XBasic

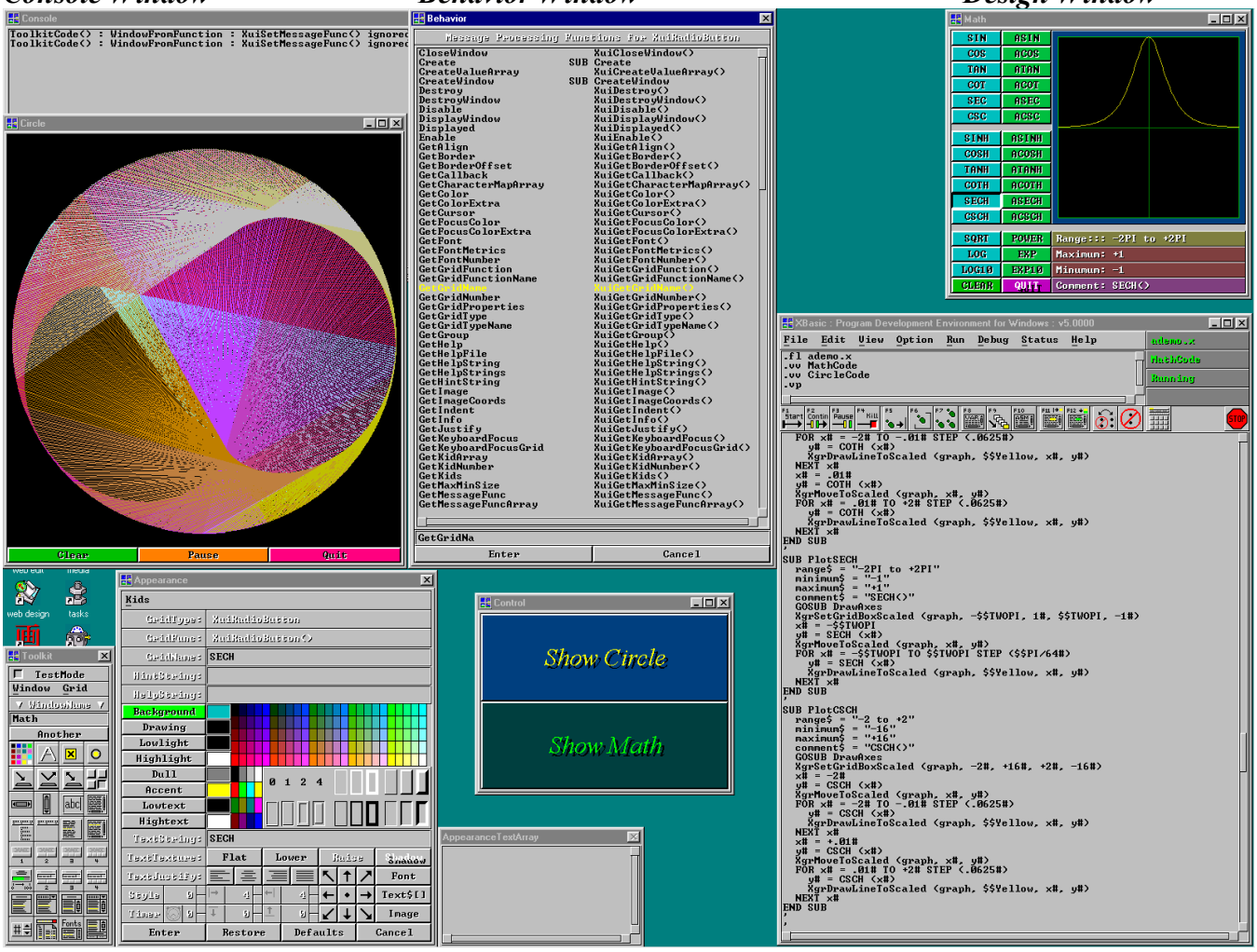
Program Development Environment
(PDE)

Installation
and
Introduction

Revision 0.0022
November 1, 1995

Table of Contents

Console Window *Behavior Window* *Design Window*



Toolkit *Appearance Window* *Main Window*

More windows than normally displayed at once
Shown on Windows95

- ***Main Window*** ***- enter, edit, load, save, run, debug programs - and set options, get help, do most development.***
- ***Console Window*** ***- conventional BASIC console I/O.***
- ***Toolkit Window*** ***- create, destroy, load, save design windows - and select controls into visible design window.***
- ***Design Window*** ***- interactive graphical GUI design layout.***
- ***Appearance Window*** ***- set appearance and behavior properties of the selected grid.***
- ***Behavior Window*** ***- display the actions of the selected grid for each message.***

Documentation

All documention is distributed as printed manuals or CDROM / diskette files. Depending upon how you acquired the documentation, you have either or both. If you have only computer files, you'll probably want to view or print them.

All documents were created with Microsoft Word 97 and saved as typical documentation files. The documents contain only two fonts, Times New Roman and Courier New, in normal***,*** **bold,** *italic*, ***and bold italic. These TrueType fonts are normally supplied with Microsoft Windows, and should already be installed on your system. If you print these documents with other fonts, line and page breaks will often be poorly placed, and other formatting problems may occur as well.*** **`xman.dot`** ***is a document template file included with and referenced by these documents.***

To view a document, run Microsoft Word 97 or higher, load a file, and view per normal practice.

To print a document, run Microsoft Word 97 or higher, load a file, and print per normal practice. Versions of Word before and after Word 97 may print documents somewhat differently, but such differences are usually not too serious.

Documentation Conventions

This font is Times New Roman: normal, **bold**, *italic*, ***bold-italic***.

`This is Courier New: normal, bold, italic, bold-italic.`

Most documentation text is Times New Roman, with portions emphasized with **bold**, *italic*, and ***bold-italic***.

The following kinds of documentation text appear in **`Courier New`**:

- Keywords and source program text of any length (one word to whole programs).
- Text on GUI components like labels and pushbuttons, such as **`Enter`** or **`Cancel`**.
- Pulldown menu headings and items, like **`FileLoad`**.

Source program text meant as descriptive placeholders are in *`italic`* or ***`bold-italic`***. For example, **`XuiGet`*`Property`*`()`** means:

```
XuiGetAlign()
XuiGetBorder()
XuiGetColor()
... etc ...
```

where ***`Property`*** is the placeholder for **`Align`**, **`Border`**, **`Color`**, etc.

Document Set

`\xb\doc\intro.doc` - *Installation and Introduction*
`\xb\doc\language.doc` - *Programming Language Guide and Reference*
`\xb\doc\graphics.doc` - *GraphicsDesigner Guide and Reference*
`\xb\doc\guigrids.doc` - *GuiDesigner Convenience Functions*
`\xb\doc\guigrids.doc` - *GuiDesigner Standard Toolkit Grids*
`\xb\doc\guiguide.doc` - *GuiDesigner Guide*
`\xb\doc\guirefer.doc` - *GuiDesigner Reference*
`\xb\doc\library.doc` - *Function Libraries*
`\xb\doc\advanced.doc` - *Advanced Features and Topics*
`\xb\doc\*.doc` - *Miscellaneous Information*

Introduction

What is this software?

This software is a comprehensive program development environment integrating a powerful editor, compiler, debugger, and *GuiDesigner* into a seamless working environment that encompasses the whole process of creating fast, efficient, reliable 32/64-bit programs.

PDE is an acronym for *program development environment*.

Program Development

Programmers perform several steps to develop a computer program:

1. *Type in and edit the program.*
2. *Run compilers/assemblers/linkers to translate the program into executable form.*
3. *Run the program and test it for errors.*

Many programmers run a separate program to perform each step:

Editor	*- Enter and edit the program*
Compiler	*- Translate program to assembly language*
Assembler	*- Convert assembly language files to object files*
Linker	*- Link object files into an executable file or library*
Debugger	*- Run program, observe errors, quit debugger and return to editor step*

This PDE is a single program that performs these & other functions. Because the components are seamlessly integrated, switching from one programming phase to another is natural and instantaneous.

GuiDesigner

GuiDesigner is an important additional capability built into the PDE**.** *GuiDesigner* helps you create attractive, sophisticated, user-friendly graphical user interfaces (GUIs) without writing any code.

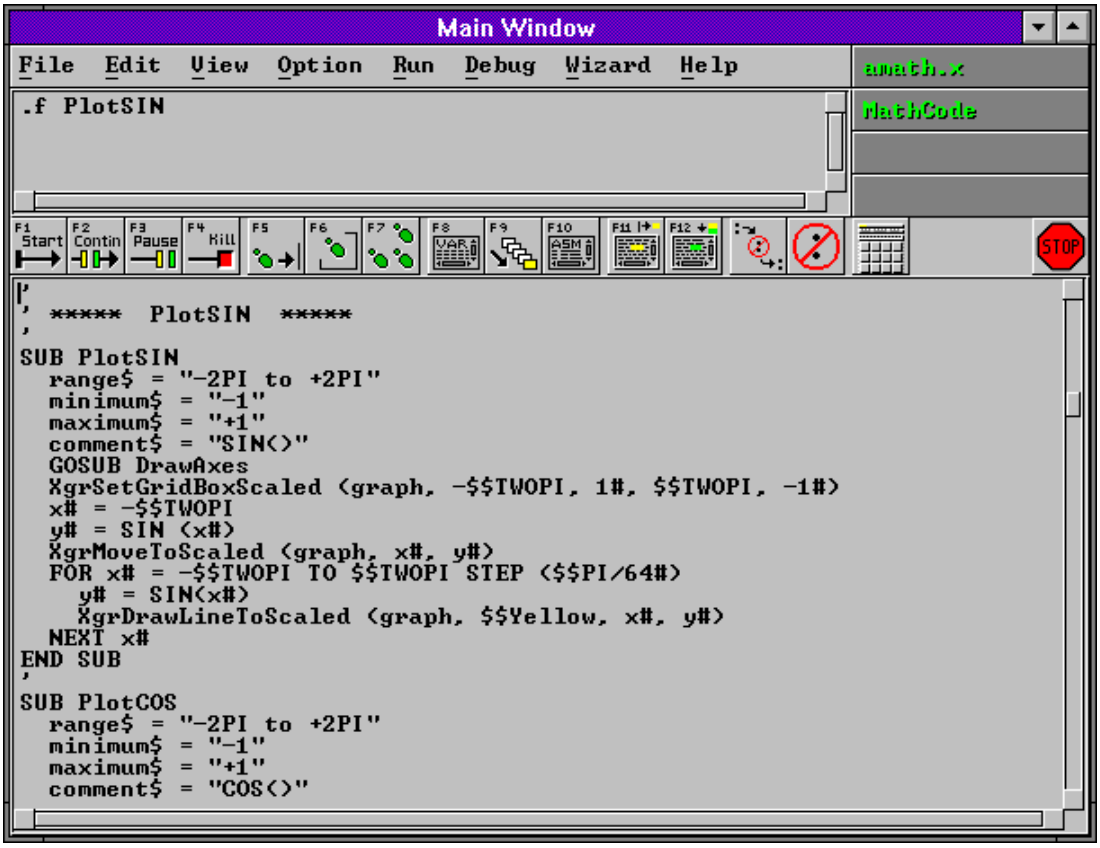
Naturally you'll want to add code of your own to give life to the GUI, but *GuiDesigner* does the hard part for you - it creates your GUI.

Requirements

Any 80386 / 80486 / 80586 / Pentium / P5 / P6 compatible computer running Microsoft Windows 3.1, Windows 3.11, Windows 4.0 aka Windows95, or WindowsNT 3.10, 3.50, 3.51+ should run this software properly, assuming the following minimums are met or exceeded:

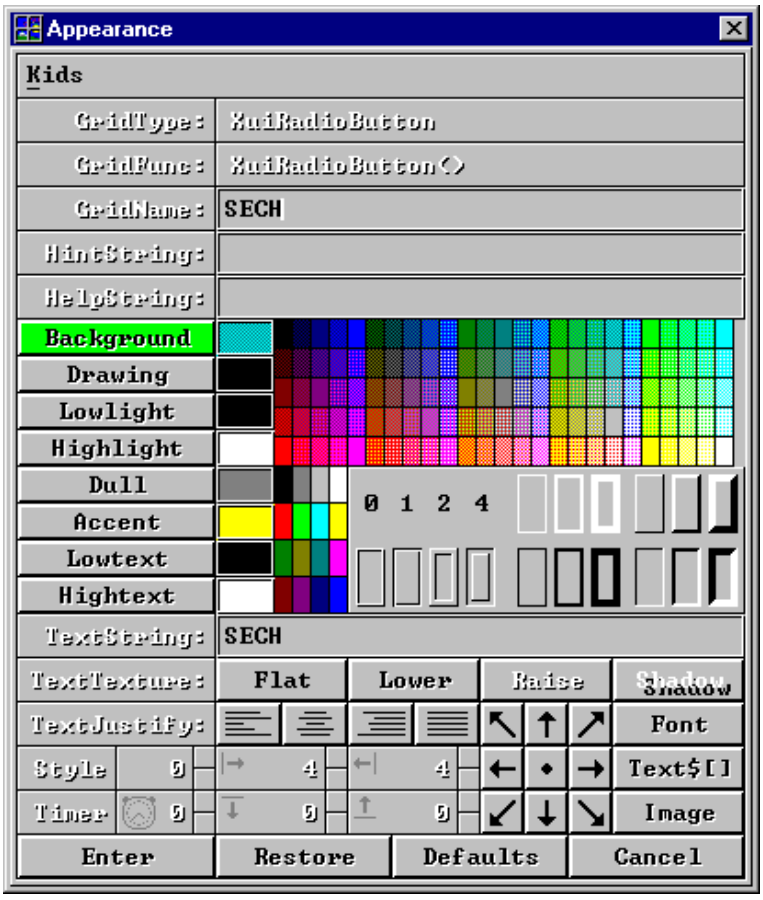
- *8 megabytes or more of accessible RAM (prefer 16MB or more).*
- *8 megabytes of more of available hard disk space (prefer 16MB or more).*
- *8 megabytes or more of swap file (prefer 16MB or more).*
- *800x600 video/screen resolution (prefer 1024x768 or more)*

This software has been reported to run [slowly and with a few annoyances] on *some* systems that have as little as 4MB of RAM plus 12MB swap file and 640x480.



Main Window

Toolkit Window



Appearance Window

Installation

Updated Instructions

This following section is hopelessly obsolete (pre-Windows95). To install downloaded XBasic, put the downloaded self-extracting zip archive file (filename xb.exe or xbpro.exe or xbsource.exe) into the root directory of one of your hard drives (as in c:\ or d:\ or e:\) and execute the file. XBasic will install itself into x:\xb where x: is the name of the hard-drive you chose. Then launch Windows Explorer, get into directory x:\xb, and drag and drop xb.exe onto your desktop to create an XBasic shortcut icon.

Preparation

Get into DOS. If you're in Windows or WindowsNT, you can click the MS-DOS icon to get into DOS. The following instructions call the source floppy or CDROM drive **`s:`** and the destination hard disk drive **`d:`**. Perform installation from the DOS command prompt.

This software runs directly on Windows 4.0 aka Windows95 and WindowsNT. This software also runs on Windows 3.1 and Windows 3.11 if you install Win32s or already have Win32s installed.

Floppy Diskette Installation

Install the software starting with floppy diskette labeled #1 and proceed sequentially until complete.

!!! Don't forget to **`unzip`** from the root directory !!!
!!! Don't forget the **`-d`** option in **`unzip`** !!!

1. Insert 1st floppy diskette into **`s:`**
2. Enter **`d:`**
3. Enter **`cd \`**
4. Enter **`copy s:\unzip.exe .`**
5. Enter **`unzip -d s:\xb0.zip`**
6. Remove 1st floppy diskette from **`s:`**
7. Insert 2nd floppy diskette into **`s:`**
8. Enter **`unzip -d s:\xb1.zip`**
9. Remove 2nd floppy diskette from **`s:`**
10. If you have the professional edition (makes standalone executables & DLLs):
    Insert 3rd floppy diskette into **`s:`**
    Enter **`unzip -d s:\xb2.zip`**
    Remove 3rd floppy diskette from **`s:`**
11. Enter **`copy \xb\run\xb.dll \windows\xb.dll`**
12. Enter **`copy \xb\xb.dll \xb\xb.dup`**
    [only necessary on Window95 until Windows95 bug is fixed]

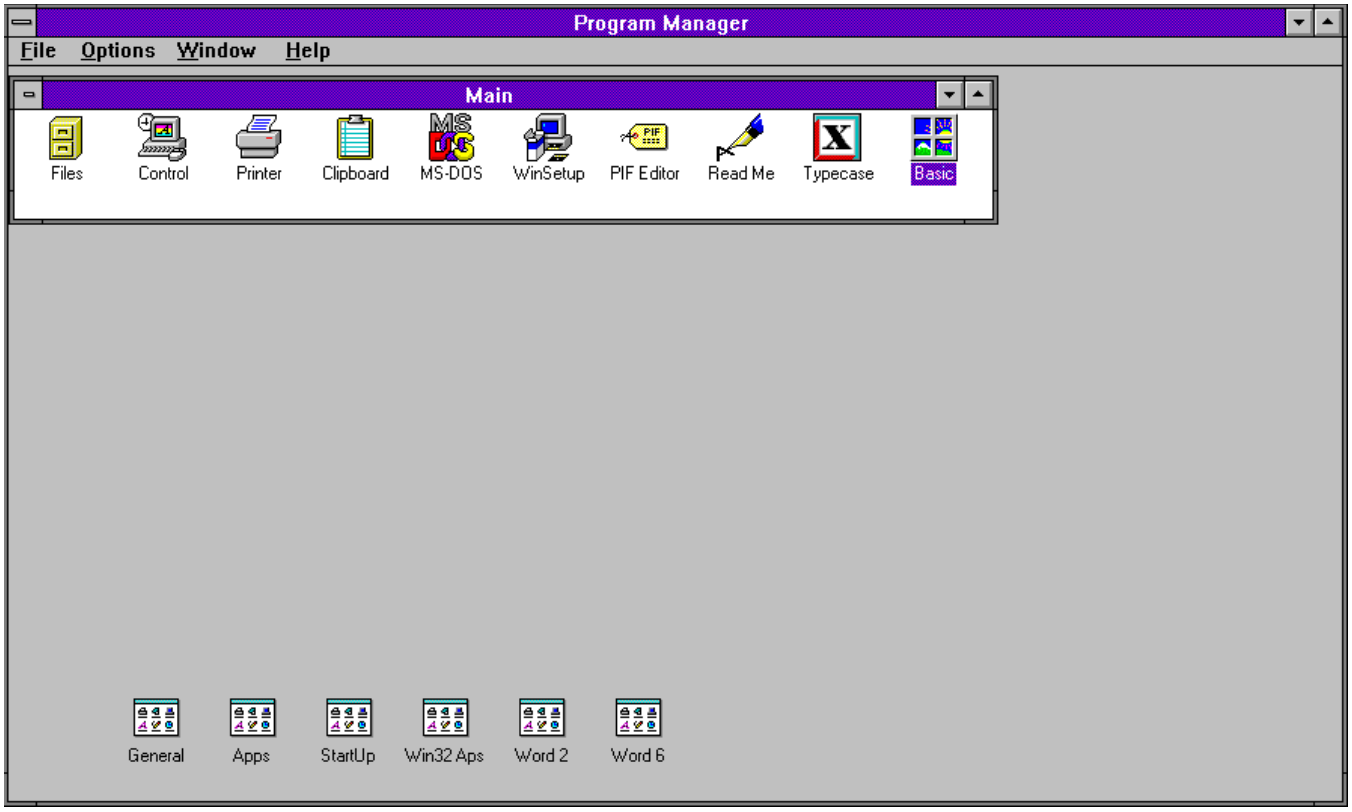
To install Win32s (Windows 3.1 and Windows 3.11 only):

1. Insert the 1st Win32s floppy diskette into **`s:`**
2. At the DOS prompt, enter **`win`** to start Windows
3. In the main menu, select File Run
4. Next to "Command Line:" enter **`s:setup`**
5. Follow Win32s installation instructions.

Program Item

To create a *program item* (for Windows95, see page bottom):

1. Select **FileNew** in the ProgramManager window.
2. Select **Program Item**.
3. Next to **Description:**, enter **Basic**
4. Next to **Command Line:**, enter **d:\xb\xb**
5. Next to **Working Directory:**, enter **d:\xb**
6. Select the **Change Icon...** button and select your preferred icon.
7. Select the **OK** button to complete the process and create your program item.



Windows95 Shortcut

To create an *shortcut* (puts icon on desktop/background) :

1. Select **Start Programs WindowsExplorer**
2. In WindowsExplorer, select appropriate drive & directory to find **xb.exe**.
3. Drag the icon next to **xb.exe** to empty area on desktop background.
4. Click right button on icon and select rename, then enter **Basic**.
5. To run the program, simply double click on the **Basic** icon.

Getting Started

Start the Program Development Environment

Start this program the same way you start any program on your computer; click its icon, or enter **`xb`** in the system **`FileRun`** menu.

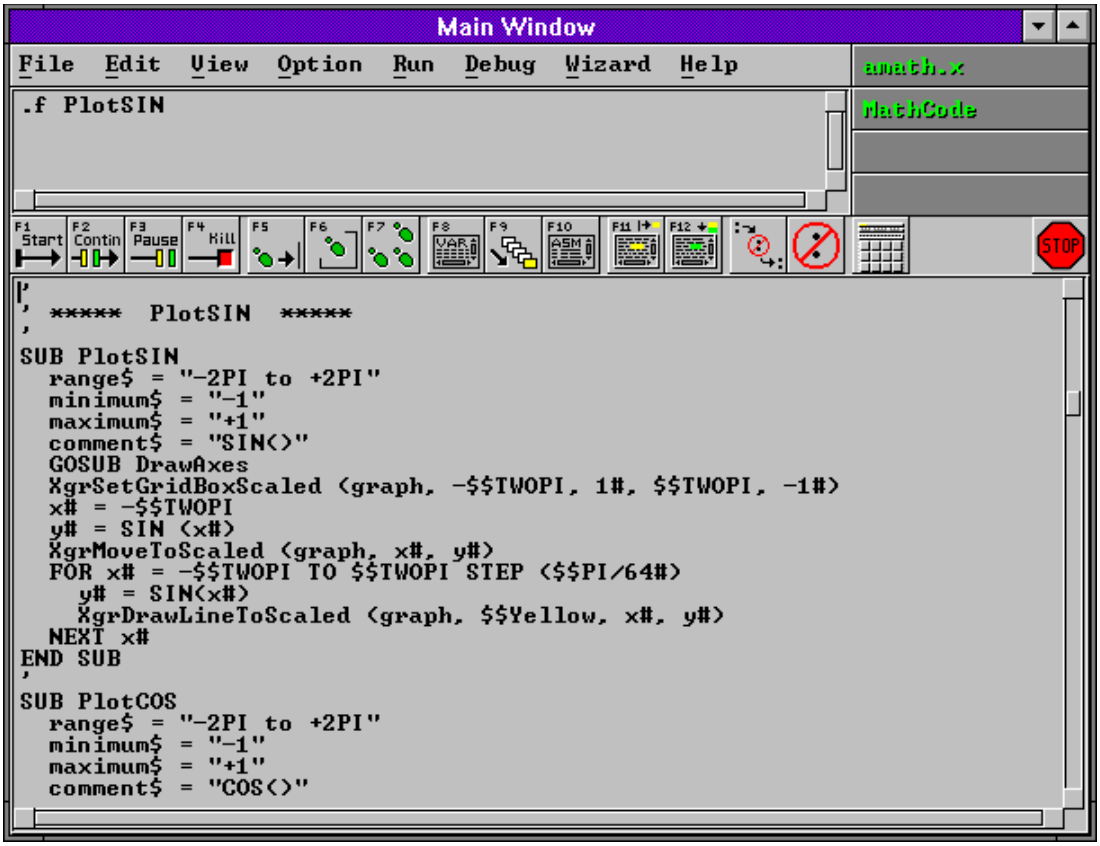
A few seconds will pass as the program loads from disk to memory and initializes. Two windows will appear on your display:

The *console window* appears in the upper left corner of your display.
The *main window* appears in the lower right corner of your display.

Main Window

As the following figure illustrates, the main window contains:

main menu	-	Operates pulldown menus of commands
status labels	-	Display status of current program, function...
upper text area	-	Displays general info & accepts commands
hot buttons	-	Execute common commands with one click
lower text area	-	Enter, edit, debug programs here

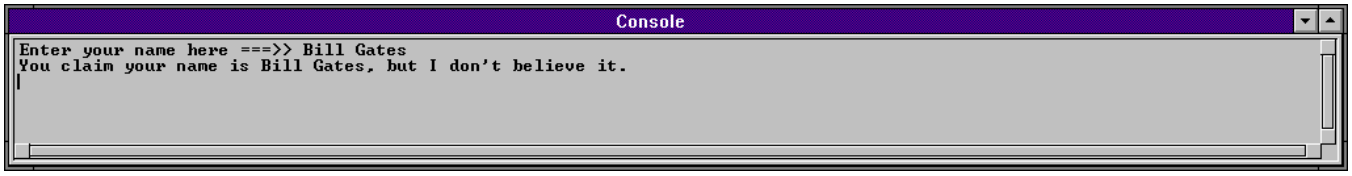


Main Window

Console Window

The console window is where conventional BASIC language console I/O operates. For example:

```
name$ = INLINE$ ("Enter your name here ===>> ")
PRINT "You claim your name is "; name$; ", but I don't believe it."
```



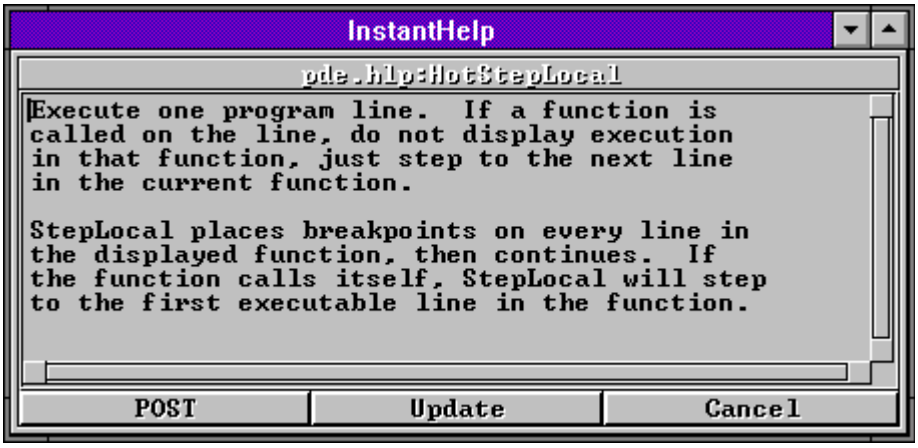
Console Window

Instant Help

At any time you can point the mouse cursor at any GUI component and press the right mouse button. An *InstantHelp* window appears instantly to explain the purpose of the component. You can sequentially click on every component in a window to get a whirlwind tour of that window.

Whenever you have a question, remember InstantHelp.

For help on the programming language, function libraries, or other aspect of this software, select the appropriate catagory in the **Help** pulldown menu in the main window menu bar.



InstantHelp Window

Main Window

Main Menu

Across the top of the main window is the *main menu.* Each word on the menu-bar names a category of commands. When you *select*[1] a category, a pulldown menu appears below it. Any command in the pulldown menu can be executed by selecting it.

To select a menu catagory, point at it with the mouse[2] cursor and depress the left button[3]. Or press **`Alt+?,`** where **`?`** is the underlined letter of the menu bar entry - as in **`Alt+F`** for **`File`**.

Upper Text Area

The *upper text area* is a small text area directly below the menu bar. Messages are printed here, and you can enter keyboard commands.

Lower Text Area

The *lower text area* has room for a number of visible lines of text. You can edit programs and text files in this area.

Text Cursor - Keyboard Focus

A *text cursor* is usually visible in one of the text areas. It is the shape of a large **|** and may blink. If the text cursor is hard to see, select **`OptionColorTextCursor`** in the main menu bar and change color.

The text area that contains the text cursor has keyboard focus - when you type on the keyboard, characters appear to the right of the cursor. You can place the text cursor anywhere in text areas - just point the mouse cursor and click the left button. To toggle back and forth between the two text areas, press the **`Escape`** key. This instantly moves the text cursor back to its previous position.

Status Labels

In the upper right corner are *status labels* that display the name of the loaded file, the name of the function displayed in the lower text area, the current program state, and compile errors.

Hot Buttons

Between the upper and lower text area are some *hot buttons* for quick execution of common commands. A single click or keystroke will execute a hot-button command.

---

1 ***Select*** is a general term that includes any means of activating or selecting an entry or command, including mouse button clicks, specially-defined keystrokes, and keyboard commands.

2 Any functionally equivalent ***pointing device*** can be used in place of a ***mouse***. Trackballs are one popular alternative. This document prefers "mouse" to the more abstract "pointing device".

3 On the mouse, the ***left button*** is generally considered the ***primary button***. On some systems the order of the buttons can be reversed, making the right button primary.

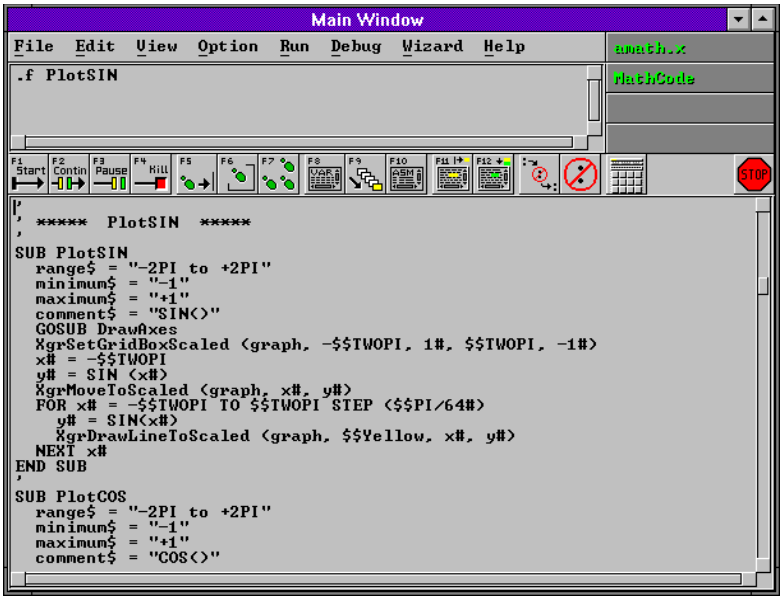
Window Title Bar ===>>

Main Menu ===>>

Upper Text Area ===>>

Hot Buttons ===>>

Lower Text Area ===>>



Hot Buttons

Click hot buttons to instantly execute the following commands:

ABORT	Stop any active load, parse, or compile.
Start	Start program execution - compiles first if necessary.
Continue	Continue program execution after a Pause.
Pause	Pause program - execution can be continued.
Kill	Terminate program - execution cannot be continued.
ToCursor	Execute program up to text insert cursor.
StepLocal	Step to next executed line in displayed function.
StepGlobal	Step to next executed line in program.
Variables	Display variables box to view and/or change variables.
Frames	Display frames box to view sequence of call frames.
Assembly	Display assembly language for text insert cursor line.
Find	Repeat most recent Find (find string in program).
Replace	Repeat most recent Replace (replace string in program).
Toolkit	Display or Hide *GuiDesigner* toolkit.

Executing Commands

Selection

Commands are executed when you *select* them. The most obvious way to select something on the screen is to *click* on it - which means point the mouse cursor at it and push/release the left mouse button.

You can also make menu selections with *keyboard accelerators*.

These specially defined sequences of keystrokes have the same effect as mouse clicks, but are often quicker because you just press keys without moving your hands from the keyboard.

For example, to select and execute the **FileLoad** command, press the **Alt** key, then the **f** key, then release both keys. This selects and displays the **File** menu. Now press the **l** key to select and execute **FileLoad.**

Commands

There are several ways to execute commands, and each has its advantages. Often one is quicker, more natural, or more convenient. Sometime a command feature is available only when the command is is executed in a particular way - you can't enter arguments with the mouse, for instance.

You can execute commands in the following ways:

*Click a command catagory in main menu, then an entry in its pulldown menu (*RunStart*).*
Enter a dot-command in the upper text area (.rs*).*
*Type an accelerator key sequence (*Alt-r, s*).*
*Type a hot key synonym key (*F1*).*
*Click a hot-button (*Start*).*

Dot Commands

You can enter a *dot command* in the upper text area to execute any pulldown menu command. Just type a dot plus the two letter command abbreviation in the upper text area.

If the text cursor is in the main text area, you can press the **Escape** key first move it to the upper text area. You don't have to enter the optional **argument** after the dot command - a prompt window will appear and ask for the information. But you'll find it's much faster to enter **argument** too, since you can do everything with keystrokes.

File *menu*

The **File** menu contains commands and actions that relate to program and text files. To select the file menu, click **File** on the main menu bar, or press **Alt+f** on your keyboard. A menu with the following commands will appear:

```
New          .fn [ filename ]    Begin new file.
TextLoad     .ft [ filename ]    Load text file.
Load         .fl [ filename ]    Load program file.
Save         .fs [ filename ]    Save text/program file.
Mode         .fm [ t | p ]       Toggle text/program mode.
Rename       .fr [ filename ]    Rename file begin edited.
Quit         .fq                 Quit development session.
```

You can execute any of these commands by clicking on its name in the pull-down menu, or by typing the first letter of its name on your keyboard. Or, you can execute them directly by entering dot commands from the preceding table.

FileNew

FileNew clears the main text area and abandons the text or program. The contents of the text area are not saved by this command, so be sure you save the file first if you may need it later.

FileNew asks what kind of new file to create. You can choose:

Text	***- Conventional text file***
Program	***- Native program with PROLOG plus any number of functions***
GuiProgram	***- GuiDesigner program with core GUI program created for you***

The *Programming Language Guide and Reference* manual contains more information the required and optional elements of all programs. The *GuiDesigner Guide* and *Reference* manuals contain more information on *GuiDesigner* programs.

FileTextLoad

FileTextLoad creates a window to ask for the name of the text file you want to load. After you enter the name, the file is loaded and displayed the file in the main text area. The file must contain ASCII text, but need not be a program. Extended characters (128-255) may be retained and/or displayed in an unpredictable way, or not at all.

The file should not contain null characters (bytes = 0x00), as the file beyond the first null may not be displayed, and may even be lost.

In text mode, the general-purpose editor appropriate for notes, letters, reports, documentation, and text of just about any other kind.

FileLoad

FileLoad creates a window to ask for the filename of the program you want to load. After you enter it, the program development environment loads and displays the program in the main text area[4].

After programs are loaded, only the PROLOG is displayed. **ViewFunction** will display other functions.

FileSave

FileSave creates a window to ask for the filename you want to save the text or program as. Edit the filename in the little window if it's not already what you want, then select the **Save** button.

FileMode

FileMode creates a window to ask for the edit mode you want. You can choose *text* or *program* mode. If you want to edit anything other than an a program, select *text* mode. If you want to edit a program, it's usually more convenient to select *program* mode and view the program one function at a time. Sometimes you may prefer to edit programs as normal text. **FileMode** switches back and forth between text and program mode.

FileRename

FileRename creates a dialog window to ask for the new name you want to give the text or program currently loaded in the environment. Enter a valid filename to update the name of the file.

FileQuit

FileQuit ends the program development session. If the text or program file has been modified since it was last saved, a dialog window appears to warn you. You can then cancel **FileQuit** and save the file, or continue and permanently lose the contents of the main text window.

[4]The PDE loads the program, breaks it up into language elements (keywords, variables, operators, etc), converts each into a 32-bit token, and holds the program in this form. To display the program in the text area, the PDE converts tokens back into text form. Reproduction of the original text is generally quite accurate.

Edit *menu*

The **Edit** menu contains commands and actions for editing text. Click **Edit** in the menu bar, or press **Alt+e** on your keyboard. A menu with the following commands will appear:

Cut	**.ec**	**Cut selected text (clipboard).**
Grab	**.eg**	**Grab selected text (clipboard).**
Paste	**.ep**	**Paste buffer at cursor (clipboard).**
Delete	**.ed**	**Delete selected text.**
Buffer	**.eb**	**Buffer selected text.**
Insert	**.ei**	**Insert buffer at cursor.**
Erase	**.ee**	**Erase selected text.**
Find	**.ef**	**.ef = .f (find: see notes).**
		.r (replace: see notes).
Read	**.er [filename]**	**Read file into clipboard.**
Write	**.ew [filename]**	**Write selected clipboard.**
Abandon	**.ea**	**Abandon edits of function.**

You can execute any of these commands by selecting its name in the pull-down menu, or by typing the first letter of its name on your keyboard. Or, you can execute them directly by entering dot commands from the preceding table.

Selected Text

Many **Edit** menu commands operate on *selected text*. To select text, place the mouse cursor at the first character of the region you want to select, press the left button. Hold the button down while you move the mouse cursor past the last character in the region you want to select, then release the button. The selected area will be highlighted in some visible way. You can repeat the process as many times as you like until you select exactly the area you want. You can cancel selections by placing the mouse cursor anywhere in the text area and clicking the left button.

You can also select text from the keyboard by holding down a shift key while you move the text cursor with the cursor control keys.

Graphical user interfaces have quick ways to select important sections of text. Position the mouse cursor in a text area. Click twice to select a word, three times to select a line, and four times to select all text, including text above and below the visible area.

Many of the **Edit** menu commands expect a selected area. If no section of text is selected, these commands have no effect.

CutBuffer

The *CutBuffer* is a non-visible text storage area. A copy of selected text is copied into the CutBuffer whenever **EditCut** or **EditGrab** is executed. **EditPaste** copies the contents of CutBuffer into the text at the text cursor position.

The CutBuffer is the *system clipboard*, shared with all programs. Thus you can copy text between applications with the clipboard. Cut, Grab, Paste from/to the CutBuffer and system clipboard are also performed by the following popular keystrokes :

```
Delete  - Ctrl+X keys  - The Delete key
Buffer  - Ctrl+C keys  - The Crtl + Insert key
Insert  - Ctrl+V keys  - The Insert key
```

EditCut

EditCut copies the selected text into *CutBuffer*, then deletes it from the text area. The previous contents of *CutBuffer* are lost.

EditGrab

EditGrab copies the selected text into *CutBuffer*, but does not delete it from the text area. The previous contents of *CutBuffer* are lost.

EditPaste

EditPaste copies *CutBuffer* into the text area at the location of the text cursor. The contents of *CutBuffer* are not altered.

DeleteBuffer

The *DeleteBuffer* is a non-visible text storage area. A copy of selected text is copied into *DeleteBuffer* whenever **EditCut** or **EditBuffer** is executed. **EditInsert** copies the contents of *DeleteBuffer* into the text at the text cursor.

CutBuffer and *DeleteBuffer* are completely independent - operating on one has no effect on the other.

EditDelete

EditDelete copies the selected text into *DeleteBuffer*, replacing its previous contents, then deletes the text from the text area.

EditBuffer

EditBuffer copies the selected text into *DeleteBuffer*, replacing its previous contents. Text is not deleted from the text area.

EditInsert

EditInsert copies *DeleteBuffer* into the text area at the location of the text cursor. The contents of *DeleteBuffer* is not altered.

EditErase

EditErase deletes the selected text without making a copy. *CutBuffer* and *DeleteBuffer* are unchanged. Think twice before you **EditErase**. It may work alot like **EditCut** and **EditDelete**, but *the text* **EditErase** *deletes is irrevocably lost!*

Find and Replace

`Find` and **`Replace`** start at the text cursor position and search the main text area for a character string that matches *FindString*.

If a match is found, **`Find`** puts the text cursor just before the first character of the match, while **`Replace`** replaces the matched string with *ReplaceString* and leaves the cursor after *ReplaceString*.

FindString and ReplaceString

You can specify any strings you want for *FindString* and *ReplaceString*, but tab, newline, and all other control characters must be entered in *backslash format*. For example, **`"Hello\tmister\nbill"`** is a valid *FindString* or *ReplaceString* with a tab character between **`"Hello"`** and **`"mister"`**, and a newline between **`"mister"`** and **`"bill"`**.

Find and Replace Options

You can control the way **`Find`** and **`Replace`** behave, including case-sensitivity and repetition-count.

Case Sensitivity

To find matching strings, *case-sensitive* searches require identical strings, while *case-insensitive* searches consider upper and lower case versions of the same character as equivalent. A case-sensitive search for **`"Hello"`** will skip **`"hello"`**, **`"HELLO"`**, and **`"HellO"`**, while a case-insensitive search will find all three.

Searches are not case sensitive until you change the case sensitivity setting in the *FindWindow*, which appears when you select **`EditFind`** in the main menu, or enter **`.ef`** in the upper text area.

Repetition

The *repetition-count* determines how many times to execute find and replace commands. When repetition-count is **`1`**, **`Find`** stops at the next occurrence of *FindString*, while **`Replace`** stops after it replaces the next occurrence of *FindString* with *ReplaceString*.

When repetition-count is greater than one, **`Find`** searches for *FindString* the specified number of times. This has the same effect as searching for the **`n`**$^{\mathbf{th}}$ occurence of *FindString*, where **`n=`**repetition count.

When the repetition-count is greater than one, **`Replace`** searches for *FindString* the specified number of times, *each time* replacing it with *ReplaceString*. All **`n`** occurrences of *FindString* are replaced by *ReplaceString*, not just the **`n`**$^{\mathbf{th}}$ occurrence.

EditFind

To execute a find or replace, select **EditFind** in the main menu, or enter **.ef** in the upper text area.

The *FindWindow* will appear to display the current *FindString*, *ReplaceString*, case-sensitivity and repetition-count. Change any values you want, then click the **Find**, **Replace**, or **Cancel** button.

.f *and* .r

Often the most convenient way to execute find and replace is to enter **.f** and **.r** commands in the upper text area. These are particularly efficient because you can initiate complex finds and replaces with keystrokes alone, bypassing the mouse and window. The syntax of the **.f** and **.r** commands is:

.[#]f[-] [*find$***][<tab>** *replace$***]**
.[#]r[-] [*find$***][<tab>** *replace$***]**

find$ is the *FindString* - keeps its value between finds
replace$ is the *ReplaceString* - keeps its value between replaces
is the repetition count
portions in [square-brackets] are optional
a <tab> character separates *FindString* and *ReplaceString*
if no *find$* argument then use most recent *find$*
if no *replace$* argument then use most recent replace$

The following table lists some combinations this syntax supports:

.f	find next occurance of *find$*
.f-	find previous occurance of *find$*
.r	replace next occurance of *find$* with *replace$*
.f axe	*find$* = "axe" : find next *find$*
.r axe	*find$* = "axe" : replace next *find$* with *replace$*
.f axe dog	*find$* = "axe" : *replace$* = "dog" : find next *find$*
.r axe dog	*find$* = "axe" : *replace$* = "dog" : replace next *find$* with *replace$*
.3f	find 3rd occurence of *find$*
.6r	replace next 6 *find$* with *replace$*
.8f pig	*find$* = "pig" : find 8th *find$*
.5f pig cat	*find$* = "pig" : *replace$* = "cat" : find 5th *find$*
.7r dog cat	*find$* = "dog" : *replace$* = "cat" : replace next 7 *find$* with *replace$*
.*r pig dog	*find$* = "pig" : *replace$* = "dog" : replace every *find$* with *replace$*

A repetition count of ***** means *all occurences*, as illustrated by the comment for the final example.

EditRead

EditRead loads the file you specify from disk and puts it in buffer 0 aka the cut buffer aka the system clipboard. From there the file can then be pasted into any grid that accepts text from the system clipboard, even other applications.

EditWrite

EditWrite creates a disk file with the name you specify and saves the text contents of buffer 0 aka cut buffer aka system clipboard, into that file. If a disk file of the specified name already exists, its old contents are lost.

EditAbandon

EditAbandon abandons all changes made to the function currently displayed in the main text area since it most recently displayed.

View *menu*

Programs consist of a PROLOG and functions. The **View** menu contains commands to create new functions, delete existing ones, rename them, clone and view them. To select the view menu, click **View** in the menu bar, or type **Alt+v**. A menu with the following commands will appear.

```
Function        .vf [funcname]           View function
PriorFunction   .vp                      View prior function (last-viewed)
NewFunction     .vn [funcname]           Create new function
DeleteFunction  .vd [funcname]           Delete a function
RenameFunction  .vr [funcname]           Rename a function
CloneFunction   .vc [funcname]           Clone a function
LoadFunction    .vl [filename funcname]  Load function from disk
SaveFunction    .vs [filename funcname]  Save function to disk
                .v-                      Previous function in program
                .v                       Next function in program
```

ViewFunction

One function or the PROLOG is displayed in the lower text area at a time. **ViewFunction** switches to another function. If no argument is given, a window containing **PROLOG** and an alphabetized list of functions is displayed. Select an item from the list to display the function in the lower text area.

ViewPriorFunction

ViewPriorFunction is a quick way to display the previously displayed function in the main text area, even when you don't remember it's name. This command is especially convenient when you need to toggle back and forth between two functions.

ViewNewFunction

ViewNewFunction displays a window to ask you to name the new function. After you enter the name, a new function with a comment header and the first and last function lines is created and displayed.

ViewDeleteFunction

ViewDeleteFunction displays an alphabetized list of functions. Select the one you want to delete.

ViewRenameFunction

ViewRenameFunction displays a window to ask for the new name you want to give to the currently displayed function. After you enter the new name, all occurrences of the old function name in the program are changed to the new name. Function names in comments are not changed.

ViewLoadFunction

ViewLoadFunction loads a function from disk and adds it to the program. If the first line in the file is a function declaration line, that line is put in the PROLOG.

ViewSaveFunction

ViewSaveFunction saves a function declaration and function code to the specified disk file.

Option *menu*

The **Option** menu contains commands that let you observe and change compiler and development environment options. To select the option menu, click **Option** in the menu bar, or press **Alt+o** on your keyboard. A menu with the following commands will appear:

```
Misc             .om              Set miscellaneous options.
ColorTextCursor  .oc              Set color of text cursors.
TabWidth         .ot [pixels]     Set tab width value.
```

OptionMisc

OptionMisc displays a window containing compiler and PDE options, including *bounds checking*.

You can turn bounds checking on or off. Whenever a program is compiled with bounds checking **on**, extra machine instructions are generated to assure all array accesses are valid. When your program is running, bounds checking prevents improper array accesses and displays an error message. Bounds checking intercepts several common errors, including:

Attempt to access an element in an empty array.
Attempt to access beyond the upper bound of a dimension.
Attempt to access data at a node (a higher dimension).
Attempt to access a node at data (lowest dimension).

OptionMisc also lets you toggle:

the size of text in the PDE between normal and large
the size of text in the console between normal and large
the size of scroll bars in the PDE between normal and large
the size of scroll bars in the console between normal and large
the color of the console colors between normal and black & white

OptionColorTextCursor

When you select **OptionColorTextCursor**, a window appears and displays a text cursor and a set of colors you can click to change the color of the text cursor. Note that the text cursor is not the same color that you click, since the text cursor color is **XOR**ed with whatever it's drawn over. Often selecting medium or light blue will produce a pleasant and easily visible yellow cursor.

OptionTabWidth

By default, tab width in the upper and lower text areas are set every 2nd column, which is best for writing properly structured programs. But you can use the built-in editor to display and edit text of any kind, so you may want to change tab stops to every nth column. A column is usually 8 - 12 pixels.

When you select **OptionTabWidth**, a window appears and displays the current tab width value *in pixels*. To change the tab width value, change the value and press enter, or click the **Set** button.

Run *menu*

The **Run** menu contains commands for compiling and running programs. To select the run menu, click **Run** on the main menu bar or press **Alt+r** on your keyboard. A menu with the following commands will appear:

Start	**.rs**	**Start running program.**
Continue	**.rc**	**Continue running program.**
Pause	**.rp**	**Pause program execution.**
Kill	**.rc**	**Kill program execution.**
Recompile	**.rr**	**Recompile program.**
Assembly	**.ra**	**Create assembly file.**
Library	**.rl**	**Create library file.**

RunStart

RunStart begins program execution. If necessary, the program is recompiled first.

RunContinue

RunContinue resumes execution of the program, assuming it is ready to continue. Programs pause when they hit a breakpoint or have been explicitly paused by a *RunPause* command.

RunPause

RunPause suspends program execution. While your program is paused, you can display variables, function call frames, and other information about the current state of execution.

RunKill

RunKill terminates program execution. Variables, function call frames, and other status information is no longer available.

RunRecompile

RunRecompile compiles your program, but does not run it. You can use this command to check for syntax errors before your program is ready to run. If no errors are detected during compilation, you can also display the assembly language for the program with **DebugAssembly**.

RunAssembly

RunAssembly compiles your program into assembly language, and saves it on disk with a **.s** suffix. This file can subsequently be assembled and linked to create an application. Only with the professional edition can you create **.EXE** and **.DLL** files See the *Advanced Topics* manual for more information.

RunLibrary

RunLibrary compiles your program into assembly language, and saves it on disk with a **.s** suffix. This file can subsequently be assembled and linked to create a library. Only with the professional edition can you create **.EXE** and **.DLL** files. See the *Advanced Topics* manual for more information.

Debug *menu*

The **Debug** menu contains commands for controlling and monitoring program execution. To select the debug menu, click **Debug** in the menu bar, or press **Alt+d** on your keyboard. A menu with the following commands will appear:

```
ToggleBreakpoint       .dt     Toggle breakpoint.
ClearAllBreakpoints    .dc     Clear all breakpoints.
EraseLocalBreakpoints  .de     Erase func breakpoints.
Jump                   .dj     Jump PC aka IP to new location.
Memory                 .dm     Display memory.
Assembly               .da     Display assembly.
Registers              .dr     Display registers.
```

DebugToggleBreakpoint

DebugToggleBreakpoint toggles a breakpoint on or off at the beginning of the text cursor line. Program lines that start with **:** markers are breakpoint lines. You can toggle breakpoints at any time, even when your program is running.

DebugClearBreakpoints

DebugClearBreakpoints removes all breakpoints and **:** breakpoint markers in the program. You can clear breakpoints at any time, even when your program is running.

DebugEraseBreakpoints

DebugEraseBreakpoints removes all breakpoints and **:** breakpoint markers in the function displayed in the main text area. You can erase breakpoints at any time, even while the program runs.

DebugMemory

DebugMemory displays a window with a text area and text line. Enter **d starting-address length** in the TextLine grid to display memory in hexadecimal and ASCII. Enter **m** to print the valid address ranges. Enter **h** to get memory window help listing.

DebugAssembly

DebugAssembly displays a window containing assembly language for the current text cursor line, reconstructed from the binary machine instructions in memory. You can sequence through your program by clicking **Next** or **Prev**.

DebugRegisters

DebugRegisters displays a window listing the CPU registers and their contents. You can change the contents of any register by assigning a literal value in the one-line text area, as in **eax=0xDEADC0DE**.

Help *menu*

The **Help** menu contains commands for selecting information on various aspects of the program development environment. To select the help menu, click **Help** in the menu bar, or press **Alt+h** on your keyboard. A menu with the following commands will appear:

```
Notes              .hn   Notes about this release
Support            .hs   Technical support of this release
Message            .hm   Message information
Language           .hl   Keywords and Programming Language
Operator           .ho   Operator summary
DotCommand         .hd   Dot command summary
MathLibrary              Display xma.dec - Math/Trig/Log library
StandardLibrary          Display xst.dec - Standard library
GraphicsLibrary          Display xgr.dec - GraphicsDesigner library
GuiDesignerLibrary       Display xui.dec - GuiDesigner library
ComplexNumberLibrary     Display xcm.dec - Complex number library
```

HelpNotes

HelpNotes displays notes about this software release.

HelpSupport

HelpSupport displays information on how to get technical support.

HelpMessage

HelpMessage displays a list of predefined *GuiDesigner* messages.

HelpLanguage

HelpLanguage displays a list of programming language keywords.

HelpOperator

HelpOperator displays a list of programming language operators.

HelpDotCommand

HelpDotCommand displays a list of dot commands and syntax.

HelpMathLibrary

HelpMath displays **xma.dec**.

HelpStandardLibrary

HelpStandardLibrary displays **xst.dec**.

HelpGraphicsLibrary

HelpGraphicsDesigner displays **xgr.dec**.

HelpGuiDesignerLibrary

HelpGuiDesigner displays **xui.dec**.

HelpComplexNumberLibrary

HelpComplexNumber displays **xcm.dec**.

Conventional Programs

ahello.x

Before you write your own programs, load and run a small sample.

Select **File** in the main menu, then **Load** in the pulldown. Then type **ahello.x** in the dialog box that appears and press **enter**. The program will load for a couple seconds, then the beginning of its PROLOG will appear in the lower text area.

To run the program, click the **Start** hot button.

ahello.x prints **"Hello Programmer"** in the console window.

ahello.x contains the essential features of conventional style programs - a PROLOG with function declarations, plus at least one function containing executable code.

The PDE displays the PROLOG or one function at a time. To switch between functions, select from the **View** catagory in the main menu, or enter **.vf [func]**, **.vp**, **.v**, or **.v-** in the upper text area.

Your Own Conventional Program

You should now be able to write conventional programs yourself. Just remember, the console window is the focus of conventional input and output unless redirected to a disk file by **[filenumber].**

```
a$ = INLINE$(prompt$)          ' inputs line of text from console
a# = DOUBLE(a$)                ' convert input to floating point
a = XLONG(a$)                  ' convert input to integer number
PRINT a; a$, a#                ' prints line of text to console
ifile = OPEN(filename$, $$RD)  ' opens disk file for read
ofile = OPEN(file$, $$WR)      ' opens disk file for write
xfile = OPEN(file$, $$RW)      ' opens disk file for read & write
a$ = INFILE$(ifile)            ' reads line of text from disk file
PRINT [ofile], a; a$, a#       ' prints strings to disk file
READ [ifile], a, a$, a[]       ' reads binary data from disk file
WRITE [ofile], a; a$, a[]      ' writes binary data to disk file
```

The number of bytes read into variables by **READ** is the number of bytes in the variable before the **READ** is executed, so you'll probably need to size the string or array appropriately before executing the **READ** statement. For example:

```
a$ = NULL$(n)       ' prepare a$ to receive n bytes
DIM a[n]            ' prepare a[] to receive n+1 elements
```

The *Programming Language Guide and Reference* is a good place to look for detailed programming information. For quick reference, select **HelpLanguage** in the main menu.

Sample Programs

Next load & run the following sample programs for fun and practice:

```
adialog.x  - m - simple modal convenience functions
amodal.x   - m - convenience functions
ademo.x    - g - three windows at once
amath.x    - g - math library graphics
awindow.x  - g - main window simulated
aquick.x   - c - quick tour of grids & convenience funcs
aeditors.x - c - multiple editors / clipboards

             m - modal convenience function program
             g - GuiDesigner program (custom window)
             c - GuiDesigner convenience function program
```

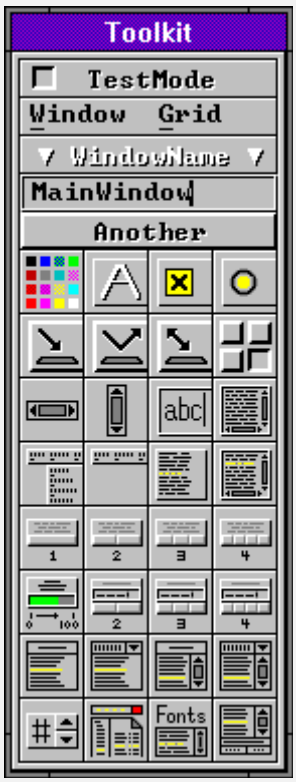
These are GUI programs - they communicate by means of a graphical user interface.

Except for some special purpose code in the **`...Code()`** function, these programs marked **`- g -`** were generated *entirely* by *GuiDesigner*. There's also one line of programmer written code in the **`Entry()`** function in **`acircle.x`** and **`ademo.x`** to show one way to write programs that need to run continuously but still respond to user events (button clicks, etc).

GuiDesigner Overview

Toolkit, Grids, Appearance Window

Toolkit Window



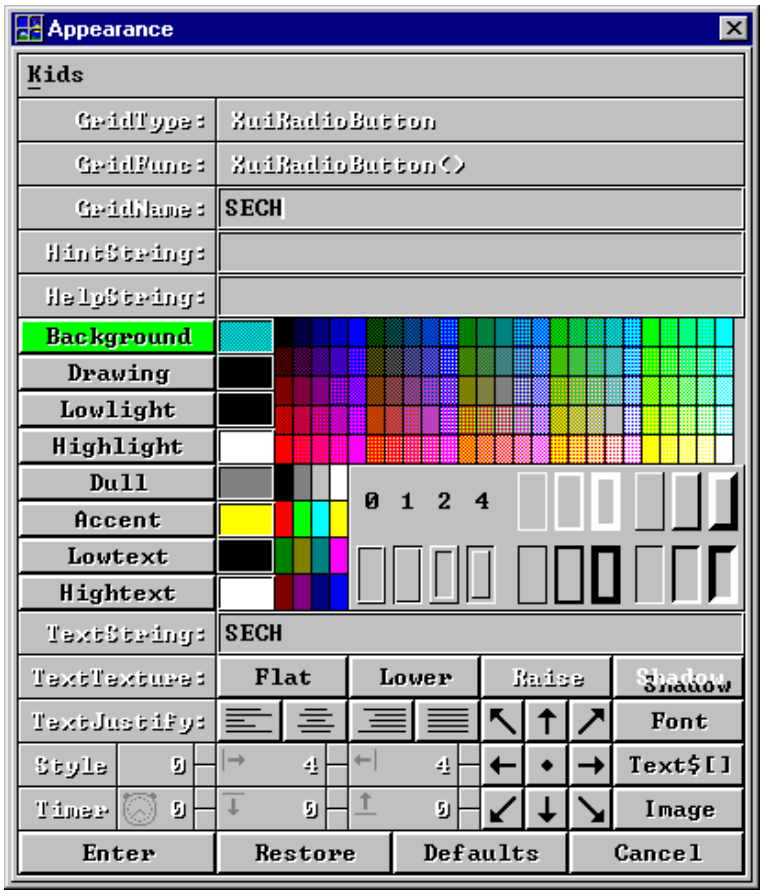
When you click the *toolkit* hot button, the *GuiDesigner* toolkit window appears in the lower left corner of the display screen.

Toolkit hot button

Selections in the toolkit menu let you create, delete, load, save, hide and display design windows, as well as convert design windows to functions and functions to design windows.

The square buttons on the tookit are "GUI components", which are commonly called "grids", "controls", and "widgets". Press a toolkit button to put the named grid in the visible design window, where it can be positioned, resized, and configured.

The appearance of any GUI component in the design window is controlled with the Appearance Window.



Appearance Window

Your First GuiDesigner Program

To create your first *GuiDesigner* program, perform these steps:

PS: If you have trouble, compare your program with `afirst.x` in your development directory.


`Toolkit` hot button


`PushButton` button

`Start` hot button

`Kill` hot button

- Select **`FileNew GuiProgram`** in the main menu bar.
  GuiDesigner creates a skeleton GUI program, some of which appears in the lower text area.
- Select the **`Toolkit`** hot button in the main window.
  A toolkit of GUI components (called grids) appears along the lower left side of the display.
- Select **`WindowNew`** in the toolkit menu bar.
  A design window appears in the upper right corner of the display screen.
- Enter "First" into the text line near the top of the toolkit.
  This names the design window. The name appears in the design window title bar.
- Click on the **`PushButton`** button in the toolkit.
  A double-box "grip" appears in the design window - the movable, resizable "selected grid".
  Only one grid can be selected at a time. Selecting one deselects any other.
  Click on bachground or another grid to deselect the selected grid and make it appear normally.
- Resize and move the grip in the design window (about 5 dots high and 10 dots wide).
  Move by dragging center of grip. Resize by dragging sides or corners of grip.
  Position it near the top center of the design window.
- Click on the **`PushButton`** button in the toolkit again to create another push button.
  Position it below your first push button.
- Click on your first push button to select it.
  The selected grid is always drawn as the resizable grip.
- Double click on the grip.
  The Appearance window appears next to the toolkit.
- Enter "First" in the TextString entry in the Appearance window and press <enter>.
  "First" appears on the first push button you created.
- Enter "FirstButton" in the GridName entry in the Appearance window and press <enter>.
  The button is named "FirstButton", but nothing visible changes.
- Click on medium-light colors until you find one you like.
  Try a medium-light green or cyan - top row, 11th or 16th color from left.
- Click on your second push button to select it.
  Your second push button is selected and turns into a grip.
- Double click on the grip.
  The contents of the Appearance window changes to reflect your second push button.
- Set color of your second push button, set TextString to "Second", set GridName to "SecondButton".
  "Second" appears in your second push button.
- Click the **`Cancel`** button in the Appearance window to make the Appearance window disappear.
- Select **`WindowToFunction`** in the toolkit menu bar.
  GuiDesigner adds **`First()`** and **`FirstCode()`** function declarations to your program.
  GuiDesigner converts the design window into two functions**,** **`First()`** and **`FirstCode().`**
  `First()` is a "grid function" that contains the code that creates and operates your window.
  `FirstCode()` is a "callback function" that responds to important events in your window.
  `FirstCode()` is visible in the lower text area of the main window.
  When **`FirstCode()`** gets callbacks, the first executable line reports the message and arguments.
- Select **`WindowDelete`** in the toolkit menu bar.
  Your design window is deleted and disappears.
- Click on the **`Start`** hot button.
  Your program parses, compiles, runs. Your window appears in the upper right portion of the display.
  The ReportMessage window appears in the upper left portion of the display (for testing only).
- Look in ReportMessage window as you click on your "First" and "Second" push buttons.
  A "Selection" callback message prints in the ReportMessage window each time you click a button.
  The **`kid`** argument should be 1 for your "FirstButton" button and 2 for your "SecondButton" button.
- Click the **`Kill`** hot button in the main window to terminate your program.
- Complete the following code in the **`SELECT CASE`** block in **`FirstCode()`**:

```
CASE $FirstButton  : XuiSendMessage (grid, #SetTextString, 0, 0, 0, 0, kid, "Hello")
                     XuiSendMessage (grid, #Redraw, 0, 0, 0, 0, kid, 0)
CASE $SecondButton : XuiSendMessage (grid, #SetTextString, 0, 0, 0, 0, kid, "World")
                     XuiSendMessage (grid, #Redraw, 0, 0, 0, 0, kid, 0)
```

- Select the **`Start`** hot button again and click the buttons again.
  The labels on your buttons will change the first time they're clicked.
  Hey, it works !!!

Congratulations. You've created a simple but complete *GuiDesigner* program that presents a GUI window and responds to user events!

What if you want to modify your window design? The next page tells how.

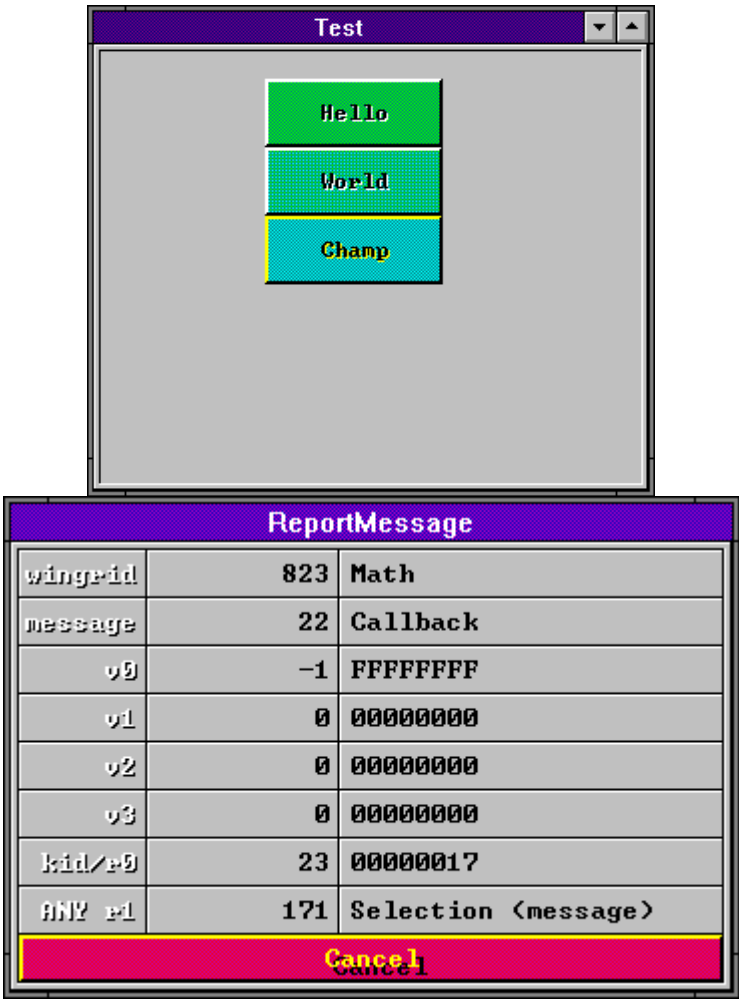
- Click the **Kill** button in the main window to stop your program.
  Your window disappears.
- Select **ViewFunction** in the main menu bar.
  A list of the functions in your program appears.
- Select function **First()** or **FirstCode()** from the function list.
  First() or **FirstCode()** is displayed in the lower text area of the main window.
- Click the *GuiDesigner* toolkit hot button near the right end of the horizontal row of hot buttons in the main window.
  The toolkit window reappears.
- Select **WindowFromFunction** in the toolkit menu bar.
  GuiDesigner converts the *code* in the displayed function **First()** into a new design window and displays it.
- Click on the **PushButton** button in the toolkit.
  A grip appears in the recreated design window.
- Position and resize the grip below your "Second" push button.
- Double click the grip to redisplay the Appearance window.
- Set the color of your third button, set GridName to "ThirdButton" set TextString to "Third".
- Click the **Cancel** button in the Appearance window.
  The Appearance window disappears.
- Select **WindowToFunction** in the toolkit menu.
  GuiDesigner converts the modified design window into a modified **First()** function.
  GuiDesigner asks whether you want to **Update** or **Replace** the existing **First()** and **FirstCode().**
- Click the **Update** buttons so the code you added to **FirstCode()** isn't changed.
  GuiDesigner updates your existing **First()** and **FirstCode()** functions to reflect your changes.
- Select **WindowDelete** in the toolkit menu.
  The modified design window is deleted and disappears.
- Add the following code to the **SELECT CASE** block at the bottom of **FirstCode().**

```
CASE $ThirdButton : XuiSendMessage (grid, #SetTextString, 0, 0, 0, 0, kid, "Champ")
                    XuiSendMessage (grid, #Redraw, 0, 0, 0, 0, kid, 0)
```

- Click the **Start** hot button in the main window.
  Your modified design window appears.
- Click your buttons and watch your program work.

Congratulations. You've now modified your first *GuiDesigner* program and made it work again.

Better yet, you're a *GuiDesigner* pro! That's because developing all GUI programs is essentially the same. And now, you've already been there.



afirst.x - *Your First GUI Design* *ReportMessage window*