Introduction to Amulet

What is Amulet?

Amulet Installation

Building the Amulet Libraries

Amulet Tutorial

Amulet and Visual C++ 6.0

Back End Communication

Application Architecture

Skeleton Program: Globals

main( )

Build the Window

Updating the Window

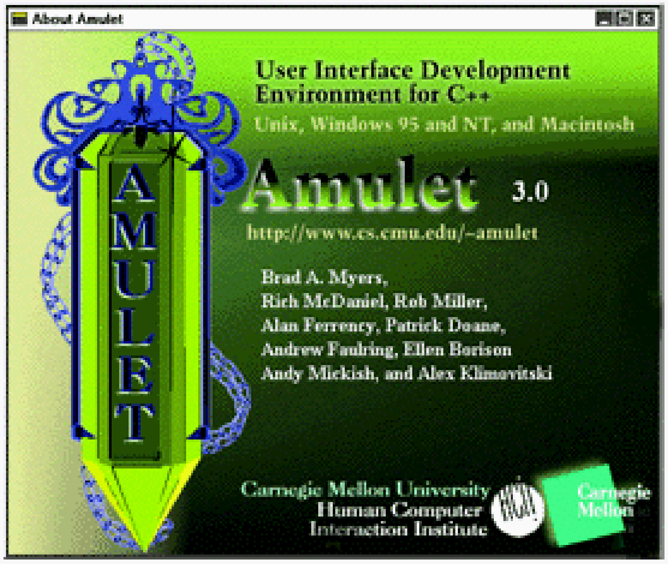
Creating the Menus

Translation Layer

P2 to Amulet Conversion

Text Input Dialog Box

Amulet GUI Example


About Amulet
User Interface Development Environment for C++
Unix, Windows 95 and NT, and Macintosh
AMULET
Amulet 3.0
http://www.cs.cmu.edu/~amulet
Brad A. Myers, Rich McDaniel, Rob Miller, Alan Ferrency, Patrick Doane, Andrew Faulring, Ellen Borison Andy Mickish, and Alex Klimovitski
Carnegie Mellon University Human Computer Interaction Institute
Carnegie Mellon

What is Amulet?

The Amulet (**A**utomatic **M**anufacture of **U**sable and **L**earnable **E**ditors and **T**oolkits) GUI is a cross-platform C++ library. Developed at the CS Dept at Carnegie-Mellon Univ. by the CMU HCI Institute team lead by Dr. Brad Myers, Senior Research Scientist.

http://www.cs.cmu.edu/~amulet/

Download the Amulet 3.0 distribution file and the Amulet 3.0 Reference Manual.

(**DO NOT** join the Amulet listservs – they do not exist to provide tech support for student projects.)

Amulet is cross-platform to the extent that pre-tested versions exist for a variety of Unix environments and for MS Windows.

Amulet is distributed free-of-charge in open source format.

When the grant ended in 1997, the continued maintenance and development of Amulet was assumed by the OpenAmulet group, lead by Robert Münch:

http://www.openip.org/

Note: we will be using Amulet 3.0 (from the CMU site) for this project.

Amulet Installation

Follow the instructions in section 1.4.3 of the reference manual for installing Amulet on a PC. Some further suggestions:

Decompressing the Amulet 3.0 zipped distribution archive (`amulet.zip`):

- unzip to a directory directly off the root of a drive (`d:\amulet`)
- select the "Use Folder Names" option under WinZip
- DO NOT install to a path that contains spaces; in particular, do not select `C:\Program Files\amulet`.
- requires about 100MB of space for a complete installation and subsequent build.

Setting system environment variables:

- read Section 1.4.3.2 of the Amulet 3.0 Reference Manual (but be sure to follow the directions below).
- add the environment variable `AMULET_DIR` and set its value to be the path to the root of your Amulet installation (assume `d:\amulet` for the rest of these notes).
- add the environment variable `AMULET_VARS_FILE` and set its value to "`Makefile.vars.MSVC5.Win32`" (for either Visual 5.0 or 6.0).

Note: you may have to reboot in order for the settings changes to take effect.

Building the Amulet Libraries

Before using Amulet, you must build the library files that will be needed when linking your project:

- start Visual C++, select File/Open Workspace, and navigate to the bin directory in your Amulet installation (say, d:\amulet\bin).
- select the filter Makefiles (.mak) in the Open dialog box.
- select the file amulet.mak (probably the only one listed).
- select Build/Build All and wait while Visual C++ builds the Amulet libraries.

The library and support programs are rather extensive and will take some time to compile and link.

Because portions of the Amulet implementation are somewhat dated, from the perspective of Standard C++, it is normal to get a number of warning messages during the library build, and during your own builds later. You can safely ignore them.

If the build fails, go back through the steps outlined on the previous slide and above, and determine which one you didn't carry out.

Amulet Tutorial

Read and **work** your way through Chapter 2, "Amulet Tutorial" in the Amulet 3.0 Reference Manual.

- a sample Amulet project is located in the Tutorial directory: `d:\amulet\samples\tutorial`
- copy the directory to your working directory and open the project (`.dsp`) file in Visual; if Visual asks to convert the project file say yes.

Initially, Visual will not be able to find certain necessary header and library files, since those locations are installation dependent.

- from the Tools menu in Visual, select Options and then choose the Directories tab.
- in the Show directories menu, choose Include files, and add the path to the Amulet include directory: `d:\amulet\include`
- in the Show directories menu, choose Library files, and add the path to the Amulet libraries: `d:\amulet\lib`

Now you should be able to build the first Tutorial project. If not, go back and determine which step you didn't carry out.

Again: **work your way through the Amulet Tutorial**. This DOES NOT mean "read the tutorial".

Amulet and Visual C++ 6.0

To create your own Amulet project you'll need to first create an appropriate project file that Visual C++ recognizes.

Low road:

- copy the default `d:\amulet\samples\tutorial\` project file (.dsp) into your own working directory.
- rename all of the tutorial files and tutorial folder to your project name.
- select File/New and in the New dialog, choose the Project tab and select the Add to current workspace radio button. Select the Win32 Application and then enter your own project name.
- delete the tutorial project files from the workspace.
- select Project/Settings and select the Link tab. Change output file name exe to whatever you wish to call it. Add "`amulet.lib`" and "`winmm.lib`" to the end of the Object/library modules list.

Alternatively:

- create a new project workspace of type Win32 Application
- select Project/Settings and select the Link tab. Change output file name exe to whatever you wish to call it. Add "`amulet.lib`" and "`winmm.lib`" to the end of the Object/library modules list.

Amulet – Back End Communication

The design of your project should provide both a logical and a physical code separation of the Amulet GUI code, which handles the interaction with the user from the legacy code from your linked-list program, which manages all the database operations.

Annoyance: the data display elements in the Amulet library are designed to display C-style (null-terminated) text strings.

Of course, your code should be designed to use appropriate data types for each data field, so your program sees a mixture of string objects, ints, doubles, etc. Not only that, but most of P2 should have been designed to pass and receive higher-level objects.

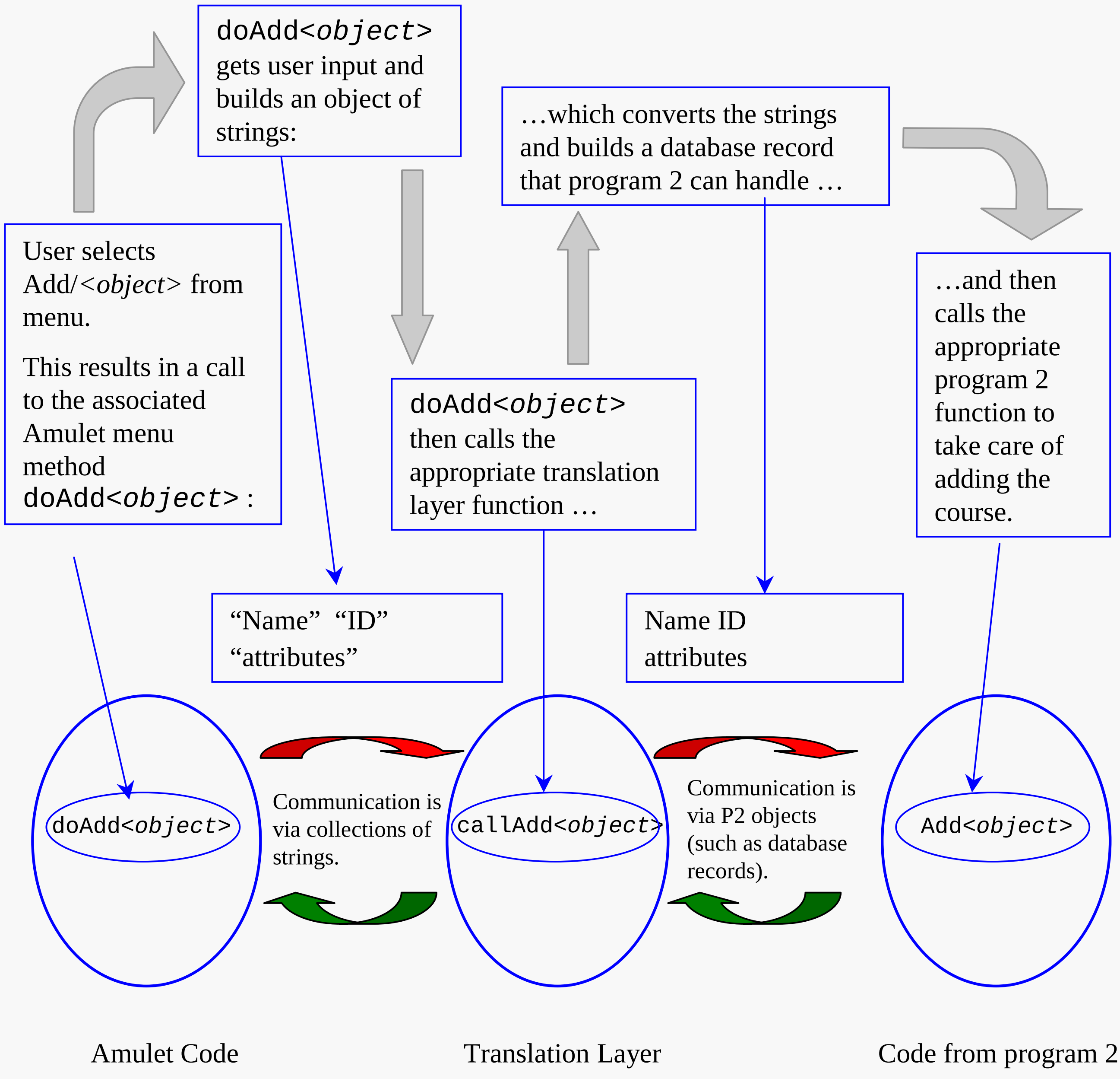
That means that a translation must take place during the communication between the P2 back end and the Amulet front end.

The <u>required</u> way to handle that is to provide an intermediate layer of functions that perform two logical tasks:

- handle the translation of data
- handle mapping the interface actions selected by the user to the appropriate back end functions recycled from P2.

Application Architecture

Your implementation will provide an intermediate layer, the Translation Layer, which will serve as a logical interface between the Amulet GUI and the back end linked-list program functions:


doAdd<*object*> gets user input and builds an object of strings:
…which converts the strings and builds a database record that program 2 can handle …
User selects Add/<*object*> from menu.
This results in a call to the associated Amulet menu method doAdd<*object*> :
doAdd<*object*> then calls the appropriate translation layer function …
…and then calls the appropriate program 2 function to take care of adding the course.
"Name" "ID" "attributes"
Name ID attributes
doAdd<*object*>
Communication is via collections of strings.
callAdd<*object*>
Communication is via P2 objects (such as database records).
Add<*object*>
Amulet Code
Translation Layer
Code from program 2

Skeleton Program: Globals

```
// P2main.cpp – main function and Amulet startup code
#pragma warning (disable:4800)
#include "console.h"
#include "myWindow.h"       // slot declarations for GUI
#include "MultiDialog.h"    // (projected)

#include "TL.h"             // interface to P2 functions
#include <fstream>
#include <iomanip>
using namespace std;

Am_Object my_win;       //define window to be global so callback
                        //       functions have access

// functions for GUI management
void InitWindow(Am_Object& my_win);  // set up window
void InitMenus (Am_Object&    my_menu_bar); // and menu bar
void updateWindow(Am_Object& my_win, amStudent myDisplay);

// Define callback functions for menu options.
// These odd-looking "functions" are how you tie code to the
// selection of a menu option in the GUI.
```

basic (generic) object type in Amulet

Amulet "function" that "hooks" a function implementation to a menu item (or other graphical object).

This is a method for an object.

Method is void and is named "OpenDo".

```
Am_Define_Method(Am_Object_Method, void, OpenDo,
                                         (Am_Object self)) {

  // get file name, read data, etc., see later slide for details
  updateWindow(my_win, myDisplay); // update window contents
}

// . . . additional callback functions omitted . . .
```

main( )

```
int main( ) {

   Am_Initialize();           // Initialize Amulet
```

This "turns on" the basic Amulet system. Don't worry about what it does, just include it.

```
   InitWindow(my_win);        // Initialize window
```

This calls a function (that you write) that sets up some of the GUI window elements you want to use.

```
   Am_Object my_menu_bar;     // declare menu bar object
   InitMenus(my_menu_bar);    // Initialize menu bar
```

This creates the menu bar and menu lists that you want to use. Again, you write this function.

```
   my_win.Add_Part(my_menu_bar);  // add menu bar to window
   Am_Screen.Add_Part(my_win);    // add window to screen
```

These statements add the menu bar to the window and add the widow to the screen (I.e., they form associations between the objects).

```
   Am_Main_Event_Loop();       // initiate Amulet GUI loop
```

This starts the execution of a loop that watches for user input (mouse clicks, etc.) and takes care of notifying the appropriate object methods that were defined before main( ). The loop is provided by the Amulet system --- you do not write it.

```
   Am_Cleanup();               // destruct objects on exit

   return 0;
}
```

Build the Window

```
void InitWindow(Am_Object& my_win) {


    my_win = Am_Window.Create ("my_win") //Create an Amulet Window
          .Set (Am_LEFT,    20)          // set left edge coord
          .Set (Am_TOP,     50)          //     top edge coord
          .Set (Am_WIDTH,  400)          //     width
          .Set (Am_HEIGHT, 200)          //     height
          .Set (Am_TITLE, "W A R P Prototype") // title bar

      .Add_Part(SSNLabel, Am_Text.Create("SSNLabel")
          .Set(Am_LEFT,   10)
          .Set(Am_TOP,    50)
          .Set(Am_WIDTH,  40)
          .Set(Am_HEIGHT, 14)
          .Set(Am_TEXT, "SSN")
          .Set(Am_LINE_STYLE, Am_Blue)
          .Set(Am_FILL_STYLE, Am_No_Style)
      )

      .Add_Part(SSNField, Am_Text.Create("SSNField")
          .Set(Am_LEFT,   60)
          .Set(Am_TOP,    50)
          .Set(Am_WIDTH,  80)
          .Set(Am_HEIGHT, 14)
          .Set(Am_TEXT, "000000000")
          .Set(Am_LINE_STYLE, Am_Black)
          .Set(Am_FILL_STYLE, Am_No_Style)
      )

// . . . code omitted
     .Add_Part(HoursField, Am_Text.Create("HoursField")
          .Set(Am_LEFT,   70)
          .Set(Am_TOP,   140)
          .Set(Am_WIDTH,  60)
          .Set(Am_HEIGHT, 14)
          .Set(Am_TEXT, "0.00")
          .Set(Am_LINE_STYLE, Am_Black)
          .Set(Am_FILL_STYLE, Am_No_Style)
      ); // end of Add_Part and Am_Window declaration
}
```

"Constructor" returns an Am_Window object.

This adds a text field to the window, setting the position on the screen, the initial text to be displayed, and the appearance of that text.

The text field is named SSNLabel.

Coordinates are expressed in pixels. Think of the NW corner of the screen/window as being the origin of a coordinate system, with the x-axis running across the top and the y-axis running down the left side.

Updating the Window

```
// Updates the relevant fields of the window object:
//
void updateWindow(Am_Object& my_win, amStudent myDisplay) {
```

myDisplay holds values for a record, which are to be displayed in the data fields defined earlier for the Amulet window.

The idea is that your Amulet code will call a function that will interact with one or more of your existing P2 functions to retrieve data, and then build a data capsule, myDisplay, which it will return to the Amulet side. The code below takes care of "posting" it:

```
 my_win.Get_Object(SSNField).Set(Am_TEXT, myDisplay.SSN.c_str());
```

Get_Object( ) returns the named object that's part of my_win.

Set( ) stores the specified value (second parameter) into that object. Am_TEXT specifies the type of value that Set( ) will store so the proper conversion can be made.

```
 my_win.Get_Object(NameField).Set(Am_TEXT,
                                        myDisplay.Name.c_str());
  my_win.Get_Object(MajorField).Set(Am_TEXT,
                                        myDisplay.Major.c_str());
  my_win.Get_Object(MinorField).Set(Am_TEXT,
                                        myDisplay.Minor.c_str());
  my_win.Get_Object(QCAField).Set(Am_TEXT,
                                        myDisplay.QCA.c_str());
  my_win.Get_Object(AltQCAField).Set(Am_TEXT,
                                        myDisplay.AltQCA.c_str());
  my_win.Get_Object(HoursField).Set(Am_TEXT,
                                        myDisplay.Hours.c_str());
}
```

Creating the Menus

```
// Creates and returns an Am_Menu_Bar object which will be added
// to the Amulet window.
//
void InitMenus(Am_Object& my_menu_bar) {


  my_menu_bar = Am_Menu_Bar.Create("my_menu_bar")
    .Set(Am_FILL_STYLE, Am_Motif_Gray)            // set bar color
    .Set(Am_ITEMS, Am_Value_List()            // set list of menus

    .Add(Am_Command.Create("FileMenu")
       .Set(Am_LABEL, "File")
       .Set(Am_IMPLEMENTATION_PARENT, true)

       .Set(Am_ITEMS, Am_Value_List()

         .Add(Am_Command.Create("OpenDo")
             .Set(Am_LABEL, "Open")
             .Set(Am_ACTIVE, true)
             .Set(Am_DO_METHOD, OpenDo))

         .Add(Am_Command.Create("SaveAsDo")
             .Set(Am_LABEL, "Save As")
             .Set(Am_ACTIVE, false)
             .Set(Am_DO_METHOD, SaveAsDo))

         .Add(Am_Menu_Line_Command.Create("my menu line"))

         .Add(Am_Quit_No_Ask_Command.Create() )

         ) // end Set Am_ITEMS

       )// end Add FileMenu

// continued . . .
```

"Constructor" returns an Am_Menu_Bar object.

Create a menu named FileMenu.

Label it “File”.

Add menu items to the FileMenu object

Add OpenDo menu item, label it “Open”, make it active, and tie it to the method named “OpenDo”.

Add a separator to the menu list

Add default Quit command to the menu list.

Creating the Menus

```
// . . . continued

    .Add (Am_Command.Create("CoursesMenu")     // all done
         .Set(Am_LABEL, "Courses")
         .Set(Am_IMPLEMENTATION_PARENT, true)
         .Set(Am_ITEMS, Am_Value_List ()
    .Add (Am_Command.Create("DisplayCoursesDo")
         .Set(Am_LABEL, "Display")
         .Set(Am_ACTIVE, true)
         //.Set(Am_ACCELERATOR, "^s")
         .Set(Am_DO_METHOD, DisplayCoursesDo))
    .Add (Am_Command.Create("AddCourseDo")
         .Set(Am_LABEL, "Add")
         .Set(Am_ACTIVE, false)
         //.Set(Am_ACCELERATOR, "^m")
         .Set(Am_DO_METHOD, AddCoursesDo))
                )
// . . . code for other menus omitted here . . .

  );  // end of my_menu_bar declaration

} // InitMenus
```

This menu item is active, which means that the user can select it.

This menu item is not active, which means that the user cannot select it, although it is still visible.

Amulet Slot Definitions

```
// myWindow.h
#ifndef MYWINDOW_H
#define MYWINDOW_H

#include "camulet.h"
//#include "TL.h"

//Define slot/part field label objects and register with amulet
Am_Slot_Key SSNLabel    = Am_Register_Slot_Name ("SSNLabel");
Am_Slot_Key NameLabel   = Am_Register_Slot_Name ("NameLabel");
Am_Slot_Key MajorLabel  = Am_Register_Slot_Name ("MajorLabel");
Am_Slot_Key MinorLabel  = Am_Register_Slot_Name ("MinorLabel");
Am_Slot_Key QCALabel    = Am_Register_Slot_Name ("QCALabel");
Am_Slot_Key AltQCALabel = Am_Register_Slot_Name ("AltQCALabel");
Am_Slot_Key HoursLabel  = Am_Register_Slot_Name ("HoursLabel");

//Define slot/part field objects and register with amulet
Am_Slot_Key SSNField    = Am_Register_Slot_Name ("SSNField");
Am_Slot_Key NameField   = Am_Register_Slot_Name ("NameField");
Am_Slot_Key MajorField  = Am_Register_Slot_Name ("MajorField");
Am_Slot_Key MinorField  = Am_Register_Slot_Name ("MinorField");
Am_Slot_Key QCAField    = Am_Register_Slot_Name ("QCAField");
Am_Slot_Key AltQCAField = Am_Register_Slot_Name ("AltQCAField");
Am_Slot_Key HoursField  = Am_Register_Slot_Name ("HoursField");

//Am_Slot_Key OverView    = Am_Register_Slot_Name ("OverView");


#endif
```

These slot/parts are added to my_win window in InitWindow().

The text (value) of these field parts are set with the predefined Am_TEXT slot in InitWindow().

Translation Layer

```
// TL.cpp
#include “Larp.h”          // basic LARP types
#include “TL.h”            // prototypes for translation layer
#include “LinkList.h”      // LinkList declarations

// . . . other header inclusions as needed . . .

#include <sstream>         // for string streams

LinkList LL;               // link list for P2 database records
```

File-scoped database list is not ideal…

```
  struct amStudent {
     string SSN,
            Name,
            Major,
            Minor,
            QCA,
            AltQCA,
            Hours;
  };
```

This goes in TL.h

```
amStudent callReadStudent(string iFileName) {

   ifstream In(iFileName.c_str());
   if (In.fail()) {
      Student S;
      return toAmulet(S);
   }

   Student Next(In);
   In.close();

   LL.InsertByID(Next);
   return toAmulet(LL.getCurrentData());
}

// . . . other translation layer functions follow . . .
```

Return the Student record to be displayed to the Amulet side.

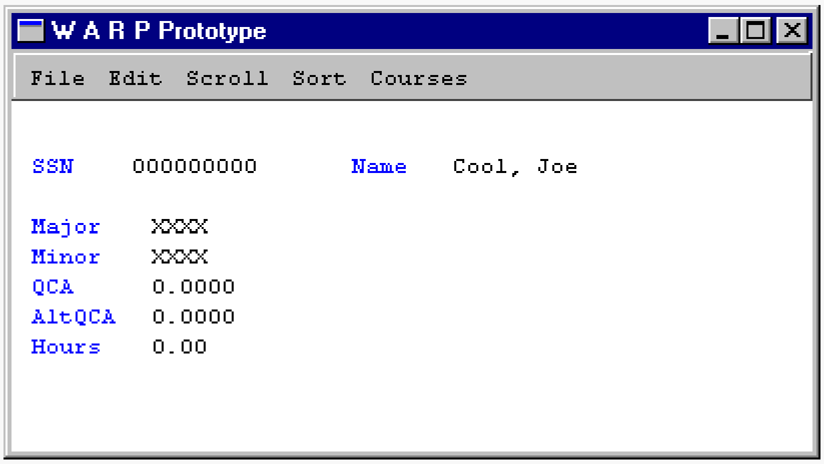
LARP to Amulet Conversion

```
amStudent toAmulet(Student toPack) {

   amStudent AmRecord;

   AmRecord.SSN   = toPack.getID();   // build AmRecord for return
   AmRecord.Name  = toPack.getName();
   AmRecord.Major = toPack.getMajor();
   AmRecord.Minor = toPack.getMinor();

   ostringstream oQCA("");
   oQCA.setf(ios::fixed, ios::floatfield);
   oQCA.setf(ios::showpoint);
   oQCA << setprecision(4) << toPack.getQCA();

   ostringstream oAltQCA("");
   oAltQCA.setf(ios::fixed, ios::floatfield);
   oAltQCA.setf(ios::showpoint);
   oAltQCA << setprecision(4) << toPack.getAltQCA();

   ostringstream oHours("");
   oHours.setf(ios::fixed, ios::floatfield);
   oHours.setf(ios::showpoint);
   oHours << setprecision(2) << toPack.getHours();

   AmRecord.QCA    = oQCA.str();
   AmRecord.AltQCA = oAltQCA.str();
   AmRecord.Hours  = oHours.str();

   return AmRecord;
}
```

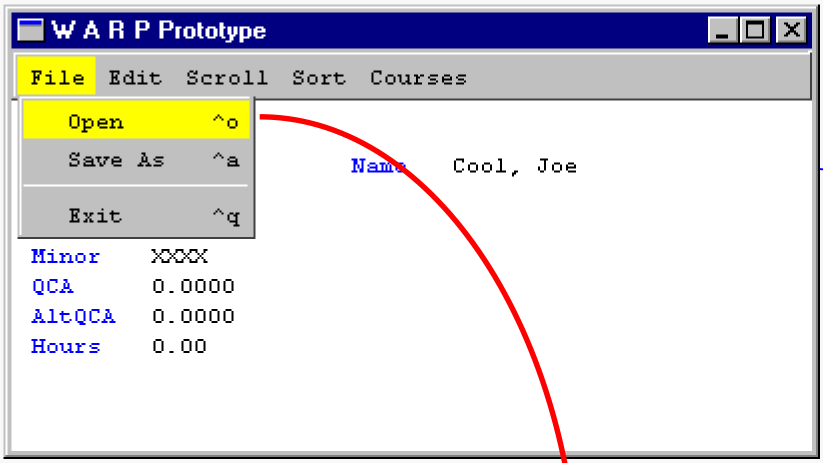
Text Input Dialog Box

```
// Menu method for File/Open choice:
//
Am_Define_Method(Am_Object_Method, void, OpenDo, (Am_Object self))
{

  // Set up dialog box prompts
  Am_Value_List prompts ;
  prompts.Add("Please enter the name of the saved database file.")
      .Add("Enter the full path name, if the file is NOT . . .")
      .Add("(Be sure to enter the file extension)");

  Am_String InFileName ; //define return value from input dialog

  int xcoor = Am_AT_CENTER_SCREEN, ycoor = Am_AT_CENTER_SCREEN ;
                       //input dialog location

  // Call Amulet input dialog box - see Amulet Manual pg. 289 –
  // dialog is Modal
  InFileName = Am_Get_Input_From_Dialog(prompts, "warp.db",
                                        xcoor, ycoor, true);

  string iFileName;

  if (InFileName == Am_No_Value) {  //cancel button pressed
     iFileName = "warp.db";
  }
  else {
     // Convert entered Amulet string to string object
     char* fstr = InFileName ;
     iFileName = string(fstr);
  }

  amStudent myDisplay = callOpenDB(iFileName);
  updateWindow(my_win, myDisplay);
}
```

WARP Prototype Example



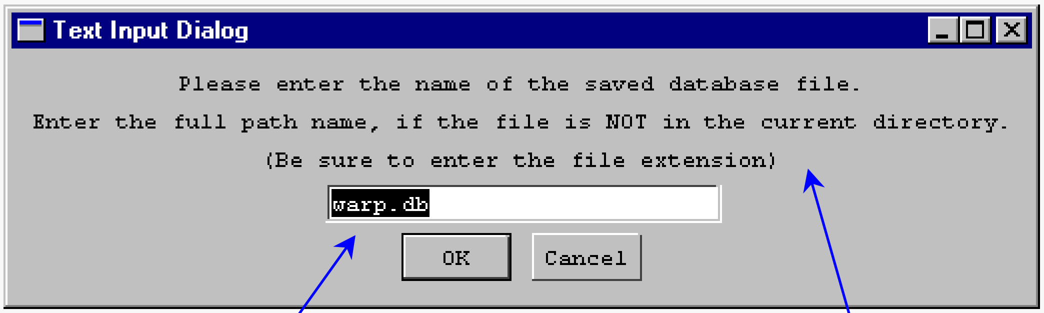
```
SSN      000000000        Name     Cool, Joe

Major     XXXX
Minor     XXXX
QCA       0.0000
AltQCA    0.0000
Hours     0.00
```



```
                          Name     Cool, Joe

Minor     XXXX
QCA       0.0000
AltQCA    0.0000
Hours     0.00
```

Selected menu item is highlighted…

Popup dialog box gets file name from user…



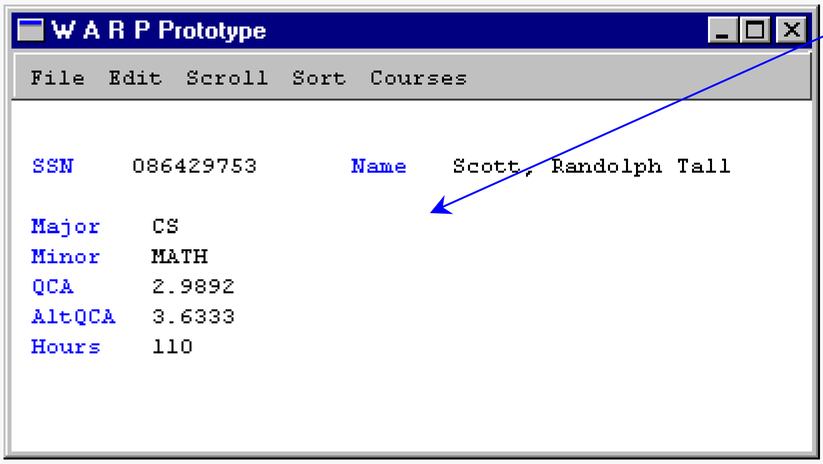
Please enter the name of the saved database file.
Enter the full path name, if the file is NOT in the current directory.
(Be sure to enter the file extension)

warp.db

Default file name provided.

Descriptive text describes required action.

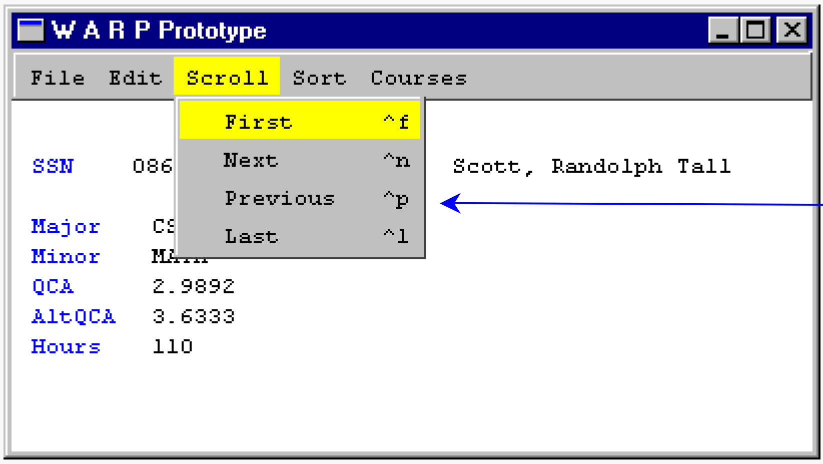
More WARP Prototype



```
SSN      086429753       Name    Scott, Randolph Tall

Major     CS
Minor     MATH
QCA       2.9892
AltQCA    3.6333
Hours     110
```

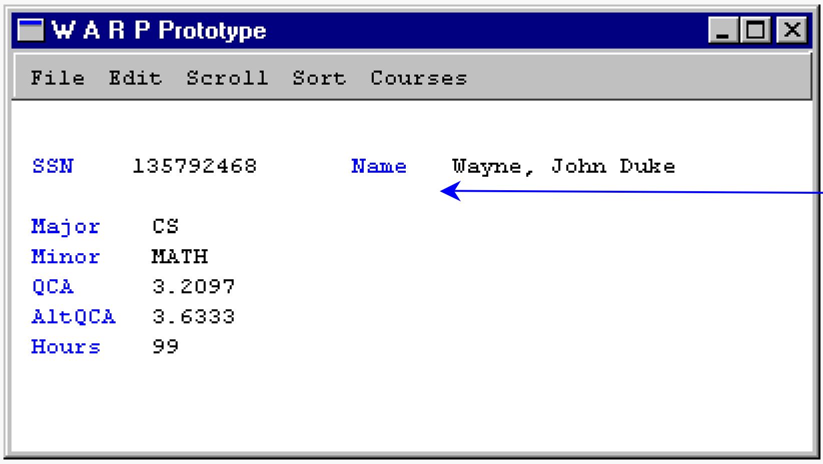
When the user selects OK, the File/Open method then calls the appropriate P2 interface function to cause the specified file to be read and an inventory database list to be created.

Window fields are then updated and displayed…



```
SSN      086                     Scott, Randolph Tall

Major     CS
Minor     MA
QCA       2.9892
AltQCA    3.6333
Hours     110
```

Accelerator keys are available (keyboard alternatives to mouse actions).

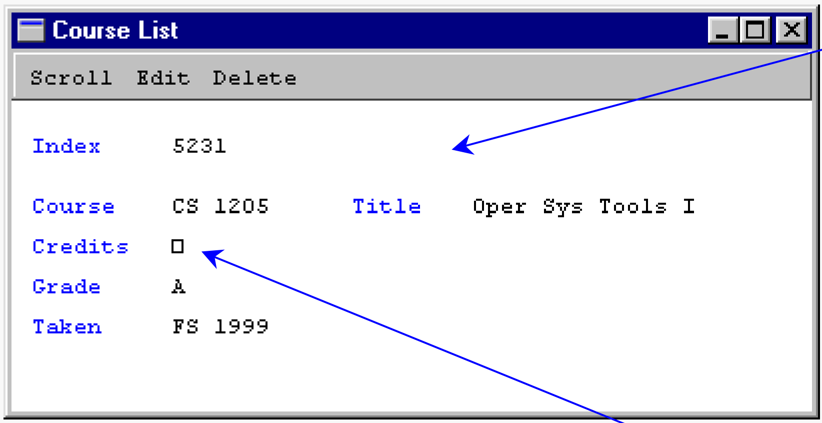


```
SSN      135792468       Name    Wayne, John Duke

Major     CS
Minor     MATH
QCA       3.2097
AltQCA    3.6333
Hours     99
```

Result of scrolling through the database…

More WARP Prototype

```
Am_Define_Method(Am_Object_Method, void, DisplayCoursesDo,
                                            (Am_Object self)) {
. . .
   Am_Value ok;  // return value from window
   Am_Pop_Up_Window_And_Wait(CourseDisplayWindow, ok, true);
. . .
}
```



```
Index      5231

Course     CS 1205       Title     Oper Sys Tools I
Credits    □
Grade      A
Taken      FS 1999
```

This popup window is displayed when the user chooses Course/Display.

The structure is very similar to the main window.

Evidently the value for this field is not yet being converted properly to a C-style string.

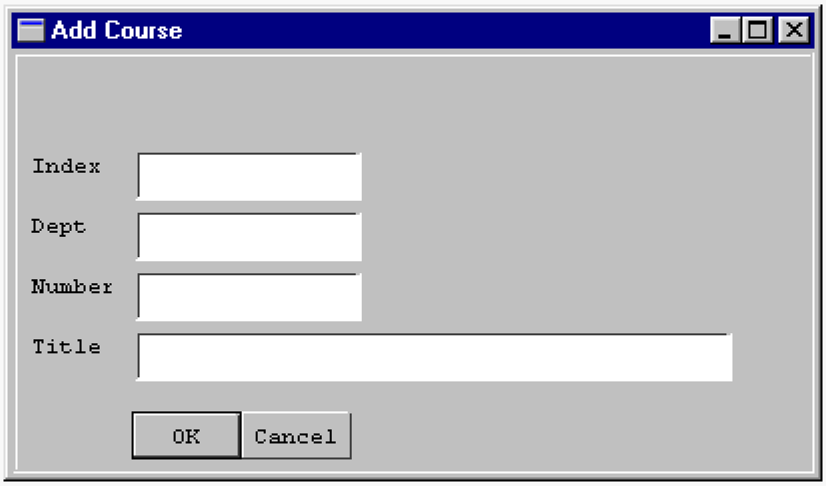
Note that the window shown does not include the specified Scroll menu.

More WARP Prototype

The (incomplete) pseudo-dialogbox shown below is popped up when the user chooses Course/Add.

Amulet does not provide a default widget for creating a dialog box with multiple input fields. I solved the problem by creating a window with the look of a dialog box and adding the appropriate fields.

When the user presses "OK", I can read the entered values from the text fields (`Am_Text_Input_Widget` objects) and assemble an amStudent variable for return to the translation layer.

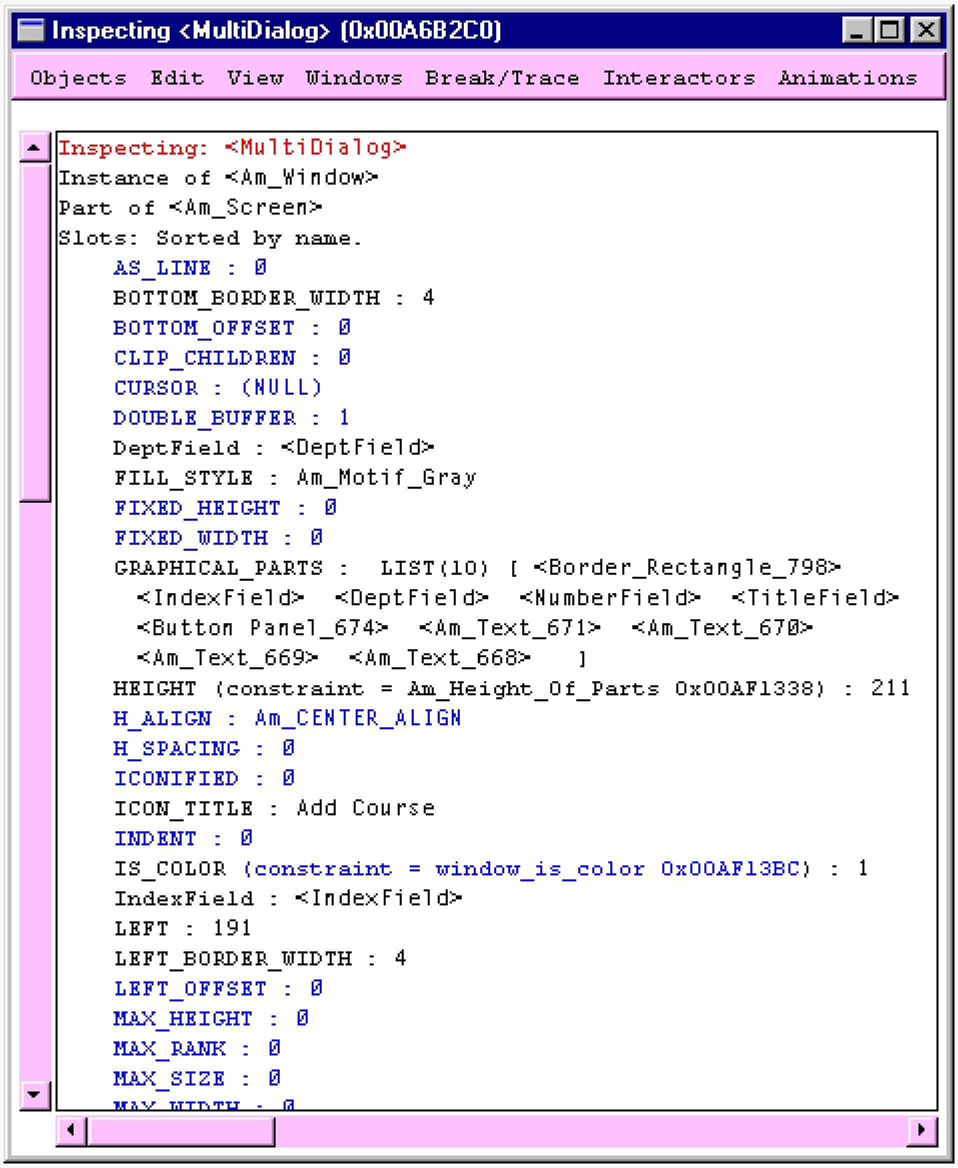


The initial layout of this window was done with GILT, the Amulet tool for building simple interfaces quickly… see slide 24 and following.

Amulet Object Inspector

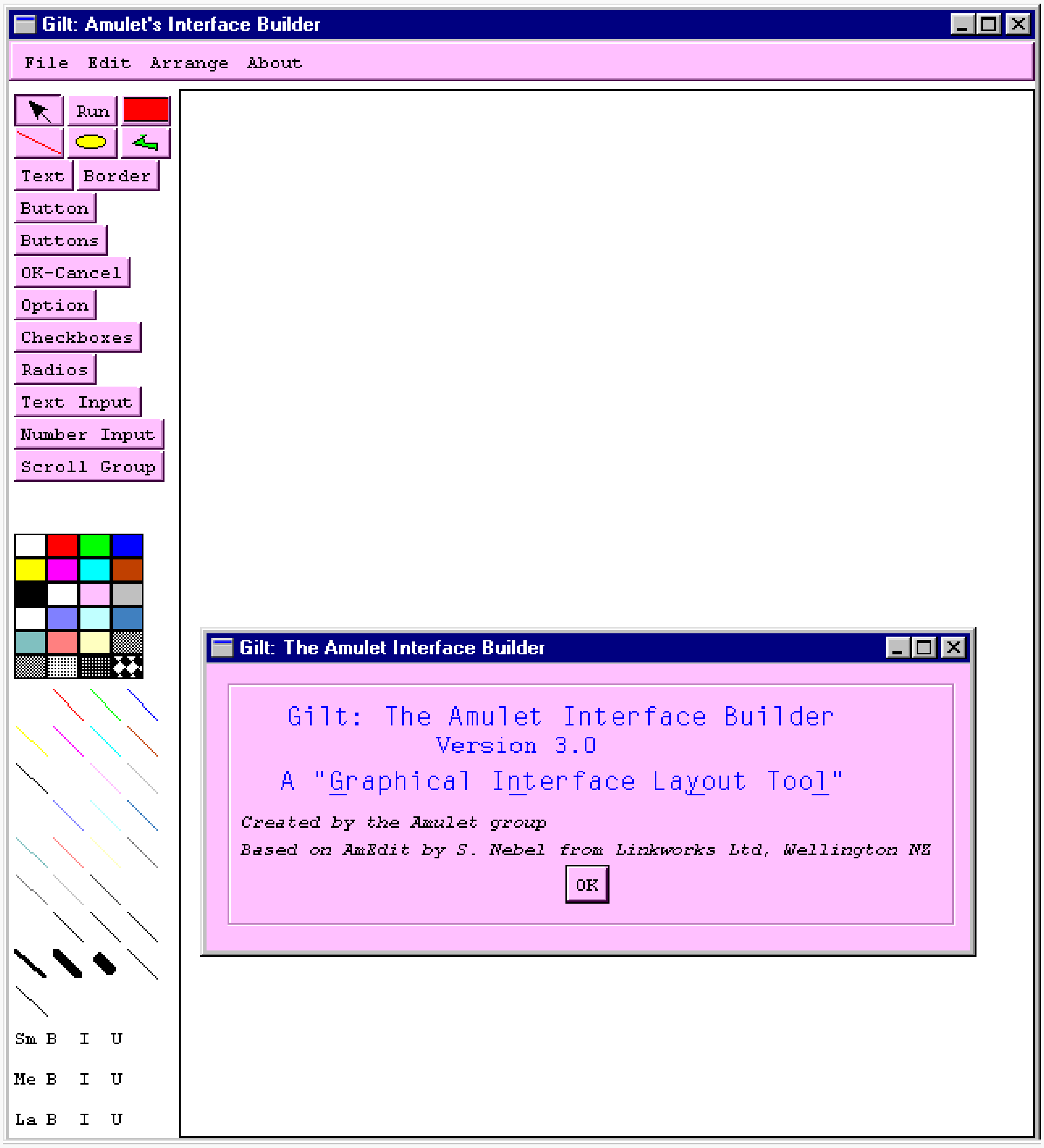
The Amulet Object Inspector Window can be opened by placing the mouse cursor on an Amulet object and pressing the F1 key.

This can provide quite a bit of useful information about Amulet objects.


Inspecting <MultiDialog> (0x00A6B2C0)
Objects Edit View Windows Break/Trace Interactors Animations

```
Inspecting: <MultiDialog>
Instance of <Am_Window>
Part of <Am_Screen>
Slots: Sorted by name.
    AS_LINE : 0
    BOTTOM_BORDER_WIDTH : 4
    BOTTOM_OFFSET : 0
    CLIP_CHILDREN : 0
    CURSOR : (NULL)
    DOUBLE_BUFFER : 1
    DeptField : <DeptField>
    FILL_STYLE : Am_Motif_Gray
    FIXED_HEIGHT : 0
    FIXED_WIDTH : 0
    GRAPHICAL_PARTS :  LIST(10) [ <Border_Rectangle_798>
      <IndexField>  <DeptField>  <NumberField>  <TitleField>
      <Button Panel_674>  <Am_Text_671>  <Am_Text_670>
      <Am_Text_669>  <Am_Text_668>    ]
    HEIGHT (constraint = Am_Height_Of_Parts 0x00AF1338) : 211
    H_ALIGN : Am_CENTER_ALIGN
    H_SPACING : 0
    ICONIFIED : 0
    ICON_TITLE : Add Course
    INDENT : 0
    IS_COLOR (constraint = window_is_color 0x00AF13BC) : 1
    IndexField : <IndexField>
    LEFT : 191
    LEFT_BORDER_WIDTH : 4
    LEFT_OFFSET : 0
    MAX_HEIGHT : 0
    MAX_RANK : 0
    MAX_SIZE : 0
```

The executable for GILT is located in the `bin` directory of the Amulet tree.



Gilt: The Amulet Interface Builder
Version 3.0
A "Graphical Interface Layout Tool"

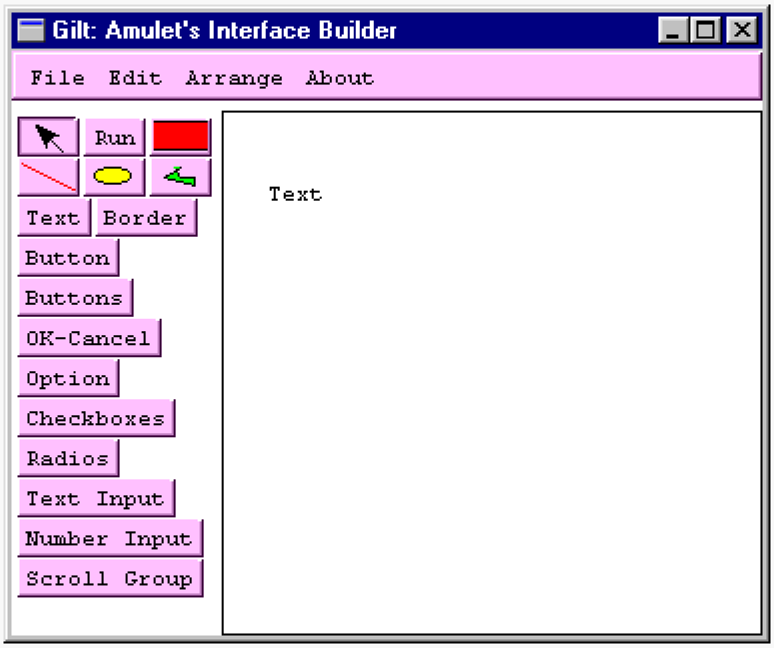
Created by the Amulet group
Based on AmEdit by S. Nebel from Linkworks Ltd, Wellington NZ

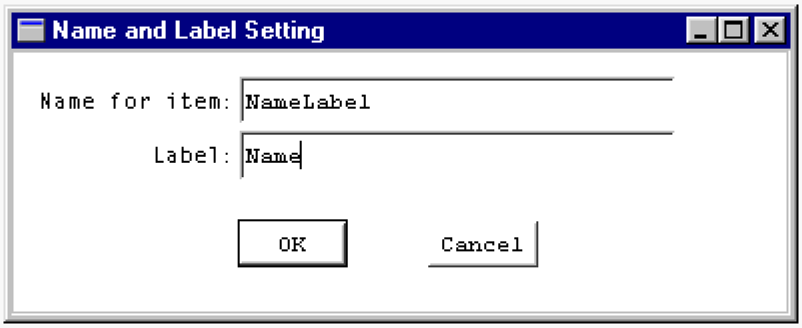
Adding a Text Label with GILT

Choose the "Text" button and click in the GILT window.

You can drag the item with the mouse or move it with the cursor keys.

Double-click to bring up a properties dialog:

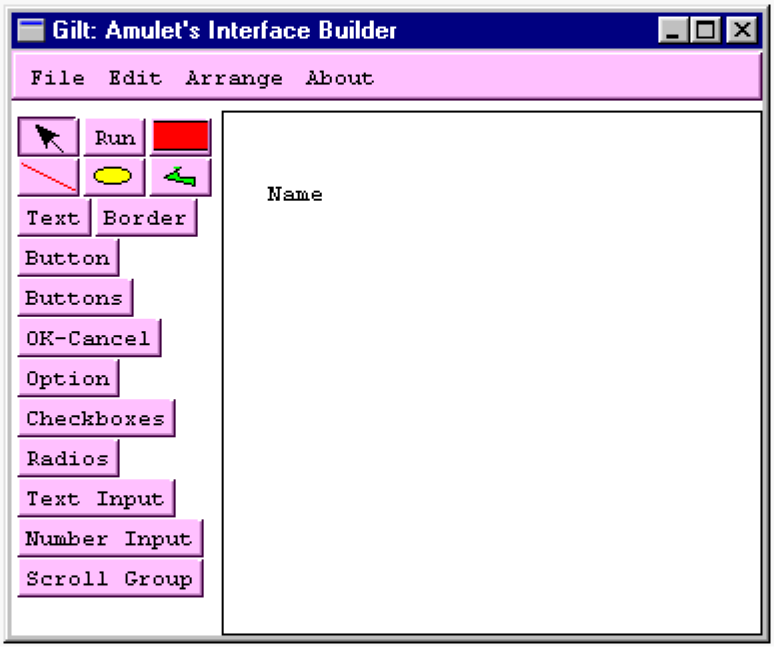



Name and Label Setting

Name for item: NameLabel

Label: Name

Fill in the dialog and press "OK" to update the GILT window…

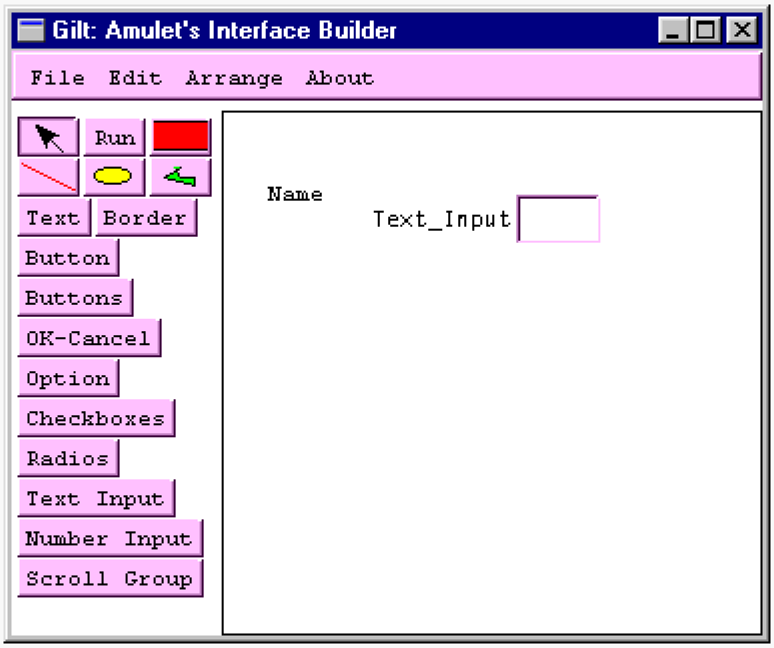


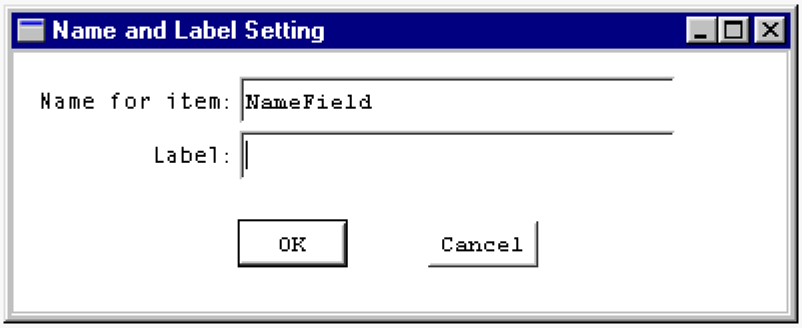
Adding a Text Input Field

Choose the “Text Input” button and click in the GILT window.

You can drag the item with the mouse or move it with the cursor keys.

Double-click to bring up a properties dialog:

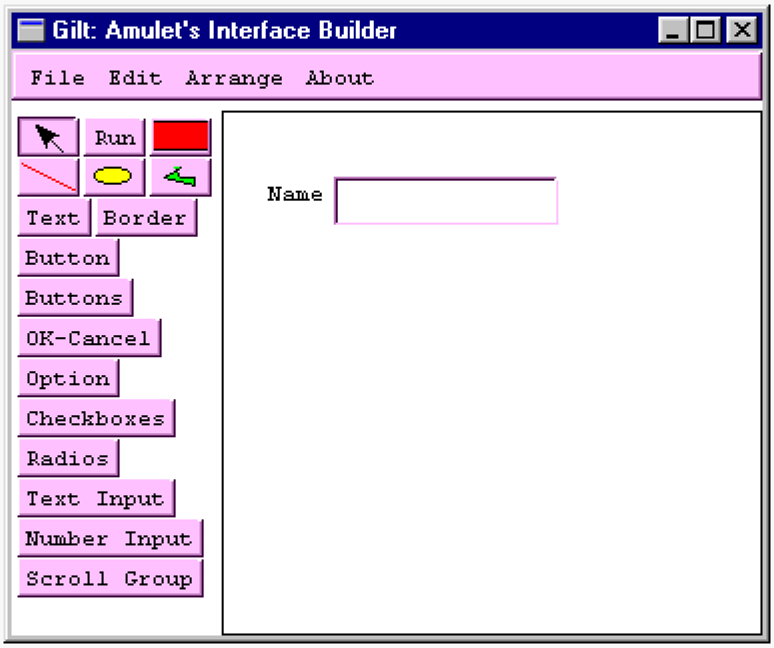




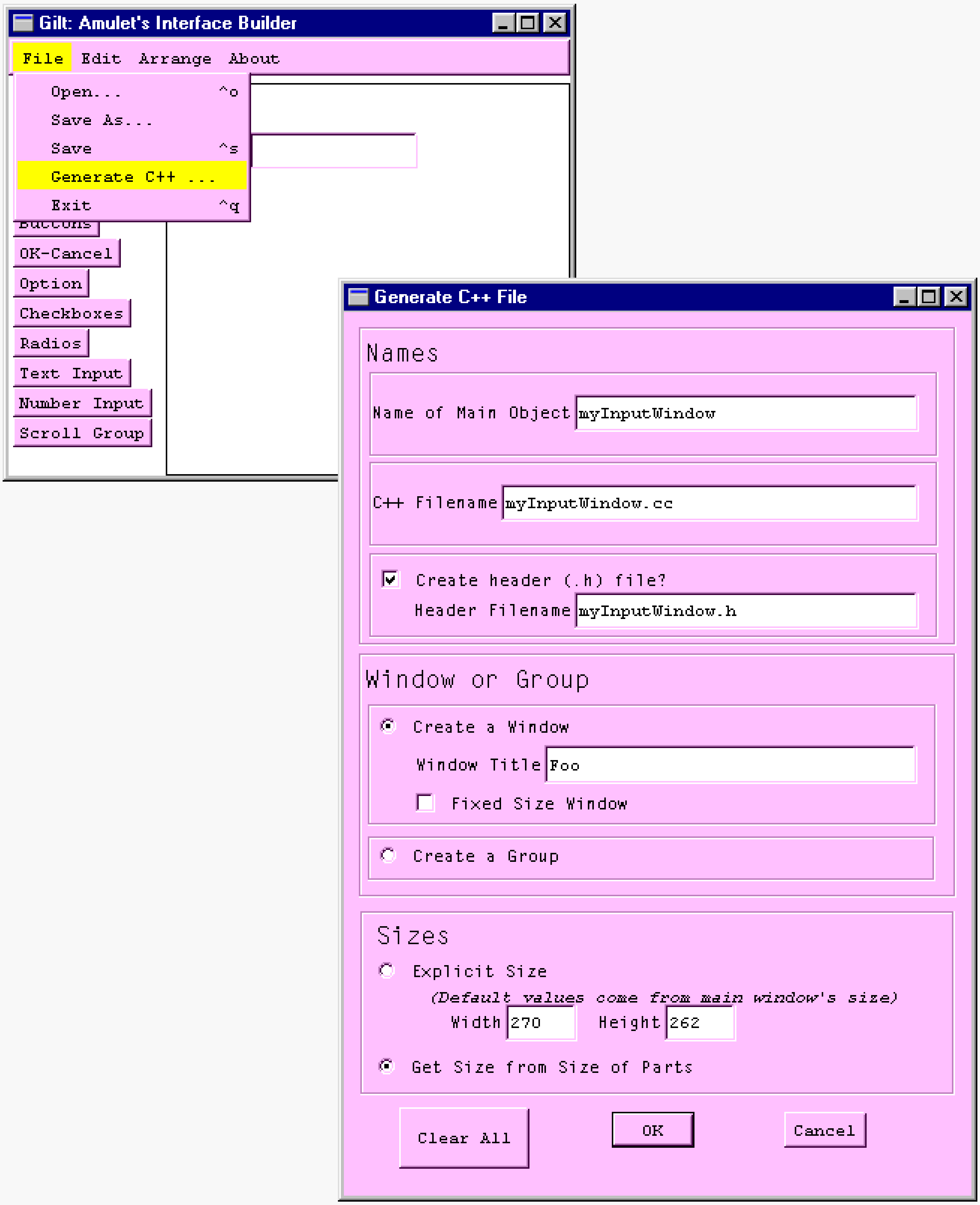
Name for item: NameField

Label:

Here, I specified a blank label (the text input object includes a label by default) since I prefer to specify a separate label field…



Generating C++ Source



Names

Name of Main Object: myInputWindow

C++ Filename: myInputWindow.cc

[✓] Create header (.h) file?

Header Filename: myInputWindow.h

Window or Group

(●) Create a Window

Window Title: Foo

[ ] Fixed Size Window

( ) Create a Group

Sizes

( ) Explicit Size

(Default values come from main window's size)

Width: 270 Height: 262

(●) Get Size from Size of Parts

GILT-generated Source

```
/****************************************************************
 *   The Amulet User Interface Development Environment
 ****************************************************************
 *   Created automatically by the Gilt program in Amulet.
 *   Do not edit this file directly.
 *   For more information on Amulet, contact amulet@cs.cmu.edu
 ****************************************************************
 *   Generated on Sun Apr 09 20:12:03 2000
 *   Amulet version 3.0
 ***************************************************************/
#include <amulet.h>

Am_Object myInputWindow;

Am_Slot_Key NameLabel = Am_Register_Slot_Name ("NameLabel");
Am_Slot_Key NameField = Am_Register_Slot_Name ("NameField");


Am_Object myInputWindow_Initialize () {
  myInputWindow = Am_Window.Create("myInputWindow")
    .Set(Am_DESTROY_WINDOW_METHOD,
Am_Default_Pop_Up_Window_Destroy_Method)
    .Set(Am_FILL_STYLE, Am_White)
    .Set(Am_TITLE, "Foo")
    .Set(Am_ICON_TITLE, "Foo")
    .Set(Am_WIDTH , Am_Width_Of_Parts)
    .Set(Am_HEIGHT, Am_Height_Of_Parts);
  myInputWindow
    .Add_Part(NameLabel, Am_Text.Create("NameLabel")
      .Set(Am_LEFT, 23)
      .Set(Am_TOP, 34)
      .Set(Am_WIDTH, 28)
      .Set(Am_HEIGHT, 14)
      .Set(Am_TEXT, "Name")
      .Set(Am_LINE_STYLE, Am_Black)
      .Set(Am_FILL_STYLE, Am_No_Style)
    )

// . . . continues . . .
```

GILT-generated Source

```
// . . . continued . . .

    .Add_Part(NameField, Am_Text_Input_Widget.Create("NameField")
      .Set(Am_LEFT, 54)
      .Set(Am_TOP, 33)
      .Set(Am_WIDTH, 115)
      .Set(Am_HEIGHT, 25)
      .Get_Object(Am_COMMAND)
        .Set(Am_LABEL, "")
        .Get_Owner()
      .Set(Am_FILL_STYLE, Am_Amulet_Purple)
    )
    .Add_Part(Am_Tab_To_Next_Widget_Interactor.Create())
  ;
  return myInputWindow;
}
```

A few notes:

- This source isn't any uglier than what I'd write myself, that's very unusual for automatically-generated code.
- GILT provides an easy way to lay out an interface and generate "starting-point" code for it. The code above will require some alterations (despite the warning in the header comment, that's a safe operation) to produce exactly what is needed.
- This source also reveals quite a bit of new information about how to make things work in Amulet, if you read it carefully.

GILT-generated Header

```
/**************************************************************
 *  The Amulet User Interface Development Environment
 **************************************************************
 *   Created automatically by the Gilt program in Amulet.
 *   Do not edit this file directly.
 *   For more information on Amulet, contact amulet@cs.cmu.edu
 **************************************************************
 *   Generated on Sun Apr 09 20:12:03 2000
 *   Amulet version 3.0
 **************************************************************/
#ifndef myInputWindow_H
#define myInputWindow_H

#include <amulet.h>

extern Am_Object myInputWindow;
extern Am_Object myInputWindow_Initialize ();

extern Am_Slot_Key NameLabel;
extern Am_Slot_Key NameField;

#endif
```