only one selection and then resets automatically.)

### 3.6. The Command Menus

The following subsections list the entries and the meaning of the four main command menus. The information is also available on-line. Other menus pop up as needed (e.g., to confirm or abort an operation just performed.) Of particular interest among them is the "Update Menu", described in section 3.8. Notice that, to a good extent, FLIP tries to use the labels of the menu entries to provide a reminder of the state of the various operations.

#### 3.6.1. General Menu

##### ??? GENERAL HELP ???

This entry will give you a general overview of FLIP and will explain how to use it. The textual information presented to you is filtered through a program called "more".

##### *Help about one menu entry*

When this entry is selected, the next one (*ONLY one*) menu entry you select will not perform its ordinary function but instead it will show you a verbal description of its effect -- as it is taking place right now. From then on, all menu entries will automatically resume their usual meaning.

##### *Explain new features*

A brief description of some of the major recent changes in FLIP will be shown. Whenever you haven't used FLIP in a while, check this entry to see if there is anything new that might be of interest to you.

##### *Report bugs , comments , etc.*

In case that you...

- (1) encounter a bug, or something that looks like a bug to you
- (2) have a suggestion on how to do something better
- (3) would like to see new features added
- (4) have a question that the HELP functions can't answer
- (5) <etc.>

...then you can use this entry to have the mail program automatically invoked with the right address. Type in your message as you normally do with the mail program (e.g., entering a period . at the beginning of a line terminates your message and sends the mail.)

##### *Keep log of all typed & output*

This entry is used to start and terminate the process of keeping a log of all text that appears on the Console window. When selected, it will prompt the user to enter the name of a file where all text is to be copied into. Then, it will toggle its name to "Stop keeping

log file" and re-selecting it will turn the log off.

#### *Update Mode*

This entry, which will be included in your menu only if your login name was found to belong to the "labstaff" group, will enable you to pop-up a new menu that will give you access to a number of update operations that will modify the database used by FLIP.

#### *Exit Program*

This is the entry to select when you are ready to leave FLIP. You will be prompted by a pop-up menu to confirm your decision to exit.

### **3.6.2. Geometric Menu**

#### *Zoom in*

Each time that a mouse button is pressed on this entry, the size of the "magnifying glass" that follows the mouse on the global view of the lab is decreased by a small fixed amount, producing the result of zooming in on the detailed view.

#### *Zoom out*

Each time that a mouse button is pressed on this entry, the size of the "magnifying glass" that follows the mouse on the global view of the lab is increased by a small fixed amount, producing the result of zooming out on the detailed view.

#### *Draw Grid*

This entry allows you to place a grid on the detailed view; you are prompted to specify a mesh size (expressed in database units) and a point (wall corner or utility junction) to align the grid with. After the selection, this entry toggles its name to "Remove <size> grid", where <size> is the mesh size. Re-selecting it will remove the grid.

#### *Compute Area*

This entry allows you to find the area of an arbitrary polygonal region defined on the top-view projection of the facility. The polygonal region may be concave or convex and may have any number of vertices. Each vertex may be either an arbitrary point or a wall corner. Vertices must be specified in order, either clockwise or counterclockwise. To specify a vertex, use the mouse as follows:

(A) If the vertex is an arbitrary point, place the mouse exactly where you want it to be and click the MIDDLE mouse button.

(B) If the vertex is a wall corner, place the mouse close enough to it for a little box to appear, then click the RIGHT mouse button. (Notice: pressing the RIGHT mouse button will always select whatever wall corner is highlighted with a box, no matter where the mouse is.)

After you have selected all the vertices as instructed above, press the LEFT mouse button. There is absolutely no need to re-select the first vertex; the polygon will automatically be closed by joining the first and last vertices as soon as the LEFT button is pressed.

#### *Iconify all windows*

Select this entry to iconify at once ALL of FLIP's windows, including menus. An icon with the name "Entire FLIP" appears in the upper left-hand corner of the screen. To de-iconify this icon, just click any of the mouse buttons on it or, alternatively, use your window manager.

### 3.6.3. Utility Menu

#### *Show Utility Colors*

When selected, this entry will draw a palette of symbolic utility colors, each with its actual name next to it, and then will change its name to "- Remove palette -", with the obvious meaning.

#### *Show All Utilities*

Selecting this entry will cause all utility networks to be drawn, superposed to walls, partitions and pieces of equipment.

#### *Hide All Utilities*

Selecting this entry will hide from view all utility networks.

#### *Show only 1 utility*

Selecting this entry will pop-up a new menu that will allow you to choose *\*one\** utility network to be drawn, superposed to walls, partitions and pieces of equipment. This selection will override any previous one made as to what utilities will be drawn.

#### *Side View of Utilities*

This entry will give you the opportunity to view the parallel projections of all the utility networks on the back and right-hand side of a box (extending from floor to ceiling), which you can specify by clicking the mouse to define its corners. (See figure 6.) When done with this type of operation, click back on the same entry, which in the meantime will have changed its name to "-Terminate side view-"

#### *Print Junction Info*

This selection will allow you to point to the junction of a utility network that is currently displayed in the magnified view window and obtain some information about it. (Its kind, height and utility that it belongs to.)

#### *Print Connection Info*

This selection will allow you to point to two endjunctions of a utility network connection that is currently displayed in the magnified view window and obtain some information about it. (Its material name and size.)

#### *List Equip Connected*

This entry allows you to see a list of names of pieces of equipment that are connected to a given utility network, which you are asked to specify through a menu. The converse operation is available through the entry "List Utilities Connected", found in the Equipment Menu.

### 3.6.4. Equipment Menu<sup>1</sup>

#### *Hide from View*

The outlines of the pieces of equipment are normally displayed in light grey. If you don't want them to appear in the magnified view (perhaps to reduce clutter or speed up the drawing), select this entry. Once selected, it will toggle its name and function to "Show Equipment"

#### *Show Textual Info*

Invoking this entry will allow you to select a piece of equipment by either pointing to it with the mouse or typing in its name. Once the selection is successfully made, textual information about the equipment will be displayed.

---

<sup>1</sup> The last several entries in the Equipment Menu appearing in Figures 1, 4 and 6, are not part of FLIP – See section 3.10

Utility Networks		General Menu		FLIP: graphic work window	
Y	sprinkler	???	GENERAL HELP ???		
	110		Help about one menu entry		
	208		Explain new features		
	208p3		Report bugs , comments , etc.		
	220		Keep log of all typed & output		
	440p3		Update Mode		
	480		EXIT PROGRAM		
	air	<b>Geometric Menu</b>			
	gases		Zoom in		
	n2		Zoom out		
	o2		Remove 2,30 Grid		
	chws		Compute Area		
	cws		Iconify all windows		
	dl	<b>Utility Menu</b>			
	icw		- Remove palette -		
	ihw		Show All Utilities		
	sewer		Hide All Utilities		
	storm		Show only 1 Utility		
	vdrain		- Terminate side view -		
	cleanvac		Print Junction Info		
	pumps		Print Connection Info		
	< Add New Name >		List Equip Connected		
		<b>Equipment Menu</b>			
			Hide from View		
			Show Textual Info	FLIP>	
			List Utilities Connected	FLIP> grid [Command understood as : 'Draw Grid']	
			Turn ON	FLIP>	
			Turn OFF	*** GRID CONSTRUCTION ***	
			List Equipment in Use	(STEP 1) Enter size of mesh (in database units) : 2,3 [Size is : 2.300]	
			List Reservations	(STEP 2) Click mouse on vertex or junction to align grid with... [Point coords : 80,500, 0,000]	
			Pumps Database	1	
			Gases Database	[Command understood as : 'Show only 1 Utility']	
				FLIP>	
				-> Click name on palette menu or type it in...	
				sprinkler	
				[Command understood as : 'sprinkler ']	
				FLIP> view side	
				[Command understood as : 'Side View of Utilities']	
				FLIP> ->Click mouse button on desired position...█	

Figure 6. Grid and side view of utilities

### *List Utilities Connected*

This entry allows you to see a list of names of utilities that a given piece of equipment is connected to. You are asked to select the equipment by pointing to it or by typing in its name. The converse operation is available through the entry "List Equip Connected" found in the Utility Menu.

### **3.7. Coordinate System and Units**

All objects share a single, right-handed, Cartesian coordinate system. The unit of measure is arbitrary but it must be consistently used for all coordinates. At the moment the first two points are entered through FLIP, a choice is made as to what coordinate system and unit of measurement is being used. Afterwards, the only way to change coordinate system or unit is by means of direct intervention on the database. The headers of the Global and Detail windows provide a constant feedback for the position of the mouse or of the "frozen" *magnifying glass*, expressed in the coordinate system and units used for the facility.

A recent addition to FLIP's database schema (described in section 4.1) allows the storage of a character string containing the name of the unit (e.g., "meters" .) No operation is currently provided in FLIP to store or modify such name, and it must be done directly through the database. Furthermore, only one of the most recently operations added to FLIP, namely "Compute Area", will show the unit name after the numerical values it computes.

### **3.8. How to Use FLIP to Alter Information in the Database**

FLIP has an *update mode* that provides a graphic editor and other operations to allow qualified users to make changes in the database that FLIP reads from. To be a "qualified" user your login name must belong to the group "labstaff". You can find out what groups you belong to by typing groups in the UNIX shell (i.e., outside of FLIP.) When FLIP is started from the computer account of a user who has update privileges, an additional entry called "Update Mode" will automatically be included in the "General Menu." Invoking such an entry will bring up the menu shown in figure 8, called the "Update Menu". Notice that no special qualification is required to use FLIP in the viewing mode. Update operations are restricted as a form of protection for the data in the database. A detailed description of the various object types that can be modified with the graphic editor is found in section 4.1.

The Update Menu is the starting point for all update operations, and other menus pop up as needed. No detailed entry-by-entry explanation for the update menus is currently available, but very detailed directions are printed out in the Console window as prompts during the execution of update operations. A summary of the meaning of the various entries is as follows :

Add Wall	<i>Define position of new wall</i>
Extend Wall	<i>Prolong an existing wall along a chosen direction</i>
Add Door	<i>Define position on an existing wall of new door</i>
Add Junction	<i>Define position and utility type of a new junction in utility network</i>
Add Connection	<i>Define connection and material type between existing junctions</i>
Extend Connection	<i>Prolong an existing utility connection along a chosen direction</i>
Extract Coordinate	<i>Display information available about a wall/door endpoint or a junction</i>

---

Update Menu
Add Wall
Extend Wall
Add Door
Add Junction
Add Connection
Extend Connection
Extract Coordinate
Add Equipment
Modify Equipment Text
Connect Equip to Utility
Delete Element
Leave Update

Figure 8. Update Menu

---

Add Equipment	<i>Define location, dimensions and network connections of a piece of equipment that must already exist in the "process" table</i>
Modify Equipment Text	<i>Edit the text file associated to a piece of equipment</i>
Connect Equip to Utility	<i>Specify network connections to a piece of equipment</i>
Delete Element	<i>Delete junctions, connections and equipment</i>
Leave Update	<i>Hide Update Menu</i>

### 3.9. Mode and Stacking of Operations

Many viewing operations may be defined to have either of two modes : *one-shot* or *repeated*. In the former case, the operation terminates as soon as the user completes the specification (e.g. by selecting something with the mouse); in the latter case, the operation remains active until the user explicitly turns it off (e.g. by pressing the RETURN key.) In some cases, the nature of the operation makes one of the two types clearly preferable, but in many cases there seems to be no clear answer. In a few instances, the mode of a particular operation was changed in earlier versions of FLIP to accommodate users' preferences.

If you are in the middle of any operation in FLIP, you may initiate a new one. The old operation will be stacked and will automatically resume when the new one is finished. The only limitation is that no new update operation may be started while in the middle of another update, to avoid the inherent problems of concurrent updates (this provision is enforced by the fact that the update menu disappears in the middle of update operations.) There is no limit to the number of incomplete operations that may be stacked but, unfortunately, no new prompt is given when

the execution of the old operation is resumed, so it is important to keep mental track of where you are.

### 3.10. FLIP as an Intermediary

It was mentioned earlier that FLIP, in addition to its own built-in functions, has the capability of acting as an intermediary between the user and other programs that take equipment names as input. This is done as follows. A file that on our system is named /usr12/lab/micro/labroot/lib/flip/flip.ext needs to be created with the following format :

```
Name1
Program1
Name2
Program2
...
```

The above file will be read at start-up and the strings Name1, Name2, etc., will be automatically added at the end of the Equipment Menu as new entries.<sup>2</sup> Whenever one of these new entries is selected, the user will be prompted to select a piece of equipment. At that point, the corresponding program will be executed as if directly typed to a terminal, with all occurrences, if any, of the symbol @ replaced with the name of the equipment. All I/O that the program may require will be done through the *xtext* window. A sample flip.ext file, using two imaginary programs *equip* and *stat\_rep*, could be :

```
Enable Machine
equip -on @
Disable Machine
equip -off @
Report Status
stat_rep -v @
```

## 4. Programmer's Guide

The following is a list of the files that include most of the code for FLIP. Each file name is followed by a brief description of what the file contains. The rest of FLIP consists of general-purpose graphic modules described in section 4.2. All the descriptions in this guide are somewhat sketchy, and for the fine details the reader is referred to the actual source code<sup>3</sup>.

flip.h	<i>Definitions File</i>
main.c	<i>Main Program and Some General Routines</i>
init.c	<i>Routines for Initialization</i>
color.c	<i>Routines for Color Allocation and Color Mapping</i>
dbm.qc	<i>Interface to Ingres (with embedded INGRES/EQUELIC code)</i>
memory.c	<i>Routines for Operations on Data in Memory</i>

---

<sup>2</sup> This is the case of the last few entries appearing in the Equipment Menu in Figures 1, 4, and 6

<sup>3</sup> Available from the author or from the Microfabrication Laboratory, Cory Hall, UC Berkeley. (flip@argon.berkeley.edu)

handler.c	<i>Routines for Handling XEvents, Input Events from xcom Window and FLIP-Specific Events</i>
geom.c	<i>Routines for Geometric Operations</i>
gfx.c	<i>Routines for Several Graphic Operations</i>
string.c	<i>Routines to Manipulate Strings</i>
misc.c	<i>Miscellaneous Routines</i>
menu.c	<i>Miscellaneous Menu Routines</i>
menu1.c	<i>Routines for "General Menu"</i>
menu2.c	<i>Routines for "Geometric Menu"</i>
menu3.c	<i>Routines for "Utilities Menu"</i>
menu4.c	<i>Routines for "Equipment Menu"</i>
update.c	<i>Declaration of Main Data-Entry Menu and Some Support Routines for Update Operations</i>
update0.c	<i>Support Routines for Data Entry</i>
update2.c	<i>Data Entry Routines for "Add Wall", "Extend Wall", "Add Door"</i>
update3.c	<i>Data Entry Routines for "Add Junction", "Add Connection", "Extend Connection", "Extract Coordinate"</i>
update4.c	<i>Data Entry Routines for "Add Equipment", "Modify Equipment Text", "Connect Equip to Util"</i>
update5.c	<i>Data Entry Routines for "Delete Element"</i>

#### 4.1. Database Schema and Coordinate Systems

Figure 9 shows the entity/relationship diagram of the relational database that FLIP expects. Figure 10 lists the INGRES tables used by FLIP. All database tables need to be set up prior to running FLIP.

The database schema has been changed several times over the evolution of FLIP. Some tables, such as *door*, as well as some table fields, are really a remnant of FLIP's early versions and represent an entity whose necessity is arguable. The fixed database schema may be FLIP's biggest limitation. While it is very desirable to have a default schema for users who don't want to specify their own, it is also rather difficult to devise a schema that will satisfy all users or let them take into account local peculiarities.

A good portion of the database is kept in memory, especially information that greatly affects the redrawing of the main displays. Most of the menus that pop up as needed, for example the menu that lists the existing types of junctions, read the data directly from the database. Update operations that alter information kept in memory don't have an effect on the database until the operation is completed and confirmed. Since all the in-core information is read from the database only at start-up, multiple users running FLIP concurrently and performing update operations may produce data inconsistencies.

All geometric objects kept in memory store both the floating-point coordinates used for the whole facility (see section 3.7), and the integer coordinates in the coordinate system of the Global window.

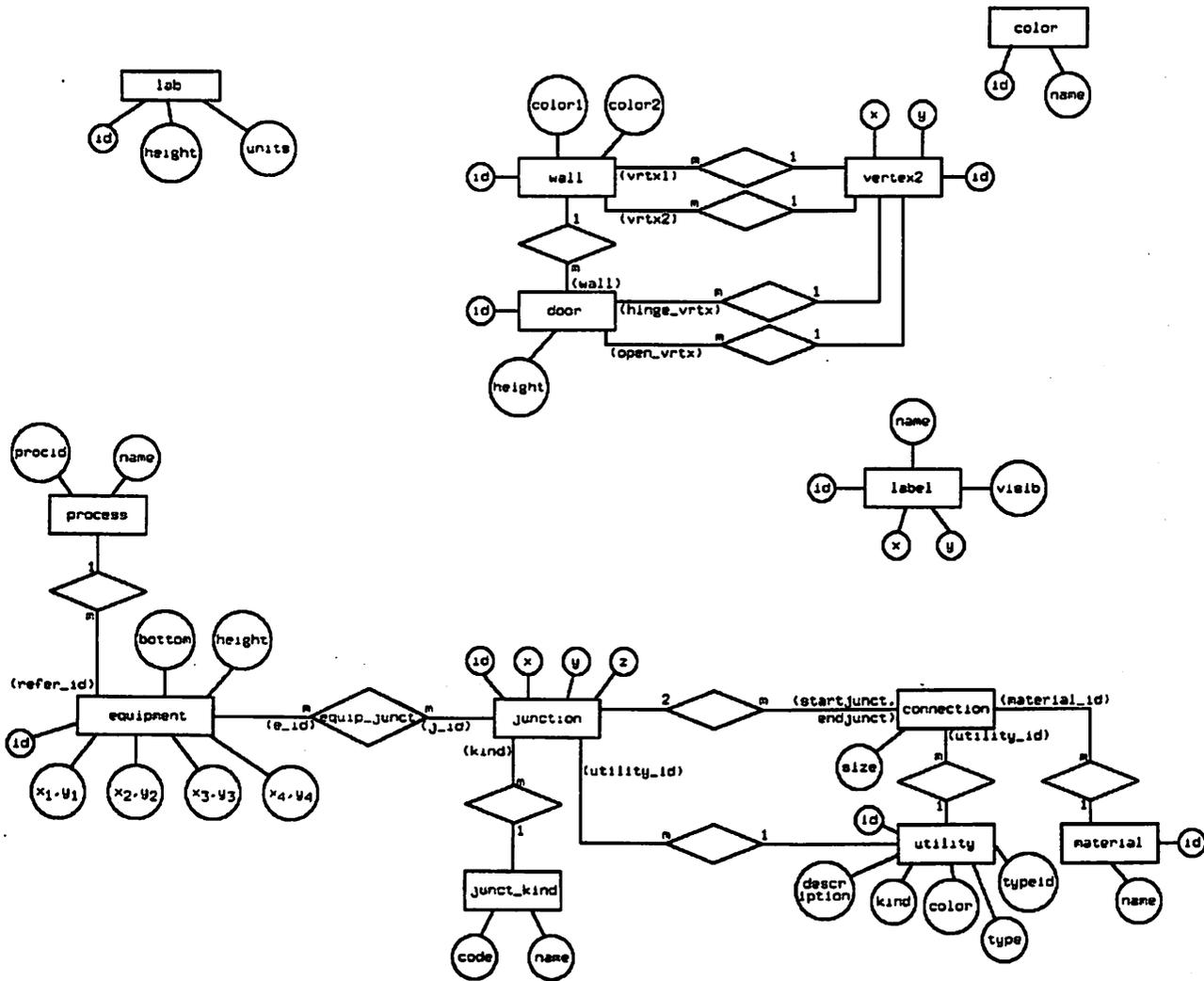


Figure 9. E.r. diagram of the database used by FLIP

## 4.2. General Graphic Tools Supporting FLIP

Several graphic modules were developed to provide support for FLIP's graphic operations but are not integral part of FLIP and may be replaced by better or more sophisticated ones when they become available. At the time when most of FLIP was written, few good graphic tools based on the X Window System (version 10) existed, and what was available was primarily the relatively low-level Xlib library. This situation may be changing soon, though, as many graphic "toolkits" have become available with version 11 of the X Window System and more may be developed and improved in the near future.

The following is the list of the graphic tools used by FLIP and a brief description of their main function.

Table	Fields	Size	Remarks
lab	id height units	integer1 float4 c10	Same floor-to-ceiling height assumed for entire lab Unit of measure for all coordinates (e.g., "meters")
vertex2	id x y	integer2 float4 float4	Two-dimensional vertices represent vertical projections onto floor A common coordinate system is used for the entire lab
wall	id vrtx1 vrtx2 color1 color2	integer2 integer2 integer2 integer1 integer1	All walls assumed to extend from floor to common ceiling Wall's first endpoint Wall's second endpoint Currently unused information Currently unused information
door	id wall hinge_vrtx open_vrtx height	integer2 integer2 integer2 integer2 float4	Wall that door is part of Door endpoint with hinges Door endpoint with latch Height of top of door from floor level
label	id name x y visib	integer2 c16 float4 float4 integer1	No data-entry support currently available for labels  Coordinates are for center of the label  "Visibility": if > 1 then label doesn't appear in Global Window
equipment	id refer_id bottom height x1 y1 x2 y2 x3 y3 x4 y4	integer2 integer4 float4 float4 float4 float4 float4 float4 float4 float4 float4 float4	Vertical elevation (from floor) of bottom of bounding box Height of bounding box Coordinates for vertices of bounding box
process	procid name	integer4 vchar/text12	No data-entry support currently available for this table Name of piece of equipment
equip_junct	e_id j_id	integer2 integer2	
junction	id x y z kind utility_id	integer2 float4 float4 float4 integer1 integer1	Height from floor level  Utility that junction is part of
junct_kind	code name	integer1 c20	Id for this table For example, "outlet", "faucet", etc.
connection	utility_id startjunct endjunct material_id size	integer2 integer2 integer2 integer1 float4	A "connection" is a portion of utility network between two junctions  Meaning of size depends on the type of material
material	id name	integer1 c30	For example, "steel pipe", "coaxial cable", etc.
utility	id kind color type typeid description	integer1 c10 integer2 c8 integer1 vchar/text60	For example, "sprinkler", "110 volts", etc. Id of symbolic color associated to utility For example, "gas", "power", etc.  A more verbose description of utility kind
color	id name	integer2 c30	This table isn't used. It's currently hardwired into FLIP Name of color corresponding to a given number

Figure 10. INGRES tables used by FLIP

<code>xcom</code>	<i>Used to turn a window into an I/O tool</i>
<code>xevents</code>	<i>Used to register requests to send X events to given routines</i>
<code>xmenuenv</code>	<i>Provides pop-up menus to make selections from</i>
<code>xtext</code>	<i>Produces a window good for doing text operations on files</i>

#### 4.2.1. The Graphic Tool `xcom`

This module turns a window into a “dialog box” (or “text window”) to provide textual I/O. FLIP uses this tool to implement the Console window. An arbitrary X window (only one at time) can be designated for use with `xcom` by the request :

```
xcom_window(window, font_name, background_color)
Window window;
char *font_name;          /* E.g., "9x15" */
int background_color;     /* Pixel value */
```

Afterwards, the function `xcom_write` may be used exactly like the ordinary `printf` to print on the designated X window, and `xcom_read` may be used to input strings of characters. A cursor is maintained on the screen and window resizes are properly handled automatically. A major feature of `xcom` is that printing on the screen can be done in *two* colors. The complete list of functions available in `xcom` is the following:

<code>xcom_window</code>	<i>Designate window for use by xcom</i>
<code>xcom_n_rows</code>	<i>Returns number of character rows in window</i>
<code>xcom_n_cols</code>	<i>Returns number of character columns in window</i>
<code>xcom_putstring</code>	<i>Print character string on window</i>
<code>xcom_write</code>	<i>Behaves just like the ordinary printf</i>
<code>xcom_colorwrite</code>	<i>Same as xcom_write but with color as argument</i>
<code>xcom_setcolor</code>	<i>Set value of current color</i>
<code>xcom_defcolor</code>	<i>Set current color to default (black)</i>
<code>xcom_event</code>	<i>Handles key-pressed or exposure events for xcom window</i>
<code>xcom_pending</code>	<i>Finds if there is unread available input</i>
<code>xcom_inpqueue</code>	<i>Returns number of unread characters in input buffer</i>
<code>xcom_read</code>	<i>Copy entire unread input into a string</i>
<code>xcom_input_flush</code>	<i>Flush away unread input</i>
<code>xcom_echo</code>	<i>Make copy of all window output into a given file</i>
<code>xcom_noecho</code>	<i>Stop action of xcom_echo</i>
<code>xcom_invoke</code>	<i>Invoke a user-passed function whenever there is I/O</i>
<code>xcom_noinvoke</code>	<i>Stop action of xcom_invoke</i>

#### 4.2.2. The Graphic Tool `xevents`

This is a very general tool that facilitates the handling of input coming from the window system. Such input comes packaged in the form of a structure called “XEvent.” XEvents are

placed on a queue by the window system as they arrive, so a typical application program will have a central loop reading from the queue and dispatching events to routines. This dispatching is the part that the tool *xevents* can help with. It permits the programmer to associate routines with X windows and XEvents of given types. After all the associations are registered, the main loop of the program needs only continue calling the function `ProcessEvents`, which will wait for the first element in the XEvent queue to become available and then will invoke the appropriate routine. The following routines make up *xevents* :

<code>SetEventHandling</code>	<i>Register association of function to window and event type</i>
<code>RemoveEventHandling</code>	<i>Remove association</i>
<code>DefineFunctionToInvoke</code>	<i>Register association of function to window</i>
<code>ProcessEvents</code>	<i>Read from head of XEvent queue and dispatch to appropriate function</i>
<code>ProcessGivenEvent</code>	<i>Use XEvent passed as argument to do the dispatching</i>
<code>DiscardEvents</code>	<i>Flush unwanted elements from XEvent queue</i>

#### 4.2.3. The Graphic Tool *xmenuenv*

This tool provides the capability of making a selection from a menu of choices. Various other menu tools are in existence today and it may be possible to create an interface simulating *xmenuenv* using other menu programs. Early in the development of FLIP, it appeared that existing menu tools could not be easily used with the control structure that FLIP then had – but which has completely changed since – and so I created *xmenuenv*. Some of its highlights are : 1) the support it provides for registering and automatically invoking a help function; 2) The largest font that will fit in the space for an entry name is automatically picked for each individual entry and gets automatically adjusted if the menu gets resized<sup>4</sup>. Weaknesses of *xmenuenv* include the fact that menus are not very dynamic entities, and for example all their entries must be specified at creation time and no entries can be added or deleted afterwards. (Existing entries may be altered, however.)

All programs making use of *xmenuenv* should have the statement :

```
#include <xmenu.h>
```

which, among other things, defines the data types `Menu` and `Entry` . The following functions are available in *xmenuenv* :

<code>DefineMenu</code>	<i>Define a menu but don't display it</i>
<code>CreateMenu</code>	<i>Define and display a menu</i>
<code>ModifyMenu</code>	<i>Similar to DefineMenu, but it re-uses the window of a previous menu</i>
<code>DisplayMenu</code>	<i>Bring a menu to view</i>
<code>HideMenu</code>	<i>Hide a menu from view</i>
<code>DestroyMenu</code>	<i>Hide menu and mark it as non-existent</i>
<code>InvokeMenu</code>	<i>Process an XEvent that corresponds to a menu window</i>

---

<sup>4</sup> One person told me he doesn't like to look at entry names of differ sizes; however, if desired, all names can be appear with the same size by defining a menu with same-length entry labels (padding with blanks as necessary.) The automatic font scaling allows to have long descriptive names for some of the entries whose meaning would be obscure if described with a short name, while at the same time not forcing a small font to all other entries.

MenuEvent	<i>Similar to InvokeMenu but if the XEvent corresponded to a selection then return a pointer to the entry selected</i>
GetMenuChoice	<i>Wait until an entry from a menu is selected and return a pointer to it (if an XEvent not belonging to the specified menu is seen while waiting, it is given to a function passed as argument)</i>
Kill_GetMenuChoice	<i>Abandon the wait started in a previous call to GetMenuChoice</i>
FirstMenu	<i>Return pointer to first menu created</i>
ModifyEntryName	<i>Change name of an entry and correct the display</i>
Highlight	<i>Highlight a given menu entry</i>
NoHighlight	<i>Remove highlight from a given menu entry</i>
InvokeSelection	<i>Behave exactly as if given selection had been selected with the mouse</i>
WindowAssoc	<i>Return a pointer to the entry associated to a given window</i>
SetHelpFunction	<i>Designate a function as being the help function</i>
EnterHelpMode	<i>The next selection will invoke the help function instead of the usual function associated to it</i>

#### 4.2.4. The Graphic Tool *xtext*

This tool provides a dedicated window where text operations, such as viewing and editing a file, may be done. Its implementation makes use of the familiar terminal emulator program called *xterm*.

There are two parts to *xtext*: one is a server, called *xtext\_backend*, that runs as an independent process, and the other is a library of functions, called *xtext\_lib*, that the client program can use to communicate with the server. The server is started with a call to the *xtext\_lib* function *xtext\_start*, which forks off a child process running the program *xterm* with some arguments. The arguments passed to *xterm* include: 1) a specification that the *xterm* window should be created in iconized form in the upper left-hand side of the screen; 2) the value of the file descriptor for the reading end of a UNIX pipe that is created by the calling program before forking. This half of the pipe will be used for interprocess communication between *xtext\_lib* and *xtext\_backend*.

The newly created process acts as a server, i.e. keeps reading from the pipe, interprets the messages coming through and performs corresponding actions. In addition, just before entering the interpreting loop, it determines the ID of the window it is associated to and unmaps (hides) it. Notice that FLIP starts up *xtext* during initialization, which explains the little iconified window sometimes appearing in the upper left-hand of the screen and soon disappearing.

It would be possible to modify *xtext* in such a way that it could be used to implement the Console window and make the tool *xcom* unnecessary. However, a major drawback would be the fact that printing in two colors would no longer be possible (unless the program *xterm* gets expanded some day to be able to handle two-color printing.)

The functions available in *xtext\_lib* are:

<i>xtext_start</i>	<i>Create a separate server process producing an xterm window</i>
<i>xtext_end</i>	<i>Eliminate the xtext window and kill the server process</i>

<code>xtext_clear</code>	<i>Clear the xtext window</i>
<code>xtext_map</code>	<i>Map (make visible) the xtext window</i>
<code>xtext_unmap</code>	<i>Unmap (remove from view) the xtext window</i>
<code>xtext_more</code>	<i>Display the file passed as argument in the xtext window using the "more" program</i>
<code>xtext_vi</code>	<i>Edit (using "vi") in the xtext window the file passed as argument</i>
<code>xtext_instr</code>	<i>Execute a generic instruction in the xtext window as if typed to an xterm terminal program</i>
<code>xtext_command</code>	<i>Same as xtext_instr but the command is passed in a different format</i>

Upon normal termination, FLIP issues the `xtext_end` command to kill the `xtext` process. Abnormal terminations, typically come across while debugging, will not send the request and, as a result, the `xtext` process will have to be killed manually. Some day, it might be desirable to utilize the currently unused second half of the communication pipe between the two processes, in order to have `xtext` periodically check on the status of the calling program and automatically terminate if the program stops responding.

### 4.3. General-Purpose Modules in FLIP

A few subroutines in FLIP are of great generality and completely isolated from the rest of the program. They can be of interest to other program, too, or maybe one day they may find their way into more general tools or libraries and be taken out of FLIP. A selected list of them is the following:

<code>PolyArea</code>	<i>Compute the area of an arbitrary (convex or concave) polygon</i>
<code>CompareWords</code>	<i>Match one sentence to a given group of sentences</i>

The first of those two functions, `PolyArea`, is in file `geom.c`. A more detailed description is :

```
typedef struct {
    float x, y;
} point;

float
PolyArea(Plist, n)
    point Plist[];
    int n;
/* Compute and return the area of an arbitrary (convex or concave) polygon with n
   vertices, listed in order in the array Plist. In case of error return a negative
   number. Overlapping vertices or edges are permissible; winding polygons are
   not. The area is determined by first finding the area of the convex hull of the
   polygon and then recursively subtracting the areas of all polygons forming the
   concave portions, if any. Error codes: -1. (polygon has less than 3 sides); -2.
   (memory allocation failure); -3. (winding polygon)
*/
```

The second function, `CompareWords`, and related functions are in file `string.c`. The function `CompareWords` is built on top of a function called `StrCmp`, and it is used as a building block for a higher-level function called `SearchList`. The listing of those 3 functions, from lower to higher level, follows:

```
StrCmp(s1, s2)
char *s1, *s2;
/* Case-insensitive, error-tolerant, string comparison. Return 0 if the strings are
equal. Return 1 if they differ only by a single-character substitution (unless string
length is 1). Return 2 if they differ only by the inversion of two consecutive char-
acters. Return 3 if s1 differs from s2 only because of a single-character omission
(unless s2 has only one character). Return 4 if s1 differs from s2 only because of
a single-character addition (unless s2 is the empty string.) Return 5 if the strings
differ by at least 2 characters and one is a leading substring of the other. Return
-1 if any of the above matches fails or if either string is NULL.
*/
```

```
CompareWords(ptr1, n_wd1, ptr2, n_wd2, match_no, err_no, inversion)
char **ptr1, **ptr2; /* WARNING: argument ptr2 gets trashed */
int n_wd1, n_wd2, *match_no, *err_no, *inversion;
/* Match the two group of words pointed to by the elements in the arrays ptr1 and
ptr2. Each array element is expected to point to a null-terminated single word.
The number of elements of the first array is n_wd1 and that of the second is
n_wd2. It is slightly more efficient to have n_wd1 <= n_wd2. The matches are
done on a word-by-word basis, based on the case-insensitive, error-tolerant word
comparator StrCmp(). Perfect word matches, as determined by StrCmp(), are
looked for first, attempting to find matches that maintain the word order in the
two groups. A second pass will then look for imperfect word matches, still
attempting to retain the positional order for ALL matches (perfect or otherwise) in
the two groups. [The reason for the two separate passes is to avoid using up
words for imperfect matches when they may give perfect ones. E.g., matching
"nat gnat" and "net gnat" might use up the "nat" from the 1st and the "gnat" from
the 2nd, producing just 1 imperfect match instead of the perfect "gnat" = "gnat"
plus the imperfect "nat" and "net".] The variable 'match_no' is set to the TOTAL
(perfect or not) number of words that match, 'err_no' is set to the number of
errors (imperfect matches), and 'inversion' is set to 0 if the matches found main-
tain the same positional order in both group of words and to 1 otherwise.
*/
```

```
SearchList(str, str_list, match1, match2)
char *str, *str_list[]; /* WARNING: 'str' will be trashed */
int *match1, *match2;
/* Match the string 'str' against all the strings in the array of strings 'str_list'. The
last entry in 'str_list' is expected to be a pointer to NULL. Each string is con-
sidered to be a list of blank(s)-separated words. Return -1 if the search is aban-
doned because the strings 'str' is too long, Return -2 if the search is abandoned
because one of the strings in str_list is too long. Return -3 if the string 'str' con-
tains only blanks. Return 0 otherwise. The matches are done on a word-by-word
basis, are insensitive to case and are tolerant to slight errors between the two
```

words [further details under the function `StrCmp()`.] The variable `match1` will be set to the index in the array `str_list` of the best match found, or -1 if none is found. The variable `match2` will be set only in case of ambiguity of best matches, or -1 otherwise. Best match is the one will the highest number of individual words that match [further details under `CompareWords()`]; if this number is 0 then no match is considered to have been found. If more than one pair has highest number of words that match, then the one with fewest errors in individual-word matches will be selected; if there is more than one, then it will be the one whose words match in the same order; if more then one, then it will be the one with the same number of words (this is necessary to distinguish, e.g., "A" from "A B"); if more than one or none then an ambiguity is declared.

\*/

#### 4.4. Flow Control in FLIP

Program execution starts in file `main.c` . Various initializations are carried out, mostly using routines in `init.c`, then the `xtext` tool is started up, a portion of the database is read into memory (files `dbm.c` and `memory.c`), other initializations follow, and finally the main loop of the program is entered by calling the function `EventHandler` (in file `handler.c`) Prior to such call, all the routines that create the various X windows will have registered, by means of calls to the tool `xevents`, which of the XEvents are to be selected for each window and what routine is to be called whenever that combination of XEvents and windows are seen. The routine `EventHandler` contains the main loop of the program. It repeatedly calls the function `ProcessEvents`, part of the tool `xevents`, which reads the next element from the front of the XEvent queue and dispatches it to the pre-requested routines within FLIP.

The most complex part of control flow is when some routine expects an input provided by the X system. The routine cannot just loop and wait for the XEvent to come around because there may be other XEvents that need immediate attention (e.g., windows may have been resized and need to be redrawn, etc.) What FLIP does, instead, is to use a group of routines, `WaitXEvent` and related ones, all contained in the file `handler.c` . These routines are given as arguments one or more XEvents and they will enter a loop that keeps looking at the front of the XEvent queue for any of the desired events. Any other event that may be encountered while waiting for the desired ones will be sent to the regular event dispatcher contained in the `xevents` tool (named `ProcessGivenEvent` .)

In addition to XEvents, FLIP deals with structures, similar in concept to XEvents, simply called *Event* . Each of those structures can be thought of as a local, FLIP "event", representing a package of information about some higher-level object. Its components are described in `flip.h` .

#### Acknowledgements

I would like to especially thank the following people. Prof. David Hodges for making this project possible. James Hopkin for most of the data entry, as well as for design suggestions and feedback on this paper. Katalin Voros for extensive critique and proofreading of this paper, and for user feedback on FLIP's functions. David Mudie for design ideas and for detailed feedback on this paper. Bob Hamilton for his help in proofreading.

## References

1. Jim Gettys, Ron Newman and Tony Della Fera, "Xlib - C Language X Interface. Protocol Version 10", MIT Project Athena, 1985
2. Tom Muller and Dipti Shabde, "Facilities Maintenance System, UCB Microlab", Spring 1986, unpublished.