Implementation of the Graphical User Interface of the ParInt Package in Java

Gwowen Fu

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation
https://scholarworks.wmich.edu/masters_theses/4240
IMPLEMENTATION OF THE GRAPHICAL USER INTERFACE OF THE PARENT PACKAGE IN JAVA

by

Gwowen Fu

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
April 1998
ACKNOWLEDGMENTS

Special thanks go to Dr. Elise de Doncker for giving me the opportunity to work on this project and helping me solve all the difficulties I have encountered. Thanks go also to Dr. Ajay Gupta for suggesting some very useful features to add to this project to make it more user-friendly. I also would like to thank Dr. Robert Trenary for other support.

The most important person to whom I am thankful is my wife. She spent all her time taking care of our son and me. Without her I could not finish this project.

Gwown Fu
IMPLEMENTATION OF THE GRAPHICAL USER INTERFACE
OF THE PARINT PACKAGE IN JAVA

Gwowen Fu, M.S.

Western Michigan University, 1998

The graphical user interface (GUI) of the ParInt interface package provides a convenient and efficient way for users to integrate multivariate functions on a parallel system or a network.

This thesis introduces the basic structure of the GUI and gives a detailed description of each class of how the classes communicate with each other and how the connection with the parallel program is made.
TABLE OF CONTENTS

ACKNOWLEDGMENTS................................................................. ii

LIST OF FIGURES........................................................................ vi

CHAPTER

I. INTRODUCTION........................................................................... 1

II. CREATE THE PARENT APPLICATION CLASS.......................... 4

III. CREATE THE PARENT APPLICATION WINDOW CLASS.......... 7

   Menus.......................................................................................... 8

   File Menu..................................................................................... 9

   Method Menu............................................................................... 9

   System Menu.............................................................................. 10

   Language Menu.......................................................................... 10

   Integration Specification Menu............................................... 11

   Help Menu.................................................................................. 11

   Item Listener............................................................................... 12

   Action Listener.......................................................................... 12

   Other Methods.......................................................................... 14

IV. CREATE THE PARENT DOCUMENT CLASS............................ 15

    Methods................................................................................... 16

V. CREATE THE CONSTANTS INTERFACE................................. 18
CHAPTER

VI. CREATE THE FUNCTIONBOX CLASS ........................................ 19
    Constructor .......................................................... 20
    Action Listener ...................................................... 21
    Item Listener ....................................................... 22
    Other Methods ...................................................... 22

VII. CREATE THE PARABOX CLASS ........................................... 23
    Constructor .......................................................... 25
    Action Listener ...................................................... 26
    Item Listener ....................................................... 26
    Other Methods ...................................................... 27

VIII. CREATE THE NPBOX CLAS .............................................. 28
     Constructor .......................................................... 29
     Action Listener ...................................................... 30
     Item Listener ....................................................... 30
     Other Methods ...................................................... 31

IX. CREATE THE CONFIGBOX CLASS ....................................... 32
    Constructor .......................................................... 33
    Action Listener ...................................................... 34

X. CREATE THE COMPILEBOX CLASS ...................................... 35
    Constructor .......................................................... 36
    Action Listener ...................................................... 37
Table of Contents—continued

CHAPTER

XI. CREATE THE INTEGRATEBOX CLASS ........................................ 39
    Constructor ...................................................... 41
    Action Listener ................................................ 42
    Item Listener .................................................. 44

XII. CREATE THE MESSAGEBOX CLASS ...................................... 45
    Constructor ...................................................... 46
    Action Listener ................................................ 46

XIII. CLASSES FOR CHECKBOX MENU ITEMS .............................. 47

XIV. CONCLUSIONS AND FUTURE WORK .................................. 48

APPENDICES

A. Class Outline .................................................. 49

B. Tutorial ......................................................... 66

C. Programs List .................................................. 78

BIBLIOGRAPHY ....................................................... 149
LIST OF FIGURES

1. The Main ParInt Window ........................................ 2
2. The Architecture of the ParInt GUI .......................... 3
3. The ParInt Class Outline ....................................... 5
4. Relationship Between ParIntFrame and Dialog Windows .... 7
5. The ParIntFrame Class Outline ................................. 8
6. The Menu Bar ..................................................... 8
7. The File Menu .................................................... 9
8. The Method Menu ................................................ 10
9. The System Menu ................................................ 10
10. The Language Menu ............................................. 11
11. The Integration Specification Menu .......................... 11
12. The Help Menu .................................................. 12
13. The ParIntDoc Class Outline .................................. 15
14. The FunctionBox Class Outline ............................... 19
15. The FunctionBox Class Outline-Continued ................. 20
16. The FunctionBox Dialog ....................................... 20
17. The ParaBox Class Outline .................................... 23
18. The ParaBox Class Outline-Continued ....................... 24
19. The ParaBox Dialog ............................................. 24
List of Figures—continued

20. The NPBox Class Outline ......................................................... 28
21. The NPBox Dialog ............................................................... 29
22. The ConfigBox Class Outline .................................................. 32
23. The ConfigBox Dialog .......................................................... 33
24. The CompileBox Class Outline ................................................. 35
25. The CompileBox Dialog ......................................................... 36
26. The IntegrateBox Dialog ........................................................ 39
27. The IntegrateBox Class Outline ............................................... 40
28. The MessageBox Class Outline ................................................. 45
29. The MessageBox Dialog ......................................................... 45
CHAPTER I

INTRODUCTION

The graphical user interface (GUI) of the ParInt package provides a convenient and efficient way for users to integrate multivariate functions on a parallel system or network. The ParInt package is a software package for parallel and distributed integration. This GUI is developed on Windows NT/95 and Solaris platforms using version 1.1.4 of the Java Development Kit, also known as JDK 1.1.4. The user interface is written in Java and the integration functions are layered over MPI (Message Passing Interface). The purpose of this project is to provide a more general access method to integrate functions on a network of workstations or other parallel environment. The GUI is shown in Figure 1.

The reason for using Java is that it is one of the most popular languages and can be run on various machines. It would be very easy to have the program maintained and features added to it by other people.

This document focuses on the implementation of the user interface and how to use it. Each object will be discussed in detail in a separate chapter. Chapter II develops the ParInt application class. This class creates the ParIntDoc and ParIntFrame objects and provides methods for other objects to access them. Chapter III develops the ParIntFrame class, which creates a window with a menu bar and
invokes other dialog windows at run time. Chapter IV describes the ParIntDoc class. This class defines variables and methods to store or update the states of the checkbox menu items. The Constants interface defines constants is covered in Chapter V.

Chapter VI describes the FunctionBox class, which provides a pop-up window where the users can select a predefined function or input a user-defined function. In Chapter VII the ParaBox class is implemented to provide a pop-up window where the users can input parameters for integrand function. Chapter VIII implements the NPBox class. This class provides a pop-up window where the users can select the number of processes to calculate the integral. Chapter IX implements the ConfigBox class, which displays information, gathered from other objects. Chapter X implements the CompileBox. The CompileBox class provides an interface to compile the integrand function. Chapter XI implements the IntegrateBox class, which runs the integration program. Chapter XII implements the MessageBox class, which defines a simple dialog to display a short message. Chapter XIII describes classes needed to handle the
item listener for checkbox menu items. The conclusions and future designs are covered in Chapter XIV.

The architecture of the ParInt GUI is depicted in Figure 2 and described as follows. There are three main objects in this program: the application object (ParInt), the application window object (ParIntFrame), and the document object (ParIntDoc).

The ParInt object manages links between the other objects involved in the program. Any object that has access to the ParInt object can communicate with any other object as long as the ParInt object has methods to make each of the objects available. The ParIntFrame object defines the application window. The application window displays the menus and is the window through which users can interact with this program. The class ParIntDoc stores values needed by the application. These values can be read or changed by the methods in the ParIntFrame class.

Figure 2. The Architecture of the ParInt GUI.
CHAPTER II

CREATE THE PARINT APPLICATION CLASS

The application class is the core class, which creates objects at run time. The class is called ParInt in this program. It simply creates the ParIntDoc and ParIntFrame objects and provides methods for other objects to access them. For those objects which need to access the ParIntDoc and ParIntFrame objects must have the ParInt object passed to them.

This class defines a method called main() which is necessary for any application. When an application is run, the main() method is executed first. Usually this method will cause methods belonging to other classes to be executed. In the ParInt application class. The main method creates an instance of the ParInt object then calls the object's init() method to initialize the object.

In the init() method, the ParIntDoc and ParIntFrame objects are constructed. In order to obtain the size of the screen we are working on, a toolkit object is created which causes its method getSize() to get the size of the screen. To set the size and position, the ParIntFrame object calls setBounds(). The most important method is addWindowListener() called by the ParIntFrame object. This method will allow the object to receive window events. The ParIntFrame object is the window we see all the time after the program is executed. Its method setVisible() is set to true to
make it visible (the default is true). Call the requestFocus() method to make sure it is
on focus. The outline of this class is shown in Figure 3.

```java
java.lang.Object
----ParInt

Essential data members
static ParInt theApp;
ParIntFrame window;
ParIntDoc document;
Dimension wndSize;

Constructor
public ParInt()

Methods
public void init()
public static void main(String args[])
public ParIntDoc getDocument()
public ParIntFrame getAppWindow()
public Dimension getWindowDimension()
```

Figure 3. The ParInt Class Outline.

In order to provide other objects with information from the ParIntDoc and
ParIntFrame objects, we need the getAppWindow() and getDocument() methods,
respectively. The method getWindowDimension() is also defined in this class to
return the screen size, so that other objects do not have to create their own method.

Once the users have done with the program, they need to exit it. This task is
handled by the window events. The windowListener object watches the window
events. We need to create a listener class to handle these events. Class
Windowhandler is implemented for this purpose. It extends the interface
WindowAdapter class and implements the windowClosing() method to handle the
window closing event. The method `getID()` returns the event type, which can be used to test what kind of event is generated. In this case, the `windowClosing` method will close the application, thus it compares the event type with `WINDOW_CLOSING`. `WINDOW_CLOSING` is the window closing event type. In the `windowClosing()` method, the `ParIntFrame` object calls the `dispose()` method to free the window resource and `System.exit(0)` is called to exit the application.
CHAPTER III

CREATE THE PARINT APPLICATION WINDOW CLASS

This chapter will be the most important one since the application window contains the menus and this object invokes a lot of dialog boxes. The class is called ParIntFrame in this program. When the object is created by the ParInt object, its window with a menu bar is constructed and shown on the screen. All the dialog objects are created and invoked by this object, but they are not created at initial. A dialog object is created when its corresponding menu item is clicked. When the user clicked on the "Exit ParInt" menu item, the application will be terminated and all the resources will be freed. The relationship between ParIntFrame and those dialog windows is presented in Figure 4 and the class outline is shown in Figure 5.

Figure 4. Relationship Between ParIntFrame and Dialog Windows.
Menus

First we will create the menus shown in Figure 6; then we will add listeners for each of them.
**File Menu**

The "File" menu, shown in Figure 7, contains "Show Variables", "Compile", "Integrate", and "Exit ParInt". All four menu items are assigned the ParIntFrame object be their action listener via the addActionListener() method. The "Show Variables" menu item is used to pop up a window to show the variables selected by the user from other windows or checkbox menu items. The "Compile" menu item pops up a window to allow the user to compile the function and integration routines. The "Integrate" menu item pops up a window to let the user to start the integral computation. The "Exit ParInt" menu item allows the user to exit the application.

![Figure 7. The File Menu.](image)

**Method Menu**

The "Method" menu, shown in Figure 8, contains "Deterministic" and "Monte-Carlo" menu items. The "Deterministic" menu item contains submenu items and the submenu items contain checkbox menu items. Only "Local PQ—No LB" and "Local PQ with LB" are presently associated with the integral computation. Others are reserved for future use.
Figure 8. The Method Menu.

System Menu

The System menu allows the user to select the desired parallel environment to calculate the integral. The "System" menu contains menu items "PVM", "nCUBE", "MPI", "Sequential", "Number Processes" and "Options". The "Options" menu item contains a "Run PVM Console" submenu, which will be used later when PVM is supported by the GUI. At present time, the application can only connect to the MPI system. The System Menu is shown in Figure 9.

Figure 9. The System Menu.

Language Menu

The "Language" menu item allows the user to choose a language (from C, Java, and Fortran) for the code of the integrand function. In this version, only the C
language is available. The other two languages are considered for future extensions.

The Language Menu is shown in Figure 10.

![Language Menu](image)

Figure 10. The Language Menu.

**Integration Specification Menu**

The "Integration Specification" menu bar contains two menu items, "Function" and "Parameters". The "Function" menu item pops up a window which allows the user to select a function to integrate (from a sample set, or the user’s own function). The "Parameters" menu item pops up a window, which allows the user to input parameters, which are used, for the integration. The Integration Specification Menu is shown in Figure 11.

![Integration Specification Menu](image)

Figure 11. The Integration Specification Menu.

**Help Menu**

The "Help" menu contains two menu items, "Help System", and "About". The "Help System" menu item invokes the Netscape Browser to display the help files. The
"About" menu item displays a simple dialog describing this application. The Help Menu is shown in Figure 12.

![Help Menu](image)

Figure 12. The Help Menu.

**Item Listener**

There are three groups of checkbox menu items, "Method", "System", and "Language", so we can create three item listeners for each group. The listener for "Method" is setPQMenuChecks() and the listener for "System" is the method setSystemMenuChecks(). According to the current item selected by the user, it will update the check mark for each checkbox menu item. The listener for "Language" is the method setLanguageMenuChecks(). When the selected item is changed, it will update all the checkbox menu items. To update a checkbox, the method setState() is used to set its state to true or false.

**Action Listener**

There is only one action listener in this class. It is the method actionPerformed(). The class ParIntFrame must implement the class ActionListener and we have to implement the actionPerformed() method.
One way to catch the event of the menu item is by the method `getActionCommand()`, which is a method of class `ActionEvent`. For example:

```java
e.getActionCommand().equals("Exit Parent")
```

In this example, `e` is an `ActionEvent` object. The value returned by the `getActionCommand()` method is the label of the menu item. Using the method `equals()` can determine if these two strings match.

When the value matches "Show Variable", the object `ConfigBox` will be created.

When the value matches "Compile", the object `CompileBox` will be created.

When the value matches "Integrate", the object `IntegrateBox` will be created.

When the value matches "Exit Parent", the `dispose()` method is called to free the resource and the `exit()` method is called to end the application.

When the value matches "About", the object `aboutBox` will be created if it did not already exist. Otherwise, it is simply set to visible.

When the value matches "Help System", a `Runtime` object is created by calling its `getRuntime()` method and the process is created by calling the method `exec()`. The command we want to execute can be put into an array and then passed to the `exec()` method. In this case, the command is "netscape index.html". A Netscape browser will be invoked to show the index file of the help system.

When the value matches "Number Processes", the object `npBox` will be created if it did not already exist. The variable `m_bShowNP` is set to true. This
variable is used to enable the IntegrationBox object. When m_bShowNP, m_bShowFunction, and m_bShowPara are all set to true then IntegrationBox is enabled.

When the value matches "Function", the object FunctionBox is created if it did not exit. The variable m_bShowFunction is set to true. If the integrateBox exists, then the checkbox for the FunctionBox is checked. If both m_bShowNP and m_bShowPara are true then the menu items "Show Variable", "Compile", and "Integrate" are enabled.

When the value matches "Parameters", the object ParaBox is created if it did not exit. The variable m_bShowPara is set to true. If the integrateBox exists then the checkbox for the ParaBox is checked. If both m_bShowNP and m_bShowFunction are true then the menu items "Show Variable", "Compile", and "Integrate" are enabled.

Other Methods

The paint() method: the Component class defines the method paint(), which is called when the object needs to be drawn. This may occur, for instance when the component size changes, or if the component was obscured and becomes visible again. This method is overridden to display an image. The method drawImage() is used to show the image.

The finalize() method is called when an object is destroyed. Clean up codes are added here. In our finalize() method all windows are disposed of.
CHAPTER IV

CREATE THE PARINT DOCUMENT CLASS

The class ParIntDoc is used for storing values needed by the application.

These values can be read or changed by methods in the ParIntFrame class. This class implements the class Constants that will be discussed in next chapter. Methods defined in this class are used to get or set the state of the checkbox menu items on the menu bar. Its outline is shown in Figure 13.

```
java.lang.Object
+-----ParIntDoc

Essential data member
protected ParInt theApp;
protected int languageType;
protected int systemType;
protected int ruleType;
protected int pqType;

Constructor
public ParIntDoc(ParInt theApp)

Methods
public int getLanguageType()
public void setLanguageType(int languageType)
public int getSystemType()
public void setSystemType(int systemType)
public int getRuleType()
public void setRuleType(int ruleType)
public int getPQType()
public void setPQType(int pqType)
```

Figure 13. The ParIntDoc Class Outline.
Protected member variables are defined for holding the current status for each checkbox menu item group.

Methods

The constructor ParlntDoc() sets the default values for language type, system type, rule type, and PQ type. A copy of the object Parlnt is assigned to the member variable theApp, which will be used by the member methods to call methods defined in the class ParlntFrame.

The getLanguageType() method is defined to return the language type. The member variable languageType is returned directly.

The setLanguageType() method is defined to set the language type. The member variable languageType is assigned a new value. The method setLanguageMenuChecks() in class ParlntFrame is called to update the check marks shown on the menus.

The getSystemType() method returns the system type. The member variable systemType is returned.

The setSystemType() method is defined to set the system type. The member variable systemType is assigned a new value. The method setSystemMenuChecks() is called to update the checkbox marks in the "System" menu.

The getRuleType() method returns the rule type. The member variable ruleType is returned.
The setRuleType() method is defined to set the rule type. The member variable ruleType is assigned a new value. The method setRuleMenuChecks() is called to update the checkbox marks in the "Method" menu.

The getPQType() method returns the PQ type. The member variable pqType is returned.

The setPQType() method is defined to set the PQ type. The member variable pqType is assigned a new value. The method setPQMenuChecks() is called to update the checkbox marks in the "Method" menu.
CHAPTER V

CREATE THE CONSTANTS INTERFACE

The Constants interface defines all the constants used by this application. All system types, language types and method types are assigned a positive unique integer for each type as their id number. A default value is also set for each type. The default system type is MPI. The default language type is C. The default method type is “Local PQ — No LB”.

The predefined functions used in the class FunctionBox, which will be discussed in next chapter, are also defined in this interface.
The purpose of the FunctionBox class is to provide a pop-up window where the users can select a predefined function or input a user defined function. In order to be a visible window on the screen, a class must be a frame or a dialog, which is derived from the class Window. This class is a dialog so it extends Dialog. The class outline is shown in Figure 14 and 15.

```
java.lang.Object
    +----java.awt.Component
        +----java.awt.Container
            +----java.awt.Window
                +----java.awt.Dialog
                    +----FunctionBox
```

**Essential data members**
- setDefaultCloseOperation
delete()
- dispose()
- dispose()

```
java.lang.Object
    +----java.awt.Component
        +----java.awt.Container
            +----java.awt.Window
                +----java.awt.Dialog
                    +----FunctionBox
```

Figure 14. The FunctionBox Class Outline.

The FunctionBox class implements the Constants interface to reference the predefined functions. It also implements the ActionListener interface to catch the action event issued by the "OK" button and the "UPDATE" button. The FunctionBox dialog is shown in Figure 16 on page 20.
Constructors
public FunctionBox(ParIntFrame, String, ParInt)

Methods
public void actionPerformed(ActionEvent e)
public String getFunction()
public boolean getVisible()

Inner classes
class choice2Listener implements ItemListener
class CheckboxHandler implements ItemListener

Figure 15. The FunctionBox Class Outline-Continued.

Figure 16. The FunctionBox Dialog.

Constructor

Most of the work is done in the constructor. The size of the dialog is set to 450 * 300. The dimension of the main window is obtained by the method getBounds() defined in the java.awt.Rectangle class and assigned to bounds. The screen size is
obtained by calling the getWindowDimension() method, which is defined in the 
ParInt class, and assigned to the member variable dim which has type Dimension. A 
calculation is done to put this dialog on a proper position of the screen. If the screen is 
big enough to show all the windows, the FunctionBox will appear on the right of the 
main window. A layout manager is created to handle the appearance of the compone-
ts in the dialog.

A TextArea object is created to display a message. The top checkbox is 
created to select predefined functions. The bottom checkbox is created to input the 
function from a file. The choice2 object is an instance of class java.awt. Choice, 
which shows a list of predefined functions when it is clicked. The choice2 object is 
enabled when the top checkbox is checked. The textField object is an instance of 
class java.awt. TextField, which lets the users input the file name of the function. It is 
associated with the bottom checkbox. It is enabled only when the bottom checkbox is 
checked. The "UPDATE" button is an instance of Button. It is used to update the 
current function input by the user. Another Button object "OK" is created to allow the 
user to hide the dialog box.

Action Listener

The method actionPerformed() is the action listener for buttons "UPDATE" 
and "OK". When the button "UPDATE" is clicked, the getText() method of the object 
textField is called to get the current text in the text field and it is assigned to the 
member variable m_function. The m_function will be returned when another object
requests the function. When the button "OK" is clicked, the member variable m_bVisible is set to false. The m_bVisible is provided for the object integrateBox to determine whether the functionBox object is displayed or hidden.

Item Listener

There are three item listeners. These listeners have to implement the ItemListener interface, so they can catch the events generated by the choice objects.

The second item listener is choice2Listener which is associated with choice2. When the object choice2 is clicked and the selected item has been changed, then the listener will execute the method itemStateChanged(). In this method, the new item is also assigned to the member variable m_function.

The third item listener is CheckboxHandler, which monitors the events triggered by the checkboxes. When one of the checkboxes is checked, the method itemStateChanged() will be started. In the itemStateChanged() method, the getSelectedCheckbox() method is used to get the selected checkbox, to enable associated objects and disable not associated objects.

Other Methods

There are two more methods. The GetFunction() method will return the member variable m_function to the calling object. The getVisible() method returns the member variable m_bVisible to the calling object.
CHAPTER VII

CREATE THE PARABOX CLASS

The purpose of this ParaBox class is to provide a pop-up window where the users can input the error tolerances, limit on the number of function evaluations, and select an integration rule. These parameters must be set for the first time to integrate the integrand function. This class extends the java.awt.Dialog class. A dialog may take input from the user. This class also implements the java.awt.event.ActionListener interface for receiving action events. The class outline is shown in Figure 17 and 18. Figure 19 shows the ParaBox dialog.

```
java.lang.Object
 +----java.awt.Component
      +----java.awt.Container
           +----java.awt.Window
                +----java.awt.Dialog
                     +----ParaBox

Essential data members
ParaIntFrame frame;
TextField toleranceField, r_toleranceField, ruleField, limitField;
TextField inputField, dimField, lowField, upField;
String m_a_tolerance, m_r_tolerance, m_rule, m_limit, m_filename;
String m_dimension;
Dimension dim;
Checkbox inputCheckbox;
Choice lowChoice, upChoice;
Button button, defButton;
Button update, addLowButton, addUpButton, lowReset, upReset;
```

Figure 17. The ParaBox Class Outline.
Constructors
public ParaBox(ParIntFrame, String, ParInt)

Methods
public void actionPerformed(ActionEvent e)
public String getA_tolerance()
public String getR_tolerance()
public String getRule()
public String getLimit()
public String getDimension()
public String getFilename()
public Choice getLowBound()
public Choice getUpBound()
public boolean getVisible()

Inner classes
class CheckboxHandler implements ItemListener

Figure 18. The ParaBox Class Outline-Continued.

Figure 19. The ParaBox Dialog.
Constructor

The class ParaBox is derived from class Dialog. To make it a modeless dialog we need to call the base constructor. The statement in the ParaBox class constructor, which does this, is:

```
super(frame, title, false);
```

This should be the first statement in the derived class constructor. The frame is the parent window for the dialog. The frame here is the ParIntFrame object. The title will be shown on the title bar. The boolean value false tells the base constructor to create a modeless dialog.

The size of the dialog is set to 450 by 300. The dimension of the main window is obtained by the method getBounds() defined in the java.awt.Rectangle class and assigned to bounds. The screen size is obtained by calling the getWDWindowDimension() method which is defined in the ParInt class and assigned to the member variable dim which has type Dimension. A calculation is done to put this dialog at a proper position in the screen. If the screen is big enough to show all the windows, the ParaBox will appear on the right of the main window. A layout manager is created to handle the appearance of the components in the dialog.

A TextArea object is created to display the information regarding the four rules. Four labels explain which values can be input into the text fields on their right side.
A checkbox is created to let the user can only choose to input the parameters or to select a file which already contains these parameters. The text field on its right side is used to input the file name. The "UPDATE" button is used to update the parameters, which are input by the user or from a file. The "OK" button is used to hide the dialog.

The "OK" button is disabled by default. It will not be enabled until the "UPDATE" button is clicked. This ensures the parameters to exist.

Action Listener

The method actionPerformed() is the action listener for buttons "UPDATE" and "OK". When the button "UPDATE" is clicked, the getText() methods of the toleranceField, r_toleranceField, ruleField, limitField, and inputField objects are called to get their current texts and to assign them to the member variables m_a_tolerance, m_r_tolerance, m_rule, m_limit, and m_filename, respectively. These variables are prepared for the integrateBox object. When the "OK" button is clicked the member variable m_bVisible is set to false. The m_bVisible is provided to the object integrateBox to determine whether the paraBox object is displayed or hidden.

Item Listener

There is an item listener, CheckboxHandler(), implementing the ItemListener interface to catch the events generated by the checkbox object. When the checkbox is checked, the inputField object is enabled and the toleranceField, r_toleranceField,
ruleField and limitField objects are disabled. If the checkbox is disabled, then the inputField object is disabled and the toleranceField, r_toleranceField, ruleField and limitField objects are enabled.

Other Methods

There are six other methods. The getA_tolerance() method returns the member variable m_a_tolerance. The getR_tolerance() method returns the member variable m_r_tolerance. The getRule() method returns the member variable m_rule. The getLimit() method returns the member variable m_limit. The getFilename() method returns the member variable m_filename if it is not null. The getVisible() method returns the member variable m_bVisible.
CHAPTER VIII

CREATE THE NPBOX CLASS

The purpose of this NPBox class is to provide a pop-up window where the users can select the number of processes to calculate the integral. This class extends the java.awt.Dialog class. This class also implements the java.awt.event.ItemListener interface to receive item events; and implements the java.awt.event.ActionListener interface to receive action events. The class outline is shown in Figure 20 and the NPBox dialog is shown in Figure 21.

```
java.lang.Object
 +----java.awt.Component
      +----java.awt.Container
          +----java.awt.Window
              +----java.awt.Dialog
                  +----NPBox

Essential data members
ParIntFrame frame;
Choice choice;
static int np;
Dimension dim;

Constructors
public NPBox(ParIntFrame frame, String title, ParInt theApp)

Methods
public void actionPerformed(ActionEvent e)
public void itemStateChanged(ItemEvent e)
public int getValue()
public boolean getVisible()
```

Figure 20. The NPBox Class Outline.
Most of the procedures for area division only support process quantities in power of 2. Thus, for ease, we impose this restriction in this version of ParInt.

Figure 21. The NPBox Dialog.

Constructor

The class NPBox is derived from class Dialog and set as a modeless dialog. The size of the dialog is set to 450 by 300. The dimension of the main window is obtained by the method getBounds() defined in the java.awt.Rectangle class and assigned to bounds. The screen size is obtained by calling the getWindowDimension() method which is defined in the ParInt class and assigned to the member variable dim which has type Dimension. A calculation is done to put this dialog at a proper location in the screen. If the screen is big enough to show all the windows, the NPBox will appear on the right of the main window. A layout manager is created to handle the appearance of the components in the dialog.

A Label object is created to display the number of processes. Another static Label object shows "Please select the number of dimensions of processes to use". A
Choice object is created to let the user select the number of dimension. A TextArea object is created to display some additional information. A Button object "OK" is created to exit the dialog box.

**Action Listener**

The method actionPerformed() is the action listener for buttons "OK". When the button "OK" is clicked, the member variable m_bVisible is set to false. The m_bVisible is provided for the object integrateBox to determine whether the paraBox object is displayed or hidden. If the IntegrateBox object exist, then the statement frame.integrateBox. npcheckBox.setState(false) will be extended to update the current visibility of the npBox object for the integrateBox object. The statement setVisible(false) hides the npBox object.

**Item Listener**

The npBox object is also an item listener. The method itemStateChanged() is implemented to catch the events generated by the choice object. When the choice object is clicked, then the index of the current item will be obtained and assigned to the member variable np. The value 2 to the power of np will be shown on the label object.
Other Methods

There are two more methods. The getValue() method returns the square of the member variable np. The getVisible() method returns the member variable m_bVisible.
CHAPTER IX

CREATE THE CONFIGBOX CLASS

This class displays information gathered from other objects. The current integration system and programming language are obtained from the object window. The number of processes is obtained from the object npBox and the current function is obtained from the object functionBox. A pop-up window is provided to show this information. This class extends the java.awt.Dialog class to be a visible dialog and implements the ActionListener interface to receive action events for the "OK" button. The class outline is shown in Figure 22 and its dialog is shown in Figure 23.

```
java.lang.Object
+----java.awt.Component
    +----java.awt.Container
        +----java.awt.Window
            +----java.awt.Dialog
                +----ConfigBox

Essential data members
ParInt theApp;
ParIntDoc document;
String[] languageType = { "C", "JAVA", "FORTRAN" };
String[] systemType = { "PVM", "NCUBE", "MPI", "SEQUENTIAL" };

Constructors
public ConfigBox(ParIntFrame frame, String title, ParInt theApp)

Methods
public void actionPerformed(ActionEvent e)
```

Figure 22. The ConfigBox Class Outline.
Constructor

This dialog is created as a modeless dialog by passing the Boolean value false to the method super().

The size of the dialog is set to 550 by 300 by the setSize() method. The dimension of the main window is obtained by the method getBounds() defined in the java.awt.Rectangle class and assigned to bounds. The screen size is obtained by calling the getWindowDimension() method which is defined in the ParInt class and assigned to the member variable dim which is of type Dimension. A calculation is done to put this dialog at a proper position in the screen. If the screen is big enough to show all the windows, the ConfigBox will appear on the left of the main window. A layout manager is created to handle the appearance of the components in the dialog.

Figure 23. The ConfigBox Dialog.

\[
f(x) = x_0^2 + 2\sin(x_2)^4 + x_3 + x_4 + x_5 - 1 \cdot \text{lgamma}(x_6)
\]
The dialog box is divided into two parts. The left column contains the description, the right column the value. The first row shows the titles for both columns. The current integration system is listed below the title. In order to get the value, we need to get the id from the method `getSystemType()` defined in the `ParIntDoc` class. Then we have to define a string array for all the system types corresponding to the id numbers. Finally, the id can be used as an index into the string array to get the current system type. This method is also used to get the language type but the id for the language type is obtained by calling the `getLanguageType()` method.

The value of the number of processes is obtained by calling the `getValue()` method defined in `npBox` object. We can not call it directly. It must be called by the frame object because the `npBox` object is created in the frame object. The value is an integer number and it is shown as a caption. The caption has to be a string so it is converted into a string by the method `Integer.toString()`. The current function is obtained using the same way as above, but the method `functionBox.getFunction()` is used to get the function.

**Action Listener**

The method `actionPerformed()` is the action listener for "OK" button. When the "OK" button is clicked, the method `dispose()` is called to free the resource and exit the dialog.
CHAPTER X

CREATE THE COMPILBOX CLASS

The CompileBox class provides an interface to compile the integrand function and link with ParInt. The "Makefile" used for compiling the MPI programs is invoked by this class. This class is also a dialog where the users can click on the "MAKE" button to run the "Makefile" or the "OK" button to exit the dialog. There is a TextArea object, which displays the messages of the compilation. This class extends the java.awt.Dialog class. This class also implements the java.awt.event.ActionListener interface to receive action events. The class outline is shown in Figure 24 and its dialog is shown in Figure 25.

```
java.lang.Object
   +----java.awt.Component
        +----java.awt.Container
                +----java.awt.Window
                        +----java.awt.Dialog
                                 +----CompileBox

Essential data members
ParInt theApp;
ParIntDoc document;
String[] systemType = { "PVM", "NCUBE", "MPI", "SEQUENTIAL" };
Button makeButton, okButton;

Constructors
public CompileBox(ParIntFrame frame, String title, ParInt theApp)

Methods
public void actionPerformed(ActionEvent e)
```

Figure 24. The CompileBox Class Outline.
The class CompileBox is derived from the class Dialog and set as a modeless
dialog by passing the Boolean value false to the method super(). The size of the
dialog is set to 450 by 300 by the setSize() method. The dimension of the main
window is obtained by the method getBounds() defined in the java.awt.Rectangle
class and assigned to bounds. The screen size is obtained by calling the
getWindowDimension() method which is defined in the ParInt class and assigned to
the member variable dim which is of type Dimension. A calculation is done to put
this dialog at a proper position in the screen. If the screen is big enough to show all
the windows, the CompileBox will appear on the left of the main window. A layout
manager is created to handle the appearance of the components in the dialog.
A label is added into the dialog to display a message "Click on Make to compile and link the integrand function". A "MAKE" button is used to compile and link the integrand function when it is clicked. Its listener is the CompileBox object. A TextArea object is created of 4 (rows) by 40 (columns). The messages will be displayed in this object. There is an "EXIT" button so that the user can exit the dialog. The "EXIT" button also has the CompileBox object as its listener.

Action Listener

The method actionPerformed() is the action listener the "MAKE" and "EXIT" buttons. When the "MAKE" button is clicked, a run time object is created. This is accomplished by the following statement:

```
Runtime r = Runtime.getRuntime();
```

The Runtime object r is used to create a process and run the "Makefile". We need to specify the command to be executed. A String object cmdarray[] is created to store the command we wish to execute. To run the process we need to put it in a try block so we can catch the exceptions. A try block is simply the keyword try, followed by braces enclosing the code that can throw the exception. A catch block containing the code to handle the exception follows the try block, which contains the code that may throw that particular exception. The method exec() of the r object creates the process p which runs the "Makefile". Both the "MAKE" and "EXIT" buttons are disabled and wait until the process finishes running. After the process terminated it is destroyed
and the "MAKE" and "EXIT" buttons are again enabled. Two exceptions must be caught. One is an IOException, which signals that an I/O exception of some sort has occurred. Another is an InterruptedException, which occurs when a thread is interrupted by an interrupt method in class Thread. When the process is running all the messages are written to the file "report". In a try block, the file is read into the textArea object. A following catch block catches the FileNotFoundException. To read the file, a FileReader object is created to open the file. A while loop keeps reading the file until an EOF flag is met. The read() method of the "in" object is used to read the file. We also need a try block for the read and two catch blocks to catch the EOFException and IOException.
CHAPTER XI

CREATE THE INTEGRATEBOX CLASS

The IntegrateBox class collects the information to run the integration program. The information includes the number of processes, the absolute and relative tolerances, the maximum number of integrand evaluation (limit) and the integrand function. A textArea object is derived from the TextArea class to display the messages that are output by the MPI program. Three checkboxes are used to show/hide the functionBox object, the paraBox object or the npBox object. The button with caption "Run" is used to run the integral computation. The button with caption "Exit" is used to destroy the integralBox object. Its dialog is shown in Figure 26.

Figure 26. The IntegrateBox Dialog.
java.lang.Object
    +----java.awt.Component
        +----java.awt.Container
            +----java.awt.Window
                +----java.awt.Dialog
                    +----IntegrateBox

Essential data members
ParInt theApp;
ParIntDoc document;
ParIntFrame frame;
String[] systemType = { "PVM", "NCUBE", "MPI", "SEQUENTIAL" };
Button makeButton, okButton, ClsButton;
int np_value, pq;
Checkbox funCheckbox, npCheckbox, paraCheckbox;
String cmdarray;
String m_a_tolerance, m_r_tolerance, m_rule, m_limit, m_dimension;
Choice lowBound, upBound;

Constructors
public IntegrateBox(ParIntFrame frame, String title, ParInt theApp)

Methods
public void actionPerformed(ActionEvent e)

Inner Class
class CheckboxHandler implements ItemListener

Figure 27. The IntegrateBox Class Outline.

This class extends the java.awt.Dialog class to be a dialog and implements the
Constants interface to access the predefined functions. It also implements the
java.awt.event ActionListener interface to receive action events. The class outline is
shown in Figure 27.

The integrateBox object runs the command "make" followed by the target
name. The prefix of the target in the "Makefile" is "adapt-lb" or "adapt-nolb" and the
postfix is the number of the processes. The dependency looks like:

  mpirun -np 2 adapt-lb < datain > report
Mpirun is the command to run a MPI program. The number 2 indicates how many processes are used. The executables adapt-lb are the actual programs running the integral computation. The file "datain" contains the parameters needed for running the MPI program. The output file "report" used to display the output messages.

Constructor

The class integrateBox is derived from the class Dialog and set as a modeless dialog by passing the Boolean value false to the method super(). The size of the dialog is set to 450 by 350 by the setSize() method. The dimension of the main window is obtained by the method getBounds() defined in the java.awt.Rectangle class and assigned to bounds. The screen size is obtained by calling the getWindowDimension() method which is defined in the ParInt class; it is assigned to the member variable dim which is of type Dimension. A calculation is done to put this dialog at a proper position in the screen. If the screen is big enough to show all the windows, the integrateBox will appear on the left of the main window. A layout manager is created to handle the appearance of the components in the dialog.

A textArea object derived from the TextArea class is added to the top of the dialog and displays the message Click on "Run" to run the chosen functions. Below the textArea object are three checkboxes. The left checkbox is associated with the functionBox object. When the checkbox is checked the functionBox is displayed. Otherwise it is hidden. The second checkbox is associated with the paraBox object. When it is checked then paraBox object is displayed. Otherwise it is hidden. The right
checkbox is for the npBox object. These three are registered to have the item listener CheckboxHandler() class. A "Run" button is created to run the integral computation when it is clicked. Its listener is the integrateBox object. An "EXIT" button is created to allow the user to exit the dialog. The "EXIT" button also has the integrateBox object as its listener.

Action Listener

The method actionPerformed() is the action listener for the buttons "Run" and "Exit". When the "run" button is clicked the "datain" file will be created to store the information needed by the MPI program. The first value is the function. The function is got by the getFunction() method of the functionBox object. It has to be called via the frame object, because the functionBox object is created in the frame object. The second value is the absolute tolerance obtained by the getA_tolerance() method of the paraBox object via the frame object. The third value is the relative tolerance obtained by the getR_tolerance() method of the paraBox object. The rule is also obtained from the paraBox object by the method getRule(). All these are enclosed in a try block. The catch block will catch the IOException. A try block is used to get the number of processes which is assigned to the member variable np_value by calling the getValue() method of the npBox object. Its catch block sets the member variable np_value to 1.

When all the values are set, the following statement creates a Runtime object:

```java
Runtime r = Runtime.getRuntime();
```
The Runtime object r is used to create a process and invoke the "Makefile". We need to specify the command to be executed. A String object cmdarray[] is created to store the command we wish to execute. To run the process we need to put it in a try block so we can catch the exceptions. A try block is simply the keyword try, followed by braces enclosing the code that can throw the exception. A catch block containing the code to handle the exception follows the try block, which contains the code that may throw that particular exception.

The method execO of the r object creates the process p which runs the "Makefile". Both the "Run" and "Exit" buttons are disabled and wait until the process terminates. When the process has terminated, it is destroyed and the "Run" and "Exit" buttons are back enabled. Two exceptions must be caught. One is IOException that signals that an I/O exception of some sort has occurred. Another is the InterruptedException, which occurs when a thread is interrupted by an interrupt method in class Thread.

When the program is executed all the messages are written to the file "report". In a try block, the file is read into the textArea object. A catch block catches the FileNotFoundException. To read the file, a FileReader object "in" is created to open the file. A while loop keeps reading the file until an EOF flag is met. The read() method of the "in" object is used to read the file. We also need a try block for the read and two catch blocks to catch the EOFException and IOException.
Item Listener

The item listener has the itemStateChanged() method implemented to run some specific tasks after any checkbox' status is changed. When the listener is invoked, it will execute the setVisible() methods of the npBox, functionBox, and paraBox objects. The getState() method is the key to determine whether the object is visible or not visible. If the state returned from getState() is true then it is visible. Otherwise it is not visible.
CHAPTER XII

CREATE THE MESSAGEBOX CLASS

The MessageBox class defines a simple dialog to display a short message. This class extends the java.awt.Dialog class to be a visible dialog and implements the ActionListener interface to receive action events for the "OK" button. This dialog contains the message of "ParInt Project Java version 1.0" and an "OK" button. The class outline is shown in Figure 28 and the dialog is shown in Figure 29.

```
java.lang.Object
   +----java.awt.Component
      +----java.awt.Container
         +----java.awt.Window
            +----java.awt.Dialog
               +----MessageBox

Essential data members
Frame frame;

Constructors
public MessageBox(Frame frame, String title, String message)

Methods
public void actionPerformed(ActionEvent e)
```

Figure 28. The MessageBox Class Outline.

Figure 29. The MessageBox Dialog.
Constructor

The constructor MessageBox() first puts itself in the middle of the frame. The method getBounds() of the frame object is used to get the size and the location of the frame. The setLocation() method is used to set the location for the MessageBox object.

BorderLayout is the layout manager to manage the appearance of the dialog. A panel on the top contains the label that displays the message. A panel on the bottom contains the "OK" button. The button adds the action listener by the addActionListener() method.

Action Listener

The method actionPerformed() is the action listener for the "OK" button. When the "OK" button is clicked, the method dispose() is called to free the resource and exit the dialog.
CHAPTER XIII

CLASSES FOR CHECKBOX MENU ITEMS

There are four classes that are implemented for handling the item listener for checkbox menu items. These are the RuleCommand, LanguageCommand, SystemCommand, and PQCommand classes. They import java.awt.event.ItemListener and implement ItemListener to catch the item events issued by the checkbox menu items. The menus involved are the method, system, and language menus.

Each class has the same constructor (except for the name of the constructor). There are two statements for the constructor. First, the member variable itemID assigned the value passed by the calling object to save the menu item ID for that menu. Second, the member variable document is assigned the instance of the ParlIntDoc object.

The method itemStateChanged() implements the ItemListener interface to catch the events generated by the checkbox menu items. The current menu item will be stored in the ParlIntDoc object when the checkbox menu item is checked. If the checkbox menu item is in the "method" menu, the method setRuleType() is called. If it is in the "system" menu, then setSystemType() is called. If it is in the "language" menu, then setLanguageType() is called.
CHAPTER XIV

CONCLUSIONS AND FUTURE WORK

Today, the internet is very popular and every one can access it without trouble. It would be convenient if the user could run the ParInt GUI in a web browser, so users do not have to install the Java Development Kit locally. We are considering to implement this in the future.

It will not be very easy to modify the existing ParInt GUI. A Java application is different from an applet so it cannot run in a web browser. Some methods adopted in the ParInt GUI will not be allowed in an applet.

Another future design for the ParInt GUI is to support the PVM and nCUBE integration code. The current version only supports the MPI integration code.

Tcl/Tk is an alternate tool for implementing a graphical user interface. We opted for Java for several reasons. Java is compiled and should be faster than Tcl/Tk. Java is similar in syntax to C++; thus it is easy for many programmers to gain familiarity with Java. Java is object oriented and multi-threading or is more structured than Tcl/Tk. Finally the Java class library is relatively complete.
Class Hierarchy

- class java.lang.Object
    - class java.awt.Container
      - class java.awt.Window
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - class ParIntFrame (implements Constants, java.awt.event.ActionListener)
  - interface Constants
  - class ParInt
  - class ParIntDoc (implements Constants)
Class ParInt

```java
java.lang.Object
    +----ParInt
```

public class ParInt
extends Object

--- Constructor Index ---

- ParInt()

--- Method Index ---

- getAppWindow()
- getDocument()
- getWindowDimension()
- init()
- main(String[])

--- Constructors ---

- ParInt

    public ParInt()

--- Methods ---

- init

    public void init()

- main

    public static void main(String args[])
**getDocument**

```java
public ParIntDoc getDocument()
```

**getAppWindow**

```java
public ParIntFrame getAppWindow()
```

**getWindowDimension**

```java
public Dimension getWindowDimension()
```
Class ParIntFrame

public class ParIntFrame
extends Frame
implements Constants, ActionListener

Constructor Index
- ParIntFrame(String, ParInt)

Method Index
- actionPerformed(ActionEvent)
- finalize()
- paint(Graphics)
- setLanguageMenuChecks(int)
- setPQMenuChecks(int)
- setRuleMenuChecks(int)
- setSystemMenuChecks(int)
Constructors

public ParIntFrame(String title, ParInt theApp)

Methods

public void paint(Graphics g)

Overrides:

paint in class Container

public void setLanguageMenuChecks(int languageType)

public void setSystemMenuChecks(int systemType)

public void setRuleMenuChecks(int ruleType)

public void setPQMenuChecks(int pqType)

public void actionPerformed(ActionEvent e)

protected void finalize()

Overrides:

finalize in class Object
Class ParIntDoc

java.lang.Object
  +---ParIntDoc

public class ParIntDoc
extends Object
implements Constants

Field Index

- languageType
- pgType
- ruleType
- systemType
- theApp

Constructor Index

- ParIntDoc(ParInt)

Method Index

- getLanguageType()
- getPQType()
- getRuleType()
- getSystemType()
- setLanguageType(int)
- setPQType(int)
**Fields**

- **theApp**
  
  protected Int theApp

- **languageType**
  
  protected int languageType

- **systemType**
  
  protected int systemType

- **ruleType**
  
  protected int ruleType

- **pqType**
  
  protected int pqType

**Constructors**

- **ParIntDoc**
  
  public ParIntDoc(ParInt theApp)

**Methods**

- **getLanguageType**
  
  public int getLanguageType()

- **setLanguageType**
  
  public void setLanguageType(int languageType)

- **getSystemType**
  
  public int SystemType()
public void setSystemType(int systemType)

getRuleType

public int getRuleType()

setRuleType

public void setRuleType(int ruleType)

getPQType

public int getPQType()

setPQType

public void setPQType(int pqType)
Interface Constants

public interface Constants

Field Index

• C
• DEFAULT LANGUAGE TYPE
• DEFAULT PQ TYPE
• DEFAULT RULE TYPE
• DEFAULT SYSTEM TYPE
• FORTRAN
  • fun1
  • fun10
  • fun11
  • fun12
  • fun13
  • fun2
  • fun3
  • fun4
  • fun5
  • fun6
  • fun7
  • fun8
  • fun9
```java
public static final int PQ1
public static final int PQ2
public static final int PQ3
```

```java
public static final int RULE1
public static final int RULE2
public static final int RULE3
```

```java
public static final int PVM
```

---

**Fields**

- PQ1
  - public static final int PQ1
- PQ2
  - public static final int PQ2
- PQ3
  - public static final int PQ3
- RULE1
  - public static final int RULE1
- RULE2
  - public static final int RULE2
- RULE3
  - public static final int RULE3
- PVM
public static final int PVM

public static final int NCUBE

public static final int MPI

public static final int SEQUENTIAL

public static final int C

public static final int JAVA

public static final int FORTRAN

public static final int DEFAULT_PQ_TYPE

public static final int DEFAULT_RULE_TYPE

public static final int DEFAULT_SYSTEM_TYPE

public static final int DEFAULT_LANGUAGE_TYPE

public static final String fun1

public static final String fun2

public static final String fun3
•fun4
public static final String fun4

•fun5
public static final String fun5

•fun6
public static final String fun6

•fun7
public static final String fun7

•fun8
public static final String fun8

•fun9
public static final String fun9

•fun10
public static final String fun10

•fun11
public static final String fun11

•fun12
public static final String fun12

•fun13
public static final String fun13
Index of all Fields and Methods

A

`actionPerformed(ActionEvent)`. Method in class `ParIntFrame`

C

C. Static variable in interface `Constants`

D

`DEFAULT_LANGUAGE_TYPE`. Static variable in interface `Constants`

`DEFAULT_PQ_TYPE`. Static variable in interface `Constants`

`DEFAULT_RULE_TYPE`. Static variable in interface `Constants`

`DEFAULT_SYSTEM_TYPE`. Static variable in interface `Constants`

F

`finalize()`. Method in class `ParIntFrame`

`FORTRAN`. Static variable in interface `Constants`

`fun1`. Static variable in interface `Constants`

`fun10`. Static variable in interface `Constants`

`fun11`. Static variable in interface `Constants`

`fun12`. Static variable in interface `Constants`

`fun13`. Static variable in interface `Constants`

`fun2`. Static variable in interface `Constants`

`fun3`. Static variable in interface `Constants`
fun4. Static variable in interface Constants
fun5. Static variable in interface Constants
fun6. Static variable in interface Constants
fun7. Static variable in interface Constants
fun8. Static variable in interface Constants
fun9. Static variable in interface Constants

G

getAppWindow(). Method in class ParInt
getDocument(). Method in class ParInt
getLanguageType(). Method in class ParIntDoc
getPQType(). Method in class ParIntDoc
getRuleType(). Method in class ParIntDoc
getSystemType(). Method in class ParIntDoc
ggetWindowDimension(). Method in class ParInt

I

init(). Method in class ParInt

J

JAVA. Static variable in interface Constants

L

languageType. Variable in class ParIntDoc
M

**main(String[])**. Static method in class ParInt

**MPI**. Static variable in interface Constants

N

**NCUBE**. Static variable in interface Constants

P

**paint(Graphics)**. Method in class ParIntFrame

**ParInt()**. Constructor for class ParInt

**ParIntDoc(ParInt)**. Constructor for class ParIntDoc

**ParIntFrame(String, ParInt)**. Constructor for class ParIntFrame

**PQ1**. Static variable in interface Constants

**PQ2**. Static variable in interface Constants

**PQ3**. Static variable in interface Constants

**pgType**. Variable in class ParIntDoc

**PVM**. Static variable in interface Constants

R

**RULE1**. Static variable in interface Constants

**RULE2**. Static variable in interface Constants

**RULE3**. Static variable in interface Constants

**ruleType**. Variable in class ParIntDoc
S

SEQENTIAL. Static variable in interface Constants

setLanguageMenuChecks(int). Method in class ParIntFrame

setLanguageType(int). Method in class ParIntDoc

setPQMenuChecks(int). Method in class ParIntFrame

setPQType(int). Method in class ParIntDoc

setRuleMenuChecks(int). Method in class ParIntFrame

setRuleType(int). Method in class ParIntDoc

setSystemMenuChecks(int). Method in class ParIntFrame

setSystemType(int). Method in class ParIntDoc

systemType. Variable in class ParIntDoc

T

theApp. Variable in class ParIntDoc
Appendix B

Tutorial
**Starting the Tour**

This tour shows you how to use the ParInt user interface. To work through the tour, you need to install Java 1.1 on your machine if it is not installed. Now type

```
java ParInt
```

to start the ParInt user interface.

The following window should appear on the screen. On a different platform Java will produce a different outlook. In this tour, all the pictures are captured from Windows NT. For some reason, if you are running on Solaris then you do not see the text in the middle of the window.

![The ParInt Main Window.](image)

At the start the menu item Show Variable, Compile, and Integrate in the File menu are disabled. You need to select the Number Processes menu item in the System menu, and function and parameters menu items in the Integral Specification in order to enable them.

**Select Number of Processes**

Now let's select the number of processes. Follow these steps:
1. Move your mouse and click on the System Menu.
2. Click on the Number Processes menu item.

Now you should see the Number of Processes window.

Click on the System menu

The Number of Processes Window (NPBox Dialog).

3. Click on the Choice list.
See the next picture for what happens when you click on the choice list.

Choose the number.

Now you have select the number of the processes.

4. Click on the OK button to exit the dialog.

**Select Function**

Now let's select the function from the Integral Specification.
1. Click on the Integral Specification menu.
2. Click on the Function menu item.
3. Click on the "Predefined function" checkbox and use the default function.
4. Click on the OK button and you have selected a function.
Click on the Function menu item.

You can also opt to use a function from a file.

Check the box and input the file name.
Input Parameters

Now let's input parameters for the MPI programs.

1. Click on the Integral Specification menu.
2. Click on the Parameters menu item.
3. Input parameters in those text boxes. Click on the UPDATE button to validate the parameters.
4. Click on the OK button to leave this dialog.

Click on the Parameters menu item.

Click on the UPDATE button to validate the parameters.

You can also input parameters from a file.

Click on checkbox to input the file name.

View Setting
Now the previously disabled menu items in the File menu are enabled. Let's go on to the next step to check the setting we have.

1. Click on the File menu.
2. Click on the Show Variables menu item.

Click on the Show Variables menu item to display the System Variable Configuration window.

Click on the OK button to leave this window.
Now you can see the System Variable Configuration window on the screen. Click on
the button to leave this window.

**Compile Program**

Now let's follow the steps to compile the MPI program.
1. Click on the File menu.
2. Click on the Compile menu item.
3. Click on MAKE to compile the program.

Click on the Compile menu item to display the Compile window.

Click on MAKE to compile the program.

Now you have compiled the program and are ready to run the program. Click on
EXIT to leave the Compile window.

**Run Program**

Now let's follow the steps to run the MPI program.
1. Click on the File menu.
2. Click on the Integrate menu item.
3. Click on Run to run the program.
Click on Integrate menu item to display the Integrate window.

Click on Run to run the program.

Congratulations. Now you have finished the tour and knew how to use the interface.
Select System

When you click on the System menu, you see that the MPI checkbox menu item is marked. That means that the parallel system you are going to use is MPI. In this version, the interface with PVM, nCUBE, and the Sequential version are not implemented.

The MPI checkbox menu item is the default system.

Select Language

When you click on the Language menu you see there is a mark on the C checkbox menu item. That means the program that you are going to run is written in C. In this version, the options Java and FORTRAN are not implemented.
C is checked by default.

View Help System

If you need more information, you can view the help file. Follow these steps:
1. Click on the Help menu.
2. Click on the Help System menu item.

Click on the Help System menu item will invoke Netscape to display more information about the ParInt project.
The ParInt Project

ParInt is a package that lets you integrate any function in parallel.

For more help, click on:

- Introduction
- Author Information

Unused Stuff:

- Decision Tree

It should be noted that all help files are in html format and can be viewed with your favorite web browser like Netscape or Lynx. In fact, the internal help viewer does not support external HTML help and will click on file links from the help files.

ParInt Project Information displayed by the Netscape.

Version Information

To check the version follows these steps:
1. Click on the Help menu.
2. Click on the About menu item.

The About Dialog.
Appendix C

Programs List
The application class is the core class that creates objects at run time. This class defines a method called main() which is necessary for any application. When you run an application the main() method is first executed. Usually this method will cause methods belonging to other classes to be executed. In this Parlnt application class, the main method creates an instance of the Parlnt object then calls the object's init() method to initialize the object.

```java
public class Parlnt
{
    static Parlnt theApp; // The application object
    ParlntFrame window; // The document view
    ParlntDoc document; // The document object
    Dimension wndSize;

    public void init()
    {
        document = new ParlntDoc(this); // Define the document
        window = new ParlntFrame("Parlnt", this); // Create the app window
        Toolkit theKit = window.getToolkit(); // Get the window toolkit
        wndSize = theKit.getScreenSize(); // Get screen size

        window.setBounds(120, 80, 450, 300); // position, Size
        window.addWindowListener(new WindowHandler()); // Add window listener
        window.setVisible(true); // Display the window
        window.requestFocus(); // Request the focus
    }

    public static void main(String[] args)
    {
        theApp = new Parlnt();
        theApp.init();
    }
}
```
Once the users have done with this program, they need to exit
the program. This task is handled by the window events. The
windowListener object is already created to watch the window
events. We need to create a listener class to handle these
events. Class WindowHandler is implemented to do the job. It
extends the interface WindowAdapter class and implements the
windowClosing() method to handle the window closing event.
The method getID() will return the event type that can be
used to test what kind of event is generated. In this case,
the windowClosing method will close the application so it
compares the event type with WINDOW_CLOSING. WINDOW CLOSING
is the window closing event type. In the windowClosing()
method the PaintFrame object calls dispose() method to free
the window resource and System.exit(0) is called to exit the
application.

class WindowHandler extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        if (e.getID() == WindowEvent.WINDOW_CLOSING)
        {
            window.dispose(); // Release the window resources
            System.exit(0); // End the application
        }
    }
}

In order to let other objects can get information from
PaintDoc and PaintFrame objects we need to create
getAppWindow() and getDocument() methods to return them
respectively. The method getWindowDimension() is also defined
in this class to return the screen size so that other objects
do not have to create their own method.

// Get the application window
public PaintFrame getAppWindow()
{
    return window;
}

// Get the document object
public PaintDoc getDocument()
{
    return document;
}

public Dimension getWindowDimension()
{
return wndSize;
}
public class ParlIntFrame extends Frame implements Constants, ActionListener
{
    ParlInt theApp;
    ParlIntDoc document;
    MediaTracker tracker;
    Image mylmage;
    MenuItem var, compile, integrate, np, exit, choosefunction, parameters;
    // PQ menu items
    CheckboxMenuItem pq1, pq2, pq3;
    // rules
    CheckboxMenuItem rule1, rule2, rule3;
    // systems
    CheckboxMenuItem pvm, ncube, mpi, sequential;
    // languages
    CheckboxMenuItem c, java, fortran;

    ConfigBox configBox;
    CompileBox compileBox;
    IntegrateBox integrateBox;
    NPBox npBox;
    FunctionBox functionBox;
    ParaBox paraBox;
    MessageBox aboutBox;
    boolean m_bShowIntegrate;
    boolean m_bShowFunction;
    boolean m_bShowPara;
    boolean m_bShowNP;
    // create menubar
    MenuBar menuBar = new MenuBar();

    public ParlIntFrame(String title, ParlInt theApp)
    {
        super(title);
        this.theApp = theApp;
        document = theApp.getDocument();
        m_bShowIntegrate = false;
        m_bShowFunction = false;
        m_bShowNP = false;
        tracker = new MediaTracker(this);
    
}
Toolkit toolkit = Toolkit.getDefaultToolkit();
myImage = toolkit.getImage("parInt.jpg");
tracker.addImage(myImage, 0);

// create menu
Menu menu1 = new Menu("File");
Menu menu2 = new Menu("Method");
Menu menu3 = new Menu("System");
Menu menu4 = new Menu("Language");
Menu menu5 = new Menu("Integral Specification");
Menu menu6 = new Menu("Help");
Menu menu7 = new Menu("Deterministic");
Menu menu8 = new Menu("Options");
Menu menu9 = new Menu("Adaptive");
Menu menu10 = new Menu("Non-Adaptive");

The "File" menu contains "Show Variables", "Compile", "Integrate", and "Exit ParInt" four menu items. All four menu items are assigned the ParIntFrame object be their action listener by using addActionListener() method. The "Show Variables" menu item is used to pop up a window to show the variables selected by the user from other windows or checkbox menu items. The "Compile" menu item will pop up a window to allow the user to compile the integral function. The "Integrate" menu item will pop up a window to let the user to run the integral function. The "Exit ParInt" menu item allows the user to exit the application.

The "Method" menu contains "Deterministic" and "Monte-Carlo" menu items. The "Deterministic" menu item contains submenu items and the submenu items contain checkbox menu items.

```java
menu9.add(pq1 = new CheckboxMenuitem("Local PQ--No LB", DEFAULT_PQ_TYPE == PQ1));

menu9.add(pq2 = new CheckboxMenuitem("Local PQ with LB", DEFAULT_PQ_TYPE == PQ2));
```
Construct the system pull down menu. The "System" menu contains menu items "PVM", "nCUBE", "MPI", "Sequential", "Number Processes" and "Options". The "Options" menu item contains a "Run PVM Console" submenu.

Construct the language pull down menu. The "Language" menu
item allows the user to choose a language among C, Java, and Fortran languages to integrate the integral function.

```java
    menu4.add(c = new CheckboxMenuitem("C", DEFAULT_LANGUAGE_TYPE == C));
    menu4.add(java = new CheckboxMenuitem("JAVA", DEFAULT_LANGUAGE_TYPE == JAVA));
    menu4.add(fortran = new CheckboxMenuitem("FORTRAN", DEFAULT_LANGUAGE_TYPE == FORTRAN));
    menuBar.add(menu4);

    // Add the language menu listeners
    c.addItemListener(new LanguageCommand(C, document));
    java.addItemListener(new LanguageCommand(JAVA, document));
    fortran.addItemListener(new LanguageCommand(FORTRAN, document));
```

The "Integration Specification" menu bar contains two menu items, "Function" and "Parameters". The "Function" menu item pops up a window which allows the user to select a function to integrate. The "Parameters" menu item pops up a window which allows the user to input parameters which are used for the integration function.

```java
    menu5.add(choosefunction = new MenuItem("Integration Specification"));
    choosefunction.addActionListener(this);
    menu5.add(parameters = new MenuItem("Parameters"));
    parameters.addActionListener(this);
    menuBar.add(menu5);
```

The "Help" menu contains three menu items, "Help System", "Tutorial", and "About". The "Help System" menu item invokes Netscape Browser to display the help files. The "About" menu item displays a simple dialog describing this application.

```java
    MenuItem helpfile = new MenuItem("Help System");
    helpfile.addActionListener(this);
    menu6.add(helpfile);
    MenuItem tutorial = new MenuItem("Tutorial");
    tutorial.addActionListener(this);
    menu6.add(tutorial);
    menu6.addSeparator();
    MenuItem about = new MenuItem("About");
    about.addActionListener(this);
    menu6.add(about);
    menuBar.setHelpMenu(menu6);
```

Sets the menubar for this frame to the specified menubar.

```java
    setMenuBar(menuBar);
```
The `paint()` method. The `Component` class defines the method `paint()`, which is called when the object needs to be drawn. This can occur when the component size changes, or if the component was obscured and becomes visible again for instance. This method is overrode to display an image. The method `drawImage()` is used to show the image.

```java
public void paint(Graphics g) {
    // Draw image at its natural size first.
    g.drawImage(myImage, 50, 50, this);
}
```

Updates the check mark for each checkbox menu item.

```java
public void setLanguageMenuChecks(int languageType) {
    c.setState(C == languageType);
    java.setState(JAVA == languageType);
    fortran.setState(FORTRAN == languageType);
}
```

Updates the check mark for each checkbox menu item.

```java
public void setSystemMenuChecks(int systemType) {
    pvm.setState(PVM == systemType);
    ncube.setState(NCUBE == systemType);
    mpi.setState(MPI == systemType);
    sequential.setState(SEQUENTIAL == systemType);
}
```

Updates the check mark for each checkbox menu item.

```java
public void setRuleMenuChecks(int ruleType) {
    rule1.setState(RULE1 == ruleType);
    rule2.setState(RULE2 == ruleType);
    rule3.setState(RULE3 == ruleType);
}
```

Updates the check mark for each checkbox menu item.

```java
public void setPQMenuChecks(int pqType) {
}
```
pq1.setState(PQ1 == pqType);
pq2.setState(PQ2 == pqType);
pq3.setState(PQ3 == pqType);

/*
There is only one action listener in this class. It is the
method actionPerformed(). The class ParlntFrame must
implements the class ActionListener and we have to implement
the actionPerformed() method. One way to catch the event of
the menu item is by the method getActionCommand() which is a
method of class ActionEvent.

public void actionPerformed(ActionEvent e)
{
    The value returned by the getActionCommand() method is the
label of the menu item. Using the method equals() can
determine are these two strings match. When the value matches
"Exit Parlnt". The dispose() method is called to free the
resource and the exit() method is called to end the
application.

    if (e.getActionCommand().equals("Exit Parlnt"))
    {
        dispose();  // Release the resources
        System.exit(0);  // End the application
    }

When the value matches "About". If the object aboutBox is not
exist, it will be created. Otherwise it is simply set to
visible.

    if (e.getActionCommand().equals("About"))
    {
        if(aboutBox == null)
            aboutBox = new MessageBox(this, "The Parlnt Project", "Parlnt Project
Java version 1.0");
        aboutBox.setVisible(true);
    }

When the value matches "Help System". A Runtime object is
created by calling its getRuntime() method and the process is
created by calling the method exec(). The command we want to
execute can be put into an array then pass it to the exec()
method. Now the command is "netscape index.html". A Netscape
browser will be invoked and show the index file of the help
system.

    else if (e.getActionCommand().equals("Help System"))
    {

When the value matches "tutorial". A Runtime object is created by calling its getRuntime() method and the process is created by calling the method exec(). The command we want to execute can be put into an array then pass it to the exec() method. Now the command is "netscape tour.html". A Netscape browser will be invoked and show the html file.

When the value matches "Number Processes". If the object npBox is not exist it will be created. A variable m_bShowNP is set to true. This variable is used to enable the IntegrationBox object. When m_bShowNP, m_bShowFunction, and m_bShowPara are all set to true then IntegrationBox is enabled.
if (m_bShowFunction && m_bShowPara)
{
    var.setEnabled(true);
    compile.setEnabled(true);
    integrate.setEnabled(true);
}

/****
When the value matches "Function". If the object FunctionBox
is not exit then it is created. The variable m_bShowFunction
is set to true. If the integrateBox exists then the checkbox
for the FunctionBox is checked. If both m_bShowNP and
m_bShowPara are true then enable the menu items "Show
Variable", "Compile", and "Integrate".
*/

else if (e.getActionCommand().equals("Function"))
{
    if(functionBox == null)
        functionBox = new FunctionBox(this, "Function", this.theApp);
    functionBox.setVisible(true);
    m_bShowFunction = true;
    if (integrateBox != null)
    {
        integrateBox.funCheckbox.setState(true);
    }
    if (m_bShowNP && m_bShowPara)
    {
        var.setEnabled(true);
        compile.setEnabled(true);
        integrate.setEnabled(true);
    }
}

/****
When the value matches "Parameters". If the object ParaBox is
not exit then it is created. The variable m_bShowPara is set
to true. If the integrateBox exists then the checkbox for the
ParaBox is checked. If both m_bShowNP and m_bShowFunction are
true then enable the menu items "Show Variable", "Compile",
and "Integrate".
*/

else if (e.getActionCommand().equals("Parameters"))
{
    if(paraBox == null)
        paraBox = new ParaBox(this, "Parameters", this.theApp);
    paraBox.setVisible(true);
    m_bShowPara = true;
    if (integrateBox != null)
    {
        integrateBox.paraCheckbox.setState(true);
    }
    if (m_bShowNP && m_bShowFunction)
{    var.setEnabled(true);    compile.setEnabled(true);    integrate.setEnabled(true);}

/**
 * When the value matches "Show Variable". The object ConfigBox will be created.
 */
else if (e.getActionCommand().equals("Show Variables"))
{
    configBox = new ConfigBox(this, "System Variable Configuration",this.theApp);
    configBox.setVisible(true);
}

/**
 * When the value matches "Compile". The object CompileBox will be created.
 */
else if (e.getActionCommand().equals("Compile"))
{
    compileBox = new CompileBox(this, "Compile",this.theApp);
    compileBox.setVisible(true);
}

/**
 * When the value matches "Integrate". The object IntegrateBox will be created.
 */
else if (e.getActionCommand().equals("Integrate"))
{
    integrateBox = new IntegrateBox(this, "Integrate",this.theApp);
    integrateBox.setVisible(true);
}

/**
 * The finalize() method is called when an object is destroyed. Clean up codes are added here. In our finalize() method all windows are disposed.
 */
protected void finalize()
{
    if(aboutBox != null)
        aboutBox.dispose();
    if(npBox != null)
        npBox.dispose();
}

}
The class ParlIntDoc is a storage storing values needed by the application. These values can be read or changed by methods in the ParlIntFrame class. This class implements the class Constants which will be discussed in next chapter. Methods defined in this class are used to get state or set state of the checkbox menu items on the menu bar. Protected member variables are defined for holding the current status for each checkbox menu item groups.

```java
import java.awt.*;

public class ParlIntDoc implements Constants {
    protected ParlInt theApp; // The application object
    protected int languageType; // Current language type
    protected int systemType; // Current system type
    protected int ruleType; // Current rule type
    protected int pqType; // Current PQ type

    public ParlIntDoc(ParlInt theApp) {
        this.theApp = theApp; // Store the document object
        languageType = DEFAULT_LANGUAGE_TYPE; // Initial language type
        systemType = DEFAULT_SYSTEM_TYPE; // Initial system type
        ruleType = DEFAULT_RULE_TYPE; // Initial rule type
        pqType = DEFAULT_PQ_TYPE; // Initial PQ type
    }

    public int getLanguageType() {
        return languageType;
    }

    public void setLanguageType(int languageType) {
        this.languageType = languageType;
        theApp.getAppWindow().setLanguageMenuChecks(languageType);
    }

    public int getSystemType() {
        return systemType;
    }
}
```
public void setSystemType(int systemType)
{
    this.systemType = systemType;
    theApp.getAppWindow().setSystemMenuChecks(systemType);
}

public int getRuleType()
{
    return ruleType;
}

public void setRuleType(int ruleType)
{
    this.ruleType = ruleType;
    theApp.getAppWindow().setRuleMenuChecks(ruleType);
}

public int getPQType()
{
    return pqType;
}

public void setPQType(int pqType)
{
    this.pqType = pqType;
    theApp.getAppWindow().setPQMenuChecks(pqType);
}
/**
* File Name: Constants.java
* Description: Defines application wide constants.
*/

import java.awt. •;

public interface Constants {
    int PQ1 = 111;
    int PQ2 = 112;
    int PQ3 = 113;
    // rules
    int RULE1 = 108;
    int RULE2 = 109;
    int RULE3 = 110;
    // systems
    int PVM = 104;
    int NCUBE = 105;
    int MPI = 106;
    int SEQUENTIAL = 107;
    // Language type definitions
    int C = 101;
    int JAVA = 102;
    int FORTRAN = 103;
    // Initial conditions
    int DEFAULT_PQ_TYPE = PQ1;
    int DEFAULT_RULE_TYPE = RULE1;
    int DEFAULT_SYSTEM_TYPE = MPI;
    int DEFAULT_LANGUAGE_TYPE = C;

    // Functions
    String fun1 = " f(x) = x0*x1^2*sin(x2)*(4+x3+x4+x5)^-1 * lgamma(x6) " ;
    String fun2 = " f(x) = x2^2 * x3 * exp(x2*x3) / (x0+x1+1)^2 " ;
    String fun3 = " f(x) = 8/(1 + 2(x0+x1+x2)) " ;
    String fun4 = " f(x) = cos(x0+x1+x2+x3+x4) " ;
    String fun5 = " f(x) = sin(10*x1) " ;
    String fun6 = " f(x) = cos(x0+x1) " ;
    String fun7 = " 10dim - f(x) = 1 / (x0+x1+x2) " ;
    String fun8 = " f(x) = 605*x1/((1+120(1-x1))(1+120(1-x1))^2+25x0^2*x1^2) " ;
    String fun9 = " f(x) = 1/(x0^2+0.00001)*(x1+.25)^2+.0001) " ;
    String fun10 = " f(x) = exp(x0 + x1 - 1) " ;
    String fun11 = " Birthweight Data Problem " ;
    String fun12 = " Linear Data Model Problem " ;
    String fun13 = " 10dim - f(x) = l/(sum_i(x_i)^9.1) " ;
}
* Description:

The purpose of the FunctionBox class is to provide a pop up window where the users can select a predefined function or input a user defined function. In order to be a visible window on the screen, a class must be a frame or a dialog, which is derived from the class Window. This class is a dialog so it extends Dialog.

```java
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;
import java.io.*;
```

The FunctionBox class implements the Constants interface to reference the predefined functions. It also implements the ActionListener interface to catch the action event issued by the "OK" button and the "UPDATE" button.

```java
class FunctionBox extends Dialog implements Constants, ActionListener {
    Choice choice2;
    boolean m_bVisible;
    ParIntFrame frame;
    TextField textField;
    Checkbox cbpredef, cbuserdef; //only one of these three can be selected
    CheckboxGroup cbg;
    String m_function;
    Dimension dim;
    Button update;

    Constructor: Most of the work is done in the constructor. The size of the dialog is set to 450 * 300. The dimension of the main window is obtained by the method getBounds() defined in the java.awt.Rectangle class and assigned to bounds. The screen size is obtained by calling the getWindowDimension() method, which is defined in the ParInt class, and assigned to the member variable dim which has type Dimension.

    public FunctionBox(ParIntFrame frame, String title, ParInt theApp) {
        super(frame, title, false); // Create modal dialog
        this.frame = frame;
        Rectangle bounds = frame.getBounds();
        setSize(450,300);
        dim = theApp.getWindowDimension();
    }
```

A calculation is done to put this dialog on a proper position of the screen. If the screen is big enough to show all the windows, the FunctionBox will appear on the right of the main window.
if (bounds.x + bounds.width + 450 < dim.width)
{
    setLocation(bounds.x + bounds.width, bounds.y + 50);
}
else
{
    setLocation(dim.width - 450, bounds.y + 50);
}

A layout manager is created to handle the appearance of the components in the dialog. The essential components are: "MAKE" and "OK" buttons.

GridBagLayout layout = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(layout);

A TextArea object is created to display a message. The top checkbox is created to select predefined functions. The bottom checkbox is created to input the function from a file.

CheckboxGroup cbg = new CheckboxGroup();
Checkbox cbpredef = new Checkbox("Predefined function:", cbg, false);
cbpredef.addItemListener(new CheckboxHandler());
Checkbox cbuserdef = new Checkbox("Input the file name of your function:", cbg, false);
cbuserdef.addItemListener(new CheckboxHandler());

constraints.weighty = 5;
constraints.gridwidth = 5;
constraints.gridheight = 2;
constraints.gridx = 0;
constraints.gridy = 0;
constraints.fill = GridBagConstraints.HORIZONTAL;

constraints.insets = new Insets(0,10,0,10);
TextArea textArea = new TextArea("Click on one of the check boxes to select a function\n" + "or input the file name that contains the function.\n",2,50,TextArea.SCROLLBARS_NONE);
textArea.setEditable(false);
textArea.setBackground(Color.white);
layout.setConstraints(textArea, constraints);
add(textArea);

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 3;
constraints.fill = GridBagConstraints.HORIZONTAL;
The choice2 object is an instance of class java.awt.Choice, which shows a list of predefined functions when it is clicked. The choice2 object is enabled when the top checkbox is checked. It is disabled by default.

```java
choice2 = new Choice();
choice2.addItem(fun1);
choice2.addItem(fun2);
choice2.addItem(fun3);
choice2.addItem(fun4);
choice2.addItem(fun5);
choice2.addItem(fun6);
choice2.addItem(fun7);
choice2.addItem(fun8);
choice2.addItem(fun9);
choice2.addItem(fun10);
choice2.addItem(fun11);
choice2.addItem(fun12);
choice2.addItem(fun13);
choice2.addItemListener(new choice2Listener());
choice2.setEnabled(false);
layout.setConstraints(choice2, constraints);
```

The "UPDATE" button is an instance of Button. It is used to update the current function input by the user.

```java
choice2 = new Choice();
choice2.addItem(fun1);
choice2.addItem(fun2);
choice2.addItem(fun3);
choice2.addItem(fun4);
choice2.addItem(fun5);
choice2.addItem(fun6);
choice2.addItem(fun7);
choice2.addItem(fun8);
choice2.addItem(fun9);
choice2.addItem(fun10);
choice2.addItem(fun11);
choice2.addItem(fun12);
choice2.addItem(fun13);
choice2.addItemListener(new choice2Listener());
choice2.setEnabled(false);
layout.setConstraints(choice2, constraints);
```
update = new Button("UPDATE");
update.setEnabled(false);
update.addActionListener(this); // Add the button listener
layout.setConstraints(update, constraints);
add(update);

/***************************************************************************/
The textField object is an instance of class java.awt.TextField, which lets the users input the file name of the function. It is associated with the bottom checkbox. It is enabled only when the bottom checkbox is checked.
/***************************************************************************/
constraints.weighty = 1;
constraints.gridwidth = 5;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 7;
constraints.fill = constraints.HORIZONTAL;
constraints.insets = new Insets(0,10,0,10);
textField = new TextField(50);
textField.setEnabled(false);
layout.setConstraints(textField, constraints);
add(textField); // Add the table

/***************************************************************************/
The Button object "OK" is created to allow the user to hide the dialog box.
/***************************************************************************/
constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 12;
constraints.fill = constraints.NONE;
Button button = new Button("OK"); // The OK button
button.addActionListener(this); // Add the button listener
layout.setConstraints(button, constraints);
add(button);
m_bVisible = true;

/***************************************************************************/
The method actionPerformed() is the action listener for buttons "UPDATE" and "OK".
***************************************************************************/
public void actionPerformed(ActionEvent e)
{
When the button "UPDATE" is clicked, the getTextO method of the object textField is called to get the file name from the text the file will be open to read. The file "userfcn.c" is created by appending "userfcn.tmp" and the user file.

***************/

if (e.getActionCommand().equals("UPDATE")
{
    m_function = ";
    int err = 0;
    try
    {

/*****************************/

Open output file "userfcn.c" and input files "userfcn.tmp" and user file.

/*****************************/

BufferedWriter out = new BufferedWriter(
    new FileWriter("userfcn.c"));
FileReader template = new FileReader("userfcn.tmp");
FileReader userfcn = new FileReader(textField.getText());

boolean EOF = false;
int c;

/***************************/

Append "userfcn.tmp" to "userfcn.c".

/*****************************/

while (!EOF)
{
    try
    {
        c = template.read();
        if (c == -1)
        {
            out.write("\n");
            EOF = true;
        }
        else
            out.write(c);
    }
    catch (EOFException e1)
    {
        EOF = true;
    }
    catch (IOException e1)
    {
        EOF = true;
        err = 1;
        MessageBox errBox = new
        MessageBox(this.frame,"Error message","Error reading input file.");
        errBox.setVisible(true);
    }
}
template.close();

/**********************
Append user file to "userfcn.c".
**************************/
EOF = false;
boolean found = false;
while (!EOF)
{
    try
    {
        c = userfcn.read();
        while (c != -1 && found == false && c != 123)
        {
            c = userfcn.read();
        }
        if (c == -1)
        {
            out.write("\n");
            EOF = true;
        }
        else if (found == true)
        {
            out.write(c);
            found = true;
        }
    } catch (EOFException e1)
    {
        EOF = true;
    } catch (IOException e1)
    {
        EOF = true;
        MessageBox errBox = new MessageBox(this.frame, "Error message", "Cann't read input file " + textField.getText() + ".");
        errBox.setVisible(true);
    }
}

out.close();
userfcn.close();
}
catch (IOException ioe)
{ err = 1;
    MessageBox errBox = new MessageBox(this.frame, "Error message", "Cann't open input file " + textField.getText() + " or userfcn.tmp.");
    errBox.setVisible(true);
}
if (err==0) update.setLabel("UPDATED");
}
When the button "OK" is clicked, the member variable m_bVisible is set to false. The m_bVisible is provided for the object integrateBox to determine whether the functionBox object is displayed or hidden.

```java
else {
    m_bVisible = false;
    if (frame.integrateBox != null)
    {
        frame.integrateBox.funCheckbox.setState(false);
    }
    setVisible(false); // Hide the dialog
}
```

The listener choice2Listener is associated with choice2. When the object choice2 is clicked and the selected item has been changed, then the listener will execute the method itemStateChanged(). In this method, the new item is also assigned to the member variable m_function.

```java
class choice2Listener implements ItemListener
{
    public void itemStateChanged(ItemEvent e)
    {
        m_function = choice2.getSelectedItem();
    }
}
```

// Return the selected function.
public String getFunction()
{
    return m_function;
}

// Get the status of visibility.
public boolean getVisible()
{
    return m_bVisible;
}

The listener CheckboxHandler monitors the events triggered by the checkboxes. When one of the checkboxes is checked, the method itemStateChanged() will be started. In the itemStateChanged() method, the getSelectedCheckbox() method is used to get the selected checkbox, to enable associated objects and disable not associated objects.

```java
class CheckboxHandler implements ItemListener
```
{  
    public CheckboxHandler()
    {
    
    }
    public void itemStateChanged(ItemEvent e)
    {
        if (cbg.getSelectedCheckbox() == cbpredef)
        {
            m_function = choice2.getSelectedItem();
            choice2.setEnabled(true);
            textField.setEnabled(false);
            update.setEnabled(false);
            update.setLabel("UPDATE");
        }
        else if (cbg.getSelectedCheckbox() == cbuserdef)
        {
            m_function = textField.getText();
            choice2.setEnabled(false);
            textField.setEnabled(true);
            update.setLabel("UPDATE");
            update.setEnabled(true);
        }
    }
}
The purpose of this ParaBox class is to provide a pop up window where the users can input the error tolerances, limit on the number of function evaluations, and select an integration rule. These parameters must be set for the first time to integrate the integrand function.

```java
import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

class ParaBox extends Dialog implements Constants, ActionListener {
    boolean m_bVisible;
    ParIntFrame frame;
    TextField toleranceField, r_toleranceField, ruleField, limitField;
    TextField inputField;
    TextField dimField, lowField, upField;
    String m_a_tolerance, m_r_tolerance, m_rule, m_limit, m_filename;
    String m_dimension;
    Dimension dim;
    Checkbox inputCheckbox;
    Button button;
    Choice lowChoice, upChoice;
    Button defButton, update, addLowButton, addUpButton, lowReset, upReset;

    public ParaBox(ParIntFrame frame, String title, ParInt theApp) {
        super(frame, title, false); // Create modal dialog
        this.frame = frame;
        m_bVisible = true;
    }

    The size of the dialog is set to 550 by 400. The dimension of
the main window is obtained by the method getBounds() defined in the java.awt.Rectangle class and assigned to bounds. The screen size is obtained by calling the getWindowDimension() method which is defined in the ParnInt class and assigned to the member variable dim which has type Dimension.

```java
Rectangle bounds = frame.getBounds();
setSize(550,400);
dim = theApp.getWindowDimension();
```

A calculation is done to put this dialog at a proper position in the screen. If the screen is big enough to show all the windows, the ParaBox will appear on the right of the main window.

```java
if (bounds.x + bounds.width + 450 < dim.width)
{
    setLocation(bounds.x + bounds.width, bounds.y + 80);
}
else
{
    setLocation(dim.width - 450 , bounds.y + 80);
}
```

A layout manager is created to handle the appearance of the components in the dialog.

```java
GridBagLayout layout = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(layout);
```

A TextArea object is created to display the information regarding the four rules.

```java
constraints.weighty = 10;
constraints.gridwidth = 10;
constraints.gridheight = 2;
constraints.gridx = 0;
constraints.gridy = 0;
constraints.fill = constraints.NONE;
constraints.insets = new Insets(0,10,0,10);
TextArea textArea = new TextArea("About rules:
"+
"1. gives the user a 2 dimensional degree 13 integration rule that uses 65 evaluation points.
"+
"This is a recommended choice in two dimensions.
"+
"2. gives the user a 3 dimensional degree 11 integration rule that uses 127 evaluation points.
"+
"This is a recommended choice in three dimensions.
"+
"3. gives the user a degree 9 integration rule.
"+
"This rule is recommended for oscillatory problems.
" +
```
"4. gives the user a degree 7 integration rule.
This is the recommended general rule.
" ,5,60,TextArea.SCROLLBARS_VERTICAL_ONLY);
textArea.setEditable(false);
textArea.setBackground(Color.white);
layout.setConstraints(textArea, constraints);
add(textArea);

Four labels explain which values can be input into the text
fields on their right side.

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 4;
constraints.fill = constraints.HORIZONTAL;
Label label1 = new Label("Absolute tolerance: ");
layout.setConstraints(label1, constraints);
add(label1); // Add the lable

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 4;
constraints.fill = constraints.NONE;
toleranceField = new TextField("0.000001", 20);
toleranceField.setEnabled(true);
layout.setConstraints(toleranceField, constraints);
add(toleranceField); // Add the lable

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 5;
constraints.fill = constraints.HORIZONTAL;
Label label2 = new Label("Relative tolerance: ");
layout.setConstraints(label2, constraints);
add(label2); // Add the lable

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 5;
constraints.fill = constraints.NONE;
r_toleranceField = new TextField("0.000001", 20);
r_toleranceField.setEnabled(true);
layout.setConstraints(r_toleranceField, constraints);
add(r_toleranceField); // Add the label

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 6;
constraints.fill = constraints.HORIZONTAL;
Label label3 = new Label("Cubature rule: ");
layout.setConstraints(label3, constraints);
add(label3); // Add the label

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 6;
constraints.fill = constraints.NONE;
ruleField = new TextField("4", 20);
ruleField.setEnabled(true);
layout.setConstraints(ruleField, constraints);
add(ruleField);

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 7;
constraints.fill = constraints.HORIZONTAL;
Label label4 = new Label("Fn evaluation limit: ");
layout.setConstraints(label4, constraints);
add(label4);

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 7;
constraints.fill = constraints.NONE;
limitField = new TextField("400000", 20);
limitField.setEnabled(true);
layout.setConstraints(limitField, constraints);
add(limitField); // Add the label

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 8;
constraints.fill = constraints.HORIZONTAL;

Label label5 = new Label("Dimension: "); // Label to hold message
layout.setConstraints(label5, constraints);
add(label5);

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 8;
constraints.fill = constraints.NONE;

dimField = new TextField(20);
dimField.setEnabled(true);
layout.setConstraints(dimField, constraints);
add(dimField);

constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 9;
constraints.fill = constraints.HORIZONTAL;

constraints.insets = new Insets(0,10,0,10);
lowField = new TextField(10);
lowField.setEnabled(true);
layout.setConstraints(lowField, constraints);
add(lowField);          // Add the table

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 1;
constraints.gridy = 9;
constraints.fill = constraints.NONE;
addLowButton = new Button("Add Lower Bound =>");    // The OK button
addLowButton.setEnabled(true);
addLowButton.addActionListener(this);  // Add the button listener
layout.setConstraints(addLowButton, constraints);
add(addLowButton);

constraints.weightx = 1;
constraints.weighty = 2;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 3;
constraints.gridy = 9;
constraints.fill = constraints.HORIZONTAL;
lowChoice = new Choice();
lowChoice.setEnabled(true);
layout.setConstraints(lowChoice, constraints);
add(lowChoice);

constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 5;
constraints.gridy = 9;
constraints.fill = constraints.NONE;
lowReset = new Button("0");
lowReset.addActionListener(this); // Add the button listener
layout.setConstraints(lowReset, constraints);
add(lowReset); // Add the table

constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 10;
constraints.fill = constraints.HORIZONTAL;
constraints.insets = new Insets(0,10,0,10);

upField = new TextField(10);
upField.setEnabled(true);
layout.setConstraints(upField, constraints);
add(upField);

constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 1;
constraints.gridy = 10;
constraints.fill = constraints.NONE;
addUpButton = new Button("Add Upper Bound =>");
addUpButton.setEnabled(true);
addUpButton.addActionListener(this); // Add the button listener

layout.setConstraints(addUpButton, constraints);
add(addUpButton);
constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 3;
constraints.gridy = 10;
constraints.fill = constraints.HORIZONTAL;
upChoice = new Choice();
upChoice.setEnabled(true);
layout.setConstraints(upChoice, constraints);
add(upChoice);

constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 5;
constraints.gridy = 10;
constraints.fill = constraints.NONE;
upReset = new Button("0");
A checkbox is created to let the user can only choose to input the parameters or to select a file which already contains these parameters.

```java
constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 11;
constraints.fill = constraints.NONE;

inputCheckbox = new Checkbox("Input file name: ", false);
inputCheckbox.addActionListener(new CheckboxHandler());
layout.setConstraints(inputCheckbox, constraints);
add(inputCheckbox);
```

The text field on its right side is used to input the file name.

```java
constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 1;
constraints.gridy = 11;
constraints.fill = constraints.NONE;

inputField = new TextField("", 20);
inputField.setEnabled(false);
layout.setConstraints(inputField, constraints);
add(inputField); // Add the table
```

The "UPDATE" button is used to update the parameters, which are input by the user or from a file.

```java
Button defButton = new Button("DEFAULT"); // The DEFAULT button
defButton.addActionListener(this); // Add the button listener
```

```java
layout.setConstraints(defButton, constraints);
add(defButton); // Add the button
```
constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 1;
constraints.gridy = 12;
constraints.fill = constraints.NONE;

Button update = new Button("UPDATE");
update.addActionListener(this);
layout.setConstraints(update, constraints);
add(update);

The "OK" button is used to hide the dialog. The "OK" button is disabled by default. It will not be enabled until the "UPDATE" button is clicked. This ensures the parameters to exist.

button = new Button("OK");
button.setEnabled(false);
button.addActionListener(this);
layout.setConstraints(button, constraints);
add(button);

The method actionPerformed() is the action listener for buttons "UPDATE", "DEFAULT" and "OK".

When the "DEFAULT" button is clicked. The default parameters for the selected function will be filled in the lowChoice and upChoice choices.

if (e.getActionCommand().equals("DEFAULT") && (frame.functionBox != null))
{
    String str1 = frame.functionBox.getFunction();
    dimField.setText(""");
    lowReset.setLabel("0");
    lowChoice.removeAll();
    upReset.setLabel("0");
    upChoice.removeAll();
}
int j;

if (strl == fun1)
{
    dimField.setText("8");
    for(j=0; j<3; j++)
    {
        lowChoice.addItem("0");
    }
    upChoice.addItem("2");
    upChoice.addItem("1");
    upChoice.addItem("1.570796327");
    for(j=3; j<6; j++)
    {
        lowChoice.addItem("-1");
        upChoice.addItem("1");
    }
    for(j=6; j<8; j++)
    {
        lowChoice.addItem("0.0");
        upChoice.addItem("1.0");
    }
    lowReset.setLabel("8");
    upReset.setLabel("8");
}
else if (strl == fun2)
{
    dimField.setText("4");
    for(j=0; j<3; j++)
    {
        lowChoice.addItem("0");
        upChoice.addItem("1");
    }
    lowChoice.addItem("0");
    upChoice.addItem("2");
    lowReset.setLabel("4");
    upReset.setLabel("4");
}
else if (strl == fun3)
{
    dimField.setText("3");
    for(j=0; j<3; j++)
    {
        lowChoice.addItem("0");
        upChoice.addItem("1");
    }
    lowReset.setLabel("3");
    upReset.setLabel("3");
}
else if (strl == fun4)
{
    dimField.setText("5");
}
for(j=0; j<4; j++)
{
    lowChoice.addltem("0");
    upChoice.addltem("3.1415926536");
}

lowChoice.addltem("0");
upChoice.addltem("1.570796327");
lowReset.setLabel("5");
upReset.setLabel("5");
else if (str1 == fun5)
{
    dimField.setText("4");
    for(j=0; j<4; j++)
    {
        lowChoice.addltem("0");
        upChoice.addltem("1");
    }
    lowReset.setLabel("4");
    upReset.setLabel("4");
}
else if (str1 == fun6)
{
    dimField.setText("2");
    for(j=0; j<2; j++)
    {
        lowChoice.addltem("0");
        upChoice.addltem("9.424777961");
    }
    lowReset.setLabel("2");
    upReset.setLabel("2");
}
else if (str1 == fun7)
{
    dimField.setText("10");
    for(j=0; j<10; j++)
    {
        lowChoice.addltem("0");
        upChoice.addltem("1");
    }
    lowReset.setLabel("10");
    upReset.setLabel("10");
}
else if (str1 == fun8)
{
    dimField.setText("2");
    for(j=0; j<2; j++)
    {
        lowChoice.addltem("0");
        upChoice.addltem("1");
    }
    lowReset.setLabel("2");
upReset.setLabel("2");
}
else if (str1 == fun9)
{
dimField.setText("2");
for(j=0; j<2; j++)
{
    lowChoice.addItem("0");
    upChoice.addItem("1");
}
lowReset.setLabel("2");
upReset.setLabel("2");
}
else if (str1 == fun10)
{
dimField.setText("2");
for(j=0; j<2; j++)
{
    lowChoice.addItem("0");
    upChoice.addItem("1");
}
lowReset.setLabel("2");
upReset.setLabel("2");
}
else if (str1 == fun11)
{
dimField.setText("7");
for(j=0; j<7; j++)
{
    lowChoice.addItem("0");
    upChoice.addItem("1");
}
lowReset.setLabel("7");
upReset.setLabel("7");
}
else if (str1 == fun12)
{
dimField.setText("10");
for(j=0; j<10; j++)
{
    lowChoice.addItem("0");
    upChoice.addItem("1");
}
lowReset.setLabel("10");
upReset.setLabel("10");
}
else if (str1 == fun13)
{
dimField.setText("10");
for(j=0; j<10; j++)
{
    lowChoice.addItem("0");
    upChoice.addItem("1");
}
The purpose of this NPBox class is to provide a pop up window where
the users can select the number of processes to calculate the
integral. This class extends the java.awt.Dialog class. This class
also implements the java.awt.event.ItemListener interface to receive
item events; and implements the java.awt.event.ActionListener
interface to receive action events.

/* File Name: NPBox.java
 * Description:
 * The purpose of this NPBox class is to provide a pop up window where
 * the users can select the number of processes to calculate the
 * integral. This class extends the java.awt.Dialog class. This class
 * also implements the java.awt.event.ItemListener interface to receive
 * item events; and implements the java.awt.event ActionListener
 * interface to receive action events.*/

import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

class NPBox extends Dialog implements ActionListener, ItemListener //AdjustmentListener
{
    Label label1;
    Choice choice;

    static int np;
    public boolean m_bVisible;
    ParIntFrame frame;
    Dimension dim;

    public NPBox(ParIntFrame frame, String title, ParInt theApp)
    {
        super(frame, title, false); // Create modal dialog
        this.frame = frame;
        m_bVisible = true;
        Rectangle bounds = frame.getBounds();
        setSize(450,300);
        dim = theApp.getWindowDimension();

        A calculation is done to put this dialog at a proper location
        in the screen. If the screen is big enough to show all the
        windows, the NPBox will appear on the right of the main
        window.

        if (bounds.x + bounds.width + 450 < dim.width)
        {
            setLocation(bounds.x + bounds.width, bounds.y + 110);
        }
    }
}
else
{
    setLocation(dim.width - 450, bounds.y + 110);
}

veyor_manager is created to handle the appearance of the components in the dialog. A Label object is created to display the number of processes. Another static Label object shows "Please select the number of dimensions of processes to use". A Choice object is created to let the user select the number of dimension. A TextArea object is created to display some additional information. A Button object "OK" is created to exit the dialog box.

GridBagLayout layout = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(layout);
constraints.gridx = 0;
constraints.weightx = constraints.weighty = 10.0;
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.ipadx = 30;
constraints.ipady = 5;
constraints.insets = new Insets(0,10,0,10);
label1 = new Label("Number of processes: 1"); // Label to hold message
layout.setConstraints(label1, constraints);
add(label1);

constraints.gridx = 0;
constraints.gridy = 1;
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.ipadx = 30;
constraints.ipady = 5;
Label label2 = new Label("Please select the number of dimensions of processes to use.");
layout.setConstraints(label2, constraints);
add(label2);

constraints.gridx = 0;
constraints.gridy = 2;
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.ipadx = 30;
constraints.ipady = 0;
constraints.insets = new Insets(0,12,0,12);
choice = new Choice();
choice.addItem("0");
choice.addItem("1");
choice.addItem("2");
choice.addItem("3");
choice.addItem("4");
choice.addltem("5");
choice.addltem("6");
choice.addltem("7");
choice.addltem("8");
choice.addltem("9");
choice.addltem("10");
choice.addltem("11");
choice.addltem("12");
choice.addltemListener(this);
layout.setConstraints(choice, constraints);

add(choice);

constraints.gridx = 0;
constraints.gridy = 3;
constraints.fill = constraints.NONE;
constraints.ipadx = 30;
constraints.ipady = 5;
constraints.insets = new Insets(0,10,0,10);
TextArea textarea = new TextArea("Note: Most of the procedures for area division only \n" + "support process quantities in power of 2. Thus, for ease,\n" + "we impose this restriction in this version of\n" + "ParInt,\n","5,TextArea.SCROLLBARS_NONE);
textArea.setEditable(false);
textArea.setBackground(Color.white);
layout.setConstraints(textArea, constraints);
add(textArea);

constraints.gridx = 0;
constraints.gridy = 4;
constraints.fill = constraints.NONE;
constraints.ipadx = 30;
constraints.ipady = 5;
Button button = new Button(" OK "); // The OK button
button.addActionListener(this); // Add the button listener
layout.setConstraints(button, constraints);

add(button);

The method actionPerformed() is the action listener for
business "OK". When the button "OK" is clicked, the member
variable m_bVisible is set to false. The m_bVisible is
provided for the object integrateBox to determine whether the
paraBox object is disp-layed or hidden. If the IntegrateBox
object exist, then the statement
frame.integrateBox.npCheckBox.setState(false) will be
extended to update the current visibility of the npBox object
for the integrateBox object. The statement setVisible(false)
hides the npBox object.
*********************************************************************/

public void actionPerformed(ActionEvent e)
```java
m_bVisible = false;
if (frame.integrateBox != null)
{
    frame.integrateBox.npCheckbox.setState(false);
}
setVisible(false);  // Hide the dialog
/**
 * The npBox object is also an item listener. The method
 * itemStateChanged() is implemented to catch the events
 * generated by the choice object. When the choice object is
 * clicked, then the index of the current item will be obtained
 * and assigned to the member variable np. The value 2 to the
 * power of np will be shown on the labell object.
 */
public void itemStateChanged(ItemEvent e)
{
    np = choice.getSelectedIndex();
    labell.setText("Number of processes: " + (int) Math.pow(2.0, (double) np));
}
/**
 * The getValue() method returns the square of the member
 * variable np.
 */
public int getValue()
{
    return (int) Math.pow(2.0, (double) np);
}
/**
 * The getVisible() method returns the status of visibility.
 */
public boolean getVisible()
{
    return m_bVisible;
}
```
This class displays information gathered from other objects. The current integration system and programming language are obtained from the object window. The number of processes is obtained from the object npBox and the current function is obtained from the object functionBox. A pop up window is provided to show this information. This class extends the java.awt.Dialog class to be a visible dialog and implements the ActionListener interface to receive action events for the "OK" button.

import java.awt.*;
import java.awt.event.*;
import java.lang.Math;

class ConfigBox extends Dialog implements ActionListener
{
    Parlnt theApp;
    ParlntDoc document;
    Label label1;
    String[] languageType = { "C", "JAVA", "FORTRAN" };  
    String[] systemType = { "PVM", "NCUBE", "MPI", "SEQUENTIAL" }; 
    String str;
    int index;
    Dimension dim;

    Constructor: This dialog is created as a modeless dialog by passing the Boolean value false to the method super().

    public ConfigBox(ParlntFrame frame, String title, Parlnt theApp)
    {
        super(frame, title, false); // Create modal dialog
        this.theApp = theApp;
        document = theApp.getDocument();

        Rectangle bounds = frame.getBounds();
        setSize(550,300);
        setBackground(Color.white);
        dim = theApp.getWindowDimension();

        A calculation is done to put this dialog on a proper
position of the screen. If the screen is big enough to show all the windows, the CompileBox will appear on the left of the main window.

```java
if (450 > bounds.x)
{
    setLocation(0, bounds.y + 50);
}
else
{
    setLocation(bounds.x - 450 , bounds.y + 50);
}
```

A layout manager is created to handle the appearance of the components in the dialog. The essential components are: label1 to label2 and "OK" buttons.

```java
GridBagLayout layout = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(layout);
```

The dialog box is divided into two parts. The left column contains the description, the right column the value.

```java
constraints.weightx = 5;
constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 0;
constraints.fill = GridBagConstraints.BOTH;
constraints.insets = new Insets(10, 10, 10, 10);
```

```java
label1 = new Label("Description", Label.CENTER); // Label to hold message
label1.setBackground(Color.white);
layout.setConstraints(label1, constraints);
add(label1); // Add the label
```

```java
constraints.weightx = 5;
constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 0;
constraints.fill = GridBagConstraints.BOTH;
constraints.insets = new Insets(10, 10, 10, 10);
```

```java
Label label2 = new Label("Value", Label.CENTER); // Label to hold message
label2.setBackground(Color.white);
layout.setConstraints(label2, constraints);
add(label2);
```

```java
constraints.weightx = 5;
```
In order to get the value, we need to get the id from the method getSystemType() defined in the ParIntDoc class. Then we have to define a string array for all the system types corresponding to the id numbers. Finally, the id can be used as an index into the string array to get the current system type.

```
/**
* In order to get the value, we need to get the id from the
* method getSystemType() defined in the ParIntDoc class.
* Then we have to define a string array for all the system
* types corresponding to the id numbers. Finally, the id can
* be used as an index into the string array to get the current
* system type.
*/

index = document.getSystemType();
```

```java
constraints.weightx = 5;
constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 2;
constraints.fill = constraints.BOTH;
constraints.insets = new Insets(0,15,0,15);
label4 = new Label(systemType[index - 104], Label.CENTER);
label4.setBackground(Color.white);
layout.setConstraints(label4, constraints);
add(label4); // Add the table
```

The value of the number of processes is obtained by calling the getValue() method defined in npBox object. We can not call it directly. It must be called by the frame object.
because the npBox object is created in the frame object.
The value is an integer number and it is shown as a caption.
The caption has to be a string so it is converted into a
string by the method Integer.toString().

try
{
    index = frame.npBox.getValue();
}

catch (NullPointerException e)
{
    index = 1;
}

constraints.weightx = 5;
constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 3;
constraints.fill = constraints.BOTH;
constraints.insets = new Insets(0,15,0,15);
Label label6 = new Label(Integer.toString(index), Label.CENTER);
label6.setBackground(Color.white);
layout.setConstraints(label6, constraints);
add(label6); // Add the label

constraints.weighty = 10;
constraints.gridx = 0;
constraints.weightx = 5;
constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 4;
constraints.fill = constraints.BOTH;
constraints.insets = new Insets(0,15,0,15);
Label label7 = new Label("Programming language:");
label7.setBackground(Color.white);
layout.setConstraints(label7, constraints);
add(label7); // Add the label

In order to get the language type, we need to get the id
from the method getLanguageType() defined in the ParlntDoc
class. Then we have to define a string array for all the system
types corresponding to the id numbers. Finally, the id can
be used as an index into the string array to get the current
system type.

index = document.getLanguageType();
constraints.weightx = 5;
constraints.weighty = 1;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 4;
constraints.fill = constraints.BOTH;
constraints.insets = new Insets(0,15,0,15);
Label label8 = new Label(languageType[index - 101], Label.CENTER);

label8.setBackground(Color.white);
layout.setConstraints(label8, constraints);
add(label8); // Add the label

constraints.weighty = 10;
constraints.gridx = 0;
constraints.weightx = 5;
constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 5;
constraints.fill = constraints.BOTH;
constraints.insets = new Insets(0,15,0,15);

Label label11 = new Label("Current Function: ");
label11.setBackground(Color.white);
layout.setConstraints(label11, constraints);
add(label11); // Add the label

try
{
    str = frame.functionBox.getFunction();
    if (str.compareTo("") == 0)
    {
        str = "User defined function";
    }
}
catch (NullPointerException e)
{
    System.out.println("Failed to get function string.");
}

constraints.weightx = 5;
constraints.weighty = 1;
constraints.gridwidth = 2;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 5;
constraints.fill = constraints.BOTH;
constraints.insets = new Insets(0,15,0,15);
Label label12 = new Label(str, Label.CENTER);
label12.setBackground(Color.white);
layout.setConstraints(label12, constraints);
add(label12);       // Add the label

constraints.weighty = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.fill = constraints.NONE;
constraints.gridx = 2;
constraints.gridy = 7;
Button button = new Button("OK");  // The OK button
button.addActionListener(this);     // Add the button listener
layout.setConstraints(button, constraints);
add(button);

}/*-----------------------------------------------*/

The method actionPerforrmed() is the action listener for "OK"
button. When the "OK" button is clicked, the method
dispose() is called to free the resource and exit the
dialog.
*/

public void actionPerforrmed(ActionEvent e)
{
    dispose();
}
/** File Name: CompileBox.java */
/** Description: A dialog providing an interface to compile MPI programs. */
*****************************************************************************
import java.awt.*;
import java.awt.event.*;
import java.lang.*;
import java.io.*;
class CompileBox extends Dialog implements ActionListener {
    ParInt theApp;
    ParIntDoc document;
    String[] systemType = { "PVM", "NCUBE", "MPI", "SEQUENTIAL" };
    Label label1, label2;
    Button makeButton, okButton;
    String str, str1;
    int index, index1;
    TextArea textArea;

    /******************************************************************************/
    Constructor: This dialog is created as a modeless dialog by passing the Boolean value false to the method super().
    ******************************************************************************/
    public CompileBox(ParIntFrame frame, String title, ParInt theApp) {
        super(frame, title, false); // Create modal dialog
        this.theApp = theApp;
        document = theApp.getDocument();

        /******************************************************************************/
        Get the frame's size and set the background color.
        ******************************************************************************/
        Rectangle bounds = frame.getBounds();
        setSize(450, 300);
        setBackground(Color.white);

        /******************************************************************************/
        A calculation is done to put this dialog on a proper position of the screen. If the screen is big enough to show all the windows, the CompileBox will appear on the left of the main window.
        ******************************************************************************/
        if (450 > bounds.x) {
            setLocation(0, bounds.y + 80);
        } else {
            setLocation(bounds.x - 450, bounds.y + 80);
        }
A layout manager is created to handle the appearance of
the components in the dialog. The essential components
are: "MAKE" and "OK" buttons.

```java
GridBagLayout layout = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(layout);

// Construct a label managed by the layout manager.
constraints.weighty = 5;
constraints.ipadx = 30;
constraints.ipady = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.gridx = 0;
constraints.gridy = 0;
label1 = new Label("Click on "MAKE" to compile and link the integrand function.",
Label.CENTER); // Label to hold message
    label1.setBackground(Color.white);
    add(label1);

// Construct the "MAKE" button managed by the layout manager.
constraints.weighty = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.gridx = 1;
constraints.gridy = 2;
makeButton = new Button("MAKE");
makeButton.addActionListener(this);
    layout.setConstraints(makeButton, constraints);
    add(makeButton);

// Construct a text area managed by the layout manager.
constraints.weighty = 5;
constraints.gridwidth = 3;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 4;
constraints.fill = GridBagConstraints.HORIZONTAL;
constraints.ipadx = 30;
constraints.ipady = 5;
constraints.insets = new Insets(0,10,0,10);
textArea = new TextArea("Message: ",4,40);
textArea.setEditable(false);
textArea.setBackground(Color.white);
    layout.setConstraints(textArea, constraints);
    add(textArea);

// Construct the "EXIT" button managed by the layout manager.
```
constraints.weighty = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.fill = constraints.HORIZONTAL;
constraints.gridx = 1;
constraints.gridy = 8;
okButton = new Button("EXIT");
okButton.addActionListener(this);
layout.setConstraints(okButton, constraints);
add(okButton);

public void actionPerformed(ActionEvent e)
{
    // When "MAKE" button is clicked.
    if (e.getActionCommand().equals("MAKE"))
    {
        /*--------------------------------------------
         * The Runtime object r is used to create a process
         * and run the "Makefile". We need to specify the
         * command to be executed. A String object cmdarray[]
         * is created to store the command we wish to execute.
         * To run the process we need to put it in a try block
         * so we can catch the exceptions.
         *--------------------------------------------------*/
        Runtime r = Runtime.getRuntime();
        String cmdarray[] = {"make", "all"};
        try
        {
            /*--------------------------------------------
             * The method exec() of the r object creates the
             * process p which runs the "Makefile". Both "MAKE"
             * and "EXIT" buttons are disabled and wait until
             * the process finishes running. After the process
             * is done it is destroyed and the "MAKE" and "EXIT"
             * buttons are back enabled.
             *--------------------------------------------------*/
            Process p = r.exec(cmdarray);
            makeButton.setLabel("WAIT...");
            makeButton.setEnabled(false);
            okButton.setEnabled(false);
            p.waitFor();
            p.destroy();
            makeButton.setLabel("MAKE");
            makeButton.setEnabled(true);
            okButton.setEnabled(true);
        }
    }
}
Two exceptions must be caught. One is IOException which signals that an I/O exception of some sort has occurred. Another is InterruptedException occurred when a thread is interrupted by an interrupt method in class Thread.

```java
// Code snippet
```

To read the file, a FileReader object in is created to open the file. A while loop keeps reading the file until an EOF flag is met. The read() method of the in object is used to read the file. We also need a try block for the read and two catch blocks to catch the EOFException and IOException.

```java
try {
    FileReader in = new FileReader("report");
    boolean EOF = false;
    int c;

    while (!EOF) {
        try {
            c = in.read();
            if (c == -1) {
                EOF = true;
                textArea.append(str);
            } else {
                str += (char) c;
            }
        } catch (EOFException e2) {
            EOF = true;
        } catch (IOException e2) {
            EOF = true;
            System.out.println("Error reading input file" + e2);
        }
    }
} catch (IOException exception) {
    System.err.println("Command failed with IO Exception");
} catch (InterruptedException iexception) {
    System.err.println("Command failed with Interrupted Exception");
}
```
catch(FileNotFoundException e3)
{
    System.out.println("Error reading input file" + e3);
}

/************************************
The "EXIT" button is clicked and this object is destroyed
and the resource is released.
*************************************/
else
    dispose();
}
The IntegrateBox class collects the information to run the integration program. The information includes the number of processes, the absolute and relative tolerances, the maximum number of integrand evaluation (limit) and the integrand function. A textArea object is derived from the TextArea class to display the messages that are output by the MPI program. Three checkboxes are used to show/hide the functionBox object, the paraBox object or the npBox object. The button with caption "Run" is used to run the integral computation. The button with caption "Exit" is used to destroy the integralBox object.

```java
import java.awt.*;
import java.awt.event.*;
import java.lang.*;
import java.io.*;

class IntegrateBox extends Dialog implements Constants, ActionListener {
    Parlnt theApp;
    ParlntDoc document;
    ParlntFrame frame;
    String[] systemType = { "PVM", "NCUBE", "MPI", "SEQUENTIAL" };
    Label label1, label2;
    Button makeButton, okButton, ClsButton;
    int index, index1, np_value, pq;
    TextArea textArea;
    Checkbox funCheckbox, npCheckbox, paraCheckbox;
    boolean state;
    String fileName = "datain";
    String str, str1, cmdarray;
    String m_a_tolerance, m_r_tolerance, m_rule, m_limit, m_dimension;
    Dimension dim;
    Choice lowBound, upBound;

    Constructor: The class integrateBox is derived from the class Dialog and set as a modeless dialog by passing the Boolean value false to the method super().
    public IntegrateBox(ParlntFrame frame, String title, Parlnt theApp) {
        super(frame, title, false); // Create modal dialog
    }
```
this.theApp = theApp;
document = theApp.getDocument();
this.frame = frame;

/***********************************************************/
method. The dimension of the main window is obtained by the
method getBounds() defined in the java.awt.Rectangle class
and assigned to bounds. The screen size is obtained by
calling the getWindow-Dimension() method which is defined in
the ParInt class; it is assigned to the member variable dim
which is of type Dimension.
/***********************************************************/
Rectangle bounds = frame.getBounds();
setSize(450,350);
dim = theApp.getWindowDimension();

/***********************************************************/
A calculation is done to put this dialog on a proper
position of the screen. If the screen is big enough to
show all the windows, the CompilegBox will appear on
the left of the main window.
/***********************************************************/
if (450 > bounds.x)
{
    setLocation(0, bounds.y + 110);
}
else
{
    setLocation(bounds.x - 450 , bounds.y + 110);
}

/***********************************************************/
A layout manager is created to handle the appearance of
the components in the dialog. The essential components
are: Checkbox funCheckbox, npCheckbox, and paraCheckbox.
"RUN" and "OK" buttons.
/***********************************************************/
GridBagLayout layout = new GridBagLayout();
GridBagConstraints constraints = new GridBagConstraints();
setLayout(layout);

/***********************************************************/
A textArea object derived from the TextArea class is added
to the top of the dialog and displays the message Click on
"Run" to run the chosen functions.
/***********************************************************/
constraints.weighty = 5;
constraints.gridwidth = 7;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 0;
constraints.insets = new Insets(0,10,0,10);
textArea = new TextArea("Message:Click on "Run\" to run the chosen function.\n",15,55);
textArea.setEditable(false);
textArea.setBackground(Color.white);
layout.setConstraints(textArea, constraints);
add(textArea);

/*****************************/

Below the testArea object are three checkboxes. The left checkbox is associated with the functionBox object. When the checkbox is checked the functionBox is displayed. Otherwise it is hidden. It is registered to have the item listener CheckboxHandler() class.

*******************************/

constraints.weighty = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 1;
state = frame.functionBox.getVisible();
funCheckbox = new Checkbox("Function Dialog", state);
funCheckbox.addItemListener(new CheckboxHandler());
layout.setConstraints(funCheckbox, constraints);
add(funCheckbox);

/*****************************/

The second checkbox is associated with the paraBox object. When it is checked then paraBox object is displayed. Otherwise it is hidden. It is registered to have the item listener CheckboxHandler() class.

*******************************/

constraints.weighty = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 1;
constraints.gridy = 1;
state = frame.paraBox.getVisible();
paraCheckbox = new Checkbox("Parameter Dialog", state);
paraCheckbox.addItemListener(new CheckboxHandler());
layout.setConstraints(paraCheckbox, constraints);
add(paraCheckbox);

/*****************************/

The right checkbox is for the npBox object. It is registered to have the item listener CheckboxHandler() class.

*******************************/

constraints.weighty = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 1;
state = frame.npBox.getVisible();
npCheckbox = new Checkbox("NP Dialog", state);
npCheckbox.addItemListener(new CheckboxHandler());
layout.setConstraints(npCheckbox, constraints);
add(npCheckbox);

/***************************************************/
A "Run" button is created to run the integral computation
when it is clicked. Its listener is the integrateBox object.
***************************************************/
constraints.weighty = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 0;
constraints.gridy = 2;
makeButton = new Button("Run");
makeButton.addActionListener(this);
layout.setConstraints(makeButton, constraints);
add(makeButton);

/***************************************************/
An "Clear Output" button is created to allow the user to
clear the text box. It has the integrateBox object as its
listener.
***************************************************/
constraints.weighty = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 1;
constraints.gridy = 2;
ClsButton = new Button("Clear Output");
ClsButton.addActionListener(this);
layout.setConstraints(ClsButton, constraints);
add(ClsButton);

/***************************************************/
An "EXIT" button is created to allow the user to exit the
dialog. The "EXIT" button also has the integrateBox object
as its listener.
***************************************************/
constraints.weighty = 5;
constraints.gridwidth = 1;
constraints.gridheight = 1;
constraints.gridx = 2;
constraints.gridy = 2;
okButton = new Button("Exit");
okButton.addActionListener(this);
layout.setConstraints(okButton, constraints);
add(okButton);

index = document.getSystemType();
}
The method actionPerfonned() is the action listener for the buttons "Run", "Clear Output", and "Exit".

```
public void actionPerfonned(ActionEvent e)
{
        **/)

        When the "run" button is clicked the "datain" file will be created to store the information needed by the MPI program. The first value is the function. The function is got by the getFunction() method of the functionBox object. It has to be called via the frame object, because the functionBox object is created in the frame object. The second value is the absolute tolerance obtained by the getA_tolerance() method of the paraBox object via the frame object. The third value is the relative tolerance obtained by the getR_tolerance() method of the paraBox object. The rule is also obtained from the paraBox object by the method getRule(). All these are enclosed in a try block. The catch block will catch the IOException.
```

```
if (e.getActionCommand().equals("Run"))
{
        fileName = "datain";
        try
        {
                BufferedWriter out = new BufferedWriter(
                        new FileWriter(fileName));

                str1 = frame.fonctionBox.getFunction();

                if (str1 == fun1) str1 = "1";
                else if (str1 == fun2) str1 = "2";
                else if (str1 == fun3) str1 = "3";
                else if (str1 == fun4) str1 = "4";
                else if (str1 == fun5) str1 = "5";
                else if (str1 == fun6) str1 = "6";
                else if (str1 == fun7) str1 = "7";
                else if (str1 == fun8) str1 = "8";
                else if (str1 == fun9) str1 = "9";
                else if (str1 == fun10) str1 = "10";
                else if (str1 == fun11) str1 = "11";
                else if (str1 == fun12) str1 = "12";
                else if (str1 == fun13) str1 = "13";
                else str1 = "14";

                out.write(str1 + "\n");
        }
```
FileReader in = new FileReader(fileName);
boolean EOF = false;
int c;

while (!EOF)
{
    try
    {
        c = in.read();
        if (c == -1)
        {
            EOF = true;
        }
        else
        {
            out.write(c);
        }
    } catch (EOFException e2) {
        EOF = true;
    } catch (IOException e2) {
        EOF = true;
        System.out.println("Error reading input file" + e2);
    }
}
else
{
    m_a_tolerance = frame.paraBox.getA_tolerance();
    out.write(m_a_tolerance + "\n");
    m_r_tolerance = frame.paraBox.getR_tolerance();
    out.write(m_r_tolerance + "\n");
    m_limit = frame.paraBox.getLimit();
    out.write(m_limit + "\n");
    m_rule = frame.paraBox.getRule();
    out.write(m_rule + "\n");
    m_dimension = frame.paraBox.getDimension();
    out.write(m_dimension + "\n");
    lowBound = frame.paraBox.getLowBound();
    int max = lowBound.getItemCount();
    for (int i=0; i < max; i++)
    {
        out.write(lowBound.getItem(i) + "\n");
    }
    upBound = frame.paraBox.getUpBound();
    max = upBound.getItemCount();
    for (int i=0; i < max; i++)
try {
    np_value = frame.npBox.getValue();
} catch (NullPointerException npe) {
    np_value = 1;
}

The Runtime object r is used to create a process and invoke the "Makefile". We need to specify the command to be executed. A String object cmdarray[] is created to store the command we wish to execute. To run the process we need to put it in a try block so we can catch the exceptions. A try block is simply the keyword try, followed by braces enclosing the code that can throw the exception. A catch block containing the code to handle the exception follows the try block, which contains the code that may throw that particular exception.

Runtime r = Runtime.getRuntime();
pq = document.getPQType();
if (pq == PQ1) // Local - No load balance
{
    cmdarray = "make test-nolb" + Integer.toString(np_value);
}
else if (pq == PQ2) // Local - With load balance
{
    cmdarray = "make test-lb" + Integer.toString(np_value);
}
else
{
    cmdarray = "make \"NPROC=\" + Integer.toString(np_value) + \"\"
    test-lb" + Integer.toString(np_value);
}
The method exec() of the r object creates the process p which runs the "Makefile". Both the "Run" and "Exit" buttons are disabled and wait until the process terminates. When the process has terminated, it is destroyed and the "Run" and "Exit" buttons are back enabled. Two exceptions must be caught. One is IOException that signals that an I/O exception of some sort has occurred. Another is the Interrupted-Exception, which occurs when a thread is interrupted by an interrupt method in class Thread.

try{
    Process p = r.exec(cmdarray);
    makeButton.setLabel("Wait...");
    makeButton.setEnabled(false);
    okButton.setEnabled(false);

    p.waitFor();
    p.destroy();
    makeButton.setLabel("Run");
    makeButton.setEnabled(true);
    okButton.setEnabled(true);
}
catch (IOException exception)
{
    System.err.println("Command failed with IO Exception");
}
catch (InterruptedException iexception)
{
    System.err.println("Command failed with Interrupted Exception");
}

When the program is executed all the messages are written to the file "report". In a try block, the file is read into the textArea object. A catch block catches the FileNotFoundException. To read the file, a FileReader object "in" is created to open the file. A while loop keeps reading the file until an EOF flag is met. The read() method of the "in" object is used to read the file. We also need a try block for the read and two catch blocks to catch the EOFException and IOException.

try{
    FileReader in = new FileReader("report");
    boolean EOF = false;
    int c;
    while (!EOF)
    {
        try
        {
```java
{    c = in.read();
    if (c == -1) {
        EOF = true;
        textArea.append(str);
    } else {
        str += (char) c;
    }
}
catch(EOFException e2) {
    EOF = true;
}
catch(IOException e2) {
    EOF = true;
    System.out.println("Error reading input file" + e2);
}
}
catch(FileNotFoundException e3) {
    System.out.println("Error reading input file" + e3);
}
*******************************************************************************
When the button "Clear Output" is clicked. All the contents in the textArea is wiped.
*******************************************************************************
    else if (e.getActionCommand().equals("Clear Output")) {
        textArea.replaceRange("", 0, textArea.getRows() * textArea.getColumns());
    }
*******************************************************************************
When the button "Exit" is clicked. This object is destroyed.
*******************************************************************************
    else  dispose();
*******************************************************************************
The item listener has the itemStateChanged() method implemented to run some specific tasks after any checkbox' status is changed. When the listener is invoked, it will execute the setVisible() methods of the npBox, functionBox, and paraBox objects. The getState() method is the key to determine whether the object is visible or not visible. If the state returned from getState() is true then it is visible. Otherwise it is not visible.
class CheckboxHandler implements ItemListener
{
    public CheckboxHandler()
    {
    }
    public void itemStateChanged(ItemEvent e)
    {
        frame.npBox.setVisible(npCheckbox.getState());
        frame.functionBox.setVisible(funCheckbox.getState());
        frame.paraBox.setVisible(paraCheckbox.getState());
    }
}
The MessageBox class defines a simple dialog to display a short message. It is used as an about dialog in this application. This class extends the java.awt.Dialog class to be a visible dialog and implements the ActionListener interface to receive action events for the "OK" button. This dialog contains the message of "Parint Project Java version 1.0" and an "OK" button.

```
import java.awt.*;
import java.awt.event.*;

class MessageBox extends Dialog implements ActionListener
{
    Frame frame;

    // Constructor
    public MessageBox(Frame frame, String title, String messageString)
    {
        super(frame, title, false); // Create modal dialog
        this.frame = frame;

        // Position the dialog in the parent window
        Rectangle bounds = frame.getBounds();
        setLocation(bounds.x + bounds.width/3, bounds.y + bounds.height/3);

        // BorderLayout is the layout manager to manage the appearance of the dialog. A panel on the top contains the label that displays the message. A panel on the bottom contains the "OK" button. The button adds the action listener by the addActionListener() method.
        Panel messagePane = new Panel(); // Create the message panel
        Label message = new Label(messageString); // Label to hold message
        messagePane.add(message); // Add the label
        add(messagePane, BorderLayout.CENTER); // Add the pane centrally

        Panel buttonPane = new Panel(); // Create the panel for buttons
        Button button = new Button(" OK "); // The OK button
        buttonPane.add(button); // Add the button to the pane
        button.addActionListener(this); // Add the button listener
        add(buttonPane, BorderLayout.SOUTH); // Add the button at the bottom
        pack(); // Pack the dialog window
    }
}
```
The method `actionPerformed()` is the action listener for the "OK" button. When the "OK" button is clicked, the method `dispose()` is called to free the resource and exit the dialog.

```java
public void actionPerformed(ActionEvent e)
{
    setVisible(false); // Hide the dialog
}
```
class LanguageCommand implements ItemListener
{
    int itemID;       // ID for the menu item handled
    ParIntDoc document;  // The document object

    // Constructor for item listener
    public LanguageCommand(int itemID, ParIntDoc document)
    {
        this.itemID = itemID;  // Save the menu item ID
        this.document = document;  // Store the document
    }

    // Handle element menu item state changed
    public void itemStateChanged(ItemEvent e)
    {
        document.setLanguageType(itemID);  // Store new element type
    }
}
/*******************************************************************************************/
* File Name: PQCommand.java
* Description:
* Handles method menu command. This item listener is invoked when PQ
* group menu checkbox is checked.
*******************************************************************************************/
import java.awt.event.*;

class PQCommand implements ItemListener
{
    int itemID; // ID for the menu item handled
    ParIntDoc document; // The document object

    // Constructor for item listener
    public PQCommand(int itemID, ParIntDoc document)
    {
        this.itemID = itemID; // Save the menu item ID
        this.document = document; // Store the document
    }

    // Handle element menu item state changed
    public void itemStateChanged(ItemEvent e)
    {
        document.setPQType(itemID); // Store new element type
    }
}
class RuleCommand implements ItemListener {
    int itemID;       // ID for the menu item handled
    ParIntDoc document;   // The document object

    // Constructor for item listener
    public RuleCommand(int itemID, ParIntDoc document) {
        this.itemID = itemID;       // Save the menu item ID
        this.document = document;   // Store the document
    }

    // Handle element menu item state changed
    public void itemStateChanged(ItemEvent e) {
        document.setRuleType(itemID); // Store new element type
    }
}
import java.awt.event.*;

class SystemCommand implements ItemListener
{
    int itemID; // ID for the menu item handled
    ParlntDoc document; // The document object

    // Constructor for item listener
    public SystemCommand(int itemID, ParlntDoc document)
    {
        this.itemID = itemID; // Save the menu item ID
        this.document = document; // Store the document
    }

    // Handle element menu item state changed
    public void itemStateChanged(ItemEvent e)
    {
        document.setSystemType(itemID); // Store new element type
    }
}
BIBLIOGRAPHY

"JDKTM 1.2 Beta 3 Documentation" from Sun Microsystems, Inc. Available at: 
<http://java.sun.com:80/products/jdk/1.2/docs/index.html>

"The Java Tutorial" by Mary Campione and Kathy Walrath. Available at: 
<http://java.sun.com:80/docs/books/tutorial/index.html>