4 Упрощение работы в wxPython при помощи PyCrust

Эта глава включает

- Взаимодействие с wxPython-программой
- Рассмотрение возможностей PyCrust
- Сопряжение PyCrust с wxPython-приложением
- Работа с GUI и вспомогательными модулями PyCrust
- Взаимодействие wxPython-программы с модулями PyCrust

PyCrust – это написанная на wxPython графическая оболочка, которую Вы можете использовать для анализа wxPython-программ.

Почему она названа PyCrust? Когда Патрик О'браен (Patrick O'Brien), используя wxPython, создавал диалоговую Python-оболочку, очевидно название «PyShell» уже было использовано. Вместо него выбор пал на «PyCrust».

PyCrust – это часть большого пакета Py, который включает дополнительные программы со связанными функциональными возможностями, включая PyFilling, PyAlaMode, PyAlaCarte и PyShell.

Общая идея этих программ - комбинация возможностей визуальной графической (point-and-click) среды, интерактивности wxPython и самоуправления (интроспективность) при выполнении. В отличие от любой из Py-программ, которые усиливают эту комбинацию, PyCrust представляет более полную реализацию этой идеи.

В этой главе, мы покажем Вам, что делают PyCrust и связанные с ним программы, и как Вы можете их использовать, чтобы сделать Вашу работу с потоком wxPython более равномерной. Сначала мы поговорим об обычной Python-оболочке, затем собственно о PyCrust, и, наконец, мы охватим остальные программы в пакете Py.

4.1 Как взаимодействовать с wxPython-программой?

Неизменная особенность Python в сравнении с другими языками программирования, состоит в том, что его можно использоваться двумя способами: Вы можете запускать на выполнение существующие Python-программы, или же запустить Python в интерактивном режиме командной строки. Выполнение Python в интерактивном режиме подобно диалогу с Python-интерпретатором. Вы вводите строку кода и нажимаете Enter. Python выполняет этот код, выдает ответ и запрашивает у Вас следующую строку. Такой интерактивный режим существенно отличает Python от таких языков, как C++, Visual Basic или Perl. Имея такой интерпретатор, нет необходимости, чтобы сделать простые вещи, писать в wxPython цельную программу. Фактически, Вы можете даже использовать диалоговый Python как настольный калькулятор.

В листинге 4.1 мы запустили Python в режиме командной строки и выполнили небольшие математические вычисления. Python стартует, отображая несколько строк информации, сопровождаемых его основным приглашением (>>>). Когда Вы введёте, что-нибудь, что потребует дополнительных строк кода, Python покажет вторичное приглашение (...).

Листинг 4.1 Пример интерактивного сеанса Python

```
$ Python
Python 2.3.3 (#1, Jan 25 2004, 11:06:18)
[GCC 3.2.2 (Mandrake Linux 9.1 3.2.2-3mdk)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 2
4
```

```
>>> 7 * 6
42
>>> 5 ** 3
125
>>> for n in range(5):
... print n * 9
...
0
9
18
27
36
>>>
```

Диалоговый Python – это не только хороший настольный калькулятор, но и великолепный инструмент для обучения, так как обеспечивает немедленную ответную реакцию. Если Вы в чем-то сомневаетесь, Вы можете просто запустить Python, ввести несколько строк исследуемого кода, посмотреть реакцию Python и привести в соответствие основной код. Один из лучших способов изучения работы существующего кода Python – испытание его в интерактивном режиме.

PyCrust устанавливает стандарт оболочки Python

Когда Вы работаете с Python в интерактивном режиме, Вы работаете в среде, называемой Python-*оболочкой*, которая подобна другим средам, типа окна DOS на платформах Microsoft или командной строке bash в Unix-системах.

Основа всех Python-оболочек, которую Вы видите, когда запускаете Python из командной строки, такая же, как и в листинге 4.1. Хотя Python и полезная оболочка, она полностью ориентирована на текстовый, а не графический режим, и не обеспечивает всех клавиатурных комбинаций (shortcuts) или всплывающих подсказок (hints), которые способен предоставить Python. Чтобы предоставить эти дополнительные функциональные возможности, были специально разработаны несколько графических Python-оболочек. Наиболее общеизвестный является интегрированная среда разработки IDLE (Integrated DeveLopment Environment), которая является стандартной частью дистрибутива Python. Оболочка IDLE, как показано на рисунке 4.1, выглядит подобно оболочке командной строки Python, но имеет дополнительные графические возможности, подобные подсказкам о вызове (calltips).

Другие инструменты для разработки на Python, типа PythonWin или Boa Constructor, имеют графическую оболочку Python, подобную той, что и в IDLE. Существование всех этих оболочек, подтолкнуло к созданию PyCrust. Хотя каждая из инструментальных оболочек имеет некоторые полезные возможности, например, повторный вызов команды, автозавершение и подсказки о вызове, нет инструмента, который бы имел полный набор всех этих возможностей. Одна из целей PyCrust состоит в том, чтобы обеспечить полный набор возможностей всех существующих оболочек Python.

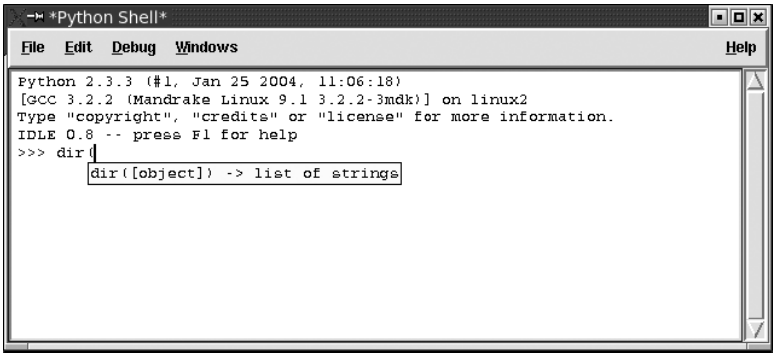

Рисунок 4.1 Оболочка IDLE предоставляет для функций и методов подсказки о вызове

Другой причиной, по которой был создан PyCrust, было то, что инструментальные средства, написанные при помощи одной GUI-библиотеки, часто не работают с кодом от другой GUI-библиотеки. Например, IDLE написан с помощью Tkinter, а не wxPython. До недавнего времени, если бы Вы попытались импортировать и использовать модуль wxPython из Python-оболочки IDLE, Вы бы получили конфликт между циклом событий wxPython и циклом событий Tkinter, что привело бы к останову или краху программы.

В действительности, эти два комплекта инструментов соперничают за право управления циклом событий. Поэтому, если Вам необходимы возможности самоуправления при выполнении, реализованные в модулях wxPython, Ваша Python-оболочка должна быть написана на wxPython. Поскольку не существует Python-оболочки, поддерживающей все множество возможностей, для того чтобы удовлетворить такую потребность, был создан PyCrust.

4.2 Какие у PyCrust полезные возможности?

Теперь мы рассмотрим некоторые возможности, которые предоставляет оболочка PyCrust. PyCrust выглядит знакомым, т. к. отображает те же самые строки информации и использует те же сообщения, что и командная строка оболочки Python. Рисунок 4.2 показывает открытый экран PyCrust.

Вы заметите, что фрейм PyCrust, содержащий элемент `wx.SplitterWindow`, разделен на две секции: верхняя секция напоминает обычную оболочку Python, нижняя секция содержит элемент управления Notebook, который включает разнообразные ярлыки, показывающие информацию о текущем пространстве имен (namespace). Верхняя секция (PyCrust-оболочка), имеет несколько полезных возможностей, обсуждаемых в следующих нескольких подразделах.

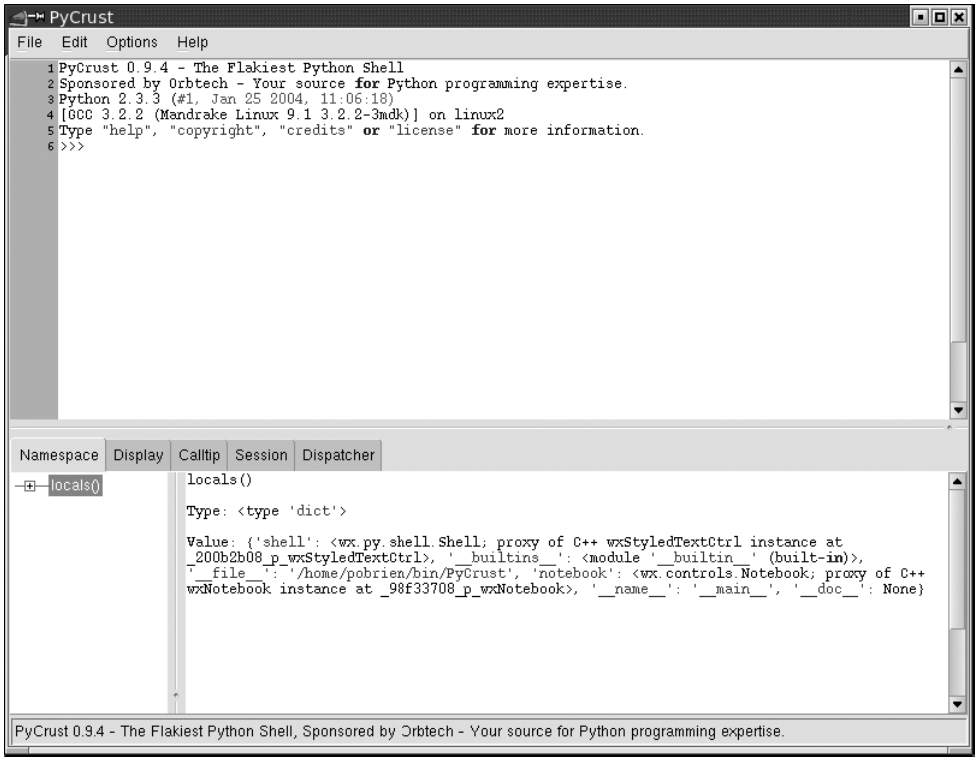

Рисунок 4.2 Запущенный PyCrust показывает оболочку и интерфейс на основе ярлыков записной книжки

4.2.1 *Автозавершение*

Когда Вы вводите имя объекта, завершаемое оператором-точкой, происходит автозавершение. PyCrust показывает алфавитный список всех известных атрибутов данного объекта. Если Вы введете дополнительные буквы, подсвеченный выбор в списке изменится, и будет соответствовать введенным Вами буквам. Если нужный Вам элемент выделен, нажмите клавишу Tab и PyCrust заполнит за Вас остальную часть имени атрибута.

На рисунке 4.3 PyCrust показывает список атрибутов для объекта строки. Список автозавершения включает все свойства и методы этого объекта.

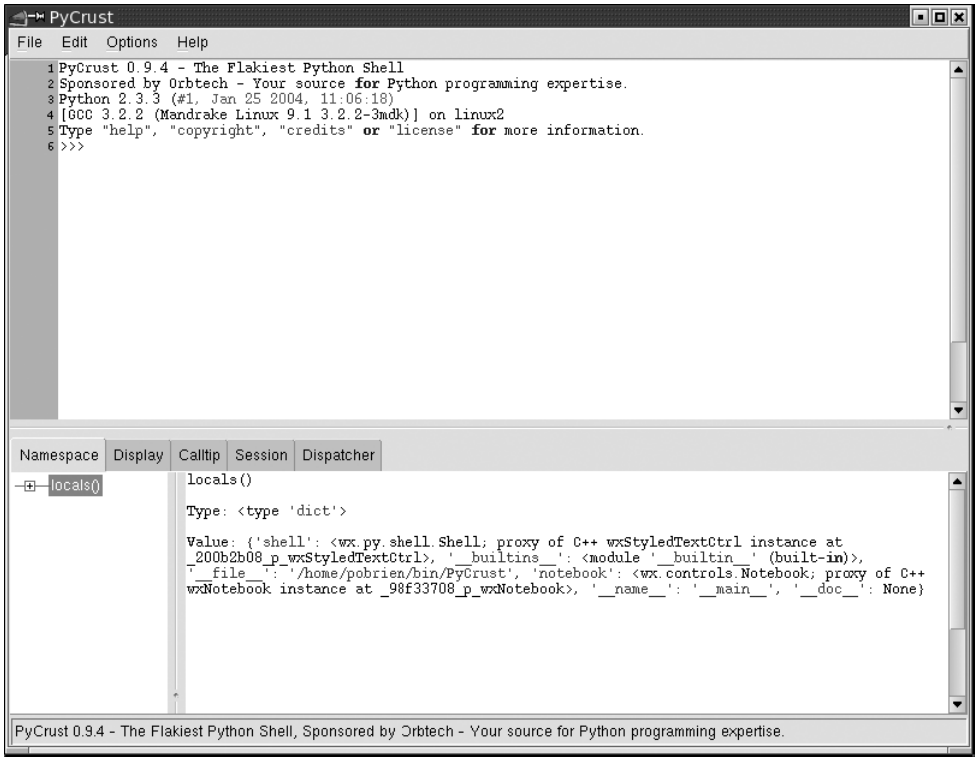

Рисунок 4.3 Отображение атрибутов объекта при автозавершении

4.2.2 *Подсказки о вызове (Calltip) и параметры по умолчанию*

Когда Вы вводите левую круглую скобку после имени вызываемого объекта, PyCrust показывает окно подсказки о вызове (calltip) (см. рисунок 4.4) с информацией о передаваемых при вызове аргументах и справочной информацией по этому вызову (если она имеется).

Вызываемый объект может быть функцией, методом, встроенной командой или классом. Все они могут принимать аргументы и могут иметь справочную информацию, предоставляющую сведения о том, что элемент делает и какой тип возвращаемого значения. Эта информация выводится во временном окне, выше или ниже знака каретки, что устраняет потребность обращаться к документации. Если Вы знаете, как использовать объект вызова, игнорируйте подсказку о вызове и продолжайте ввод.

Когда Вы вводите левую круглую скобку, PyCrust вносит стандартные параметры (параметры по умолчанию). Когда это происходит, PyCrust автоматически выбирает дополнительно созданный текст и результирующий ввод замещается. Для сохранения этих параметров нажмите любую клавишу движения курсора (клавишу стрелки) и текст станет Вам доступен для модификации.

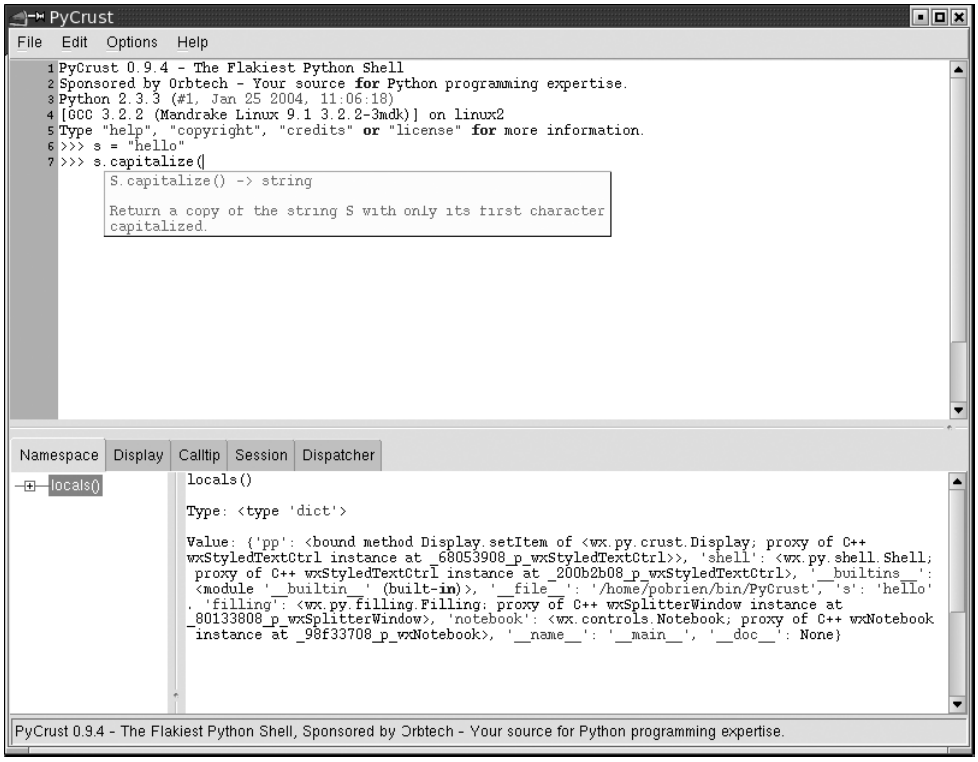

Рисунок 4.4 Подсказка о вызове выводит информацию о вызываемом объекте

4.2.3 *Выделение синтаксиса*

При вводе кода в оболочку PyCrust изменяет цвет текста в зависимости от его значения. Например, ключевые слова Python появляются в одном цвете, значения текстовых строк в другом, а комментарии в третьем. Это обеспечивает визуальное подтверждение того, что Вы не пропустили каких-либо закрывающих кавычек или не написали с орфографической ошибкой ключевое слово Python.

Многие возможности PyCrust делают возможным поставляемое с wxPython мощное управление текстом. Компонент `wx.stc.StyledTextCtrl` - wxPython обертка на исходном коде компонента редактирования Scintilla, разработанная Нейлом Ходсоном (Neil Hodgson). Scintilla (www.scintilla.org) используется различными приложениями редактирования исходного кода, включая поставляемые с wxPython демонстрационные программы. Хотя и было бы не легко создать функциональность редактора исходного кода подобную оболочке wxPython, но создать PyCrust без Scintilla было бы практически невозможно.

4.2.4 Помощь по языку Python

PyCrust обеспечивает полную поддержку функций справочной системы по языку Python. Как показано на рисунке 4.5, функция помощи выводит информацию практически обо всех аспектах Python.

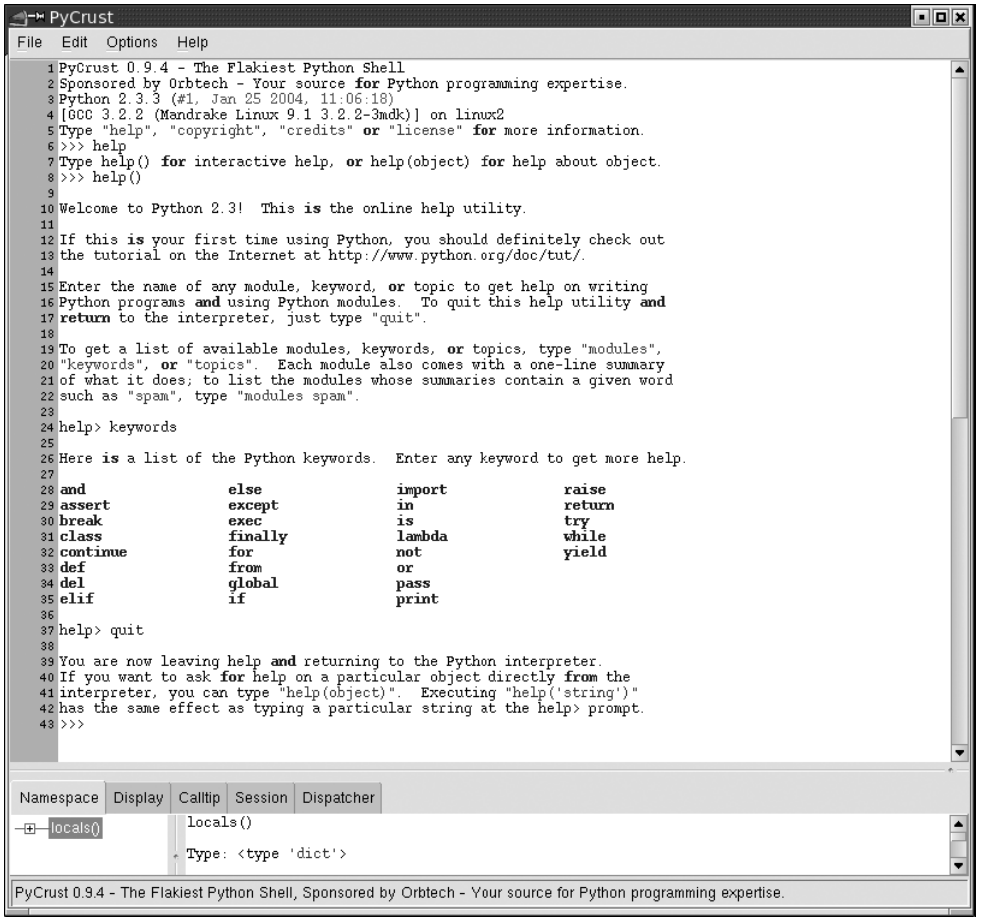

Рисунок 4.5 Использование внутри PyCrust функции помощи по Python

Функцию помощи по языку Python обеспечивает дополнительное приглашение (`help`). После использования помощи, введя `quit` в ответ на приглашение `help`, Вы можете покинуть этот режим и вернуться к обычному Python.

4.2.5 Повторный вызов команды

Есть много способов избежать ввода внутри PyCrust. Большинство из них включают в себя запоминание всего того, что Вы предварительно ввели, внесение необходимых изменений и отсылку результата интерпретатору Python. Например, PyCrust поддерживает историю всех команд, которые Вы ввели в текущем сеансе

работы. Вы можете повторно вызвать из истории команд любую предварительно введенную команду Python (однострочную или многострочную). Таблица 4.1 показывает список клавиатурных комбинаций, которые касаются этих функциональных возможностей.

Таблица 4.1 Клавиатурные комбинации повторного вызова команд оболочки PyCrust

Комбинация клавиш	Результат
Ctrl+↑	Извлечение предыдущего элемента истории
Alt+P	Извлечение предыдущего элемента истории
Ctrl+↓	Извлечение следующего элемента истории
Alt+N	Извлечение следующего элемента истории
Shift+↑	Вставка предыдущего элемента истории
Shift+↓	Вставка следующего элемента истории
F8	Завершение команды на основе элемента истории (Введите несколько символов предыдущей команды и нажмите F8)
Ctrl+Enter	Вставить новую строку в многострочную команду

Как Вы видите, отдельные команды для восстановления и вставки старых команд различаются тем, как PyCrust обращается с текстом, введенным в текущем запросе wxPython. Для замены введенного Вами текста используйте одну из комбинаций, которая извлекает элемент истории. Используйте одну из комбинаций для вставки элемента истории, чтобы вставить старую команду в позицию каретки.

Вставка строки в середину многострочной команды работает по-другому, чем вставка в команду однострочной команды. Нажимая клавишу `Enter`, Вы не можете вставить строку в многострочную команду, потому что она посылает текущую команду интерпретатору Python. Вместо этого, чтобы вставить разрыв на текущей строке, нажмите `Ctrl+Enter`. Если Вы находитесь в конце строки, за текущей строкой вставляется пустая строка. Этот процесс похож на вырезание и вставку текста в обычном текстовом редакторе.

Заключительный метод повторного вызова команды состоит в том, чтобы просто переместить каретку на команду, которую Вы хотите вызвать и нажать `Enter`. PyCrust скопирует эту команду в текущую позицию ввода Python, и поместит каретку в конец. Вы сможете затем изменить команду или снова нажать `Enter` и послать команду интерпретатору.

Эти клавиатурные комбинации позволяют Вам быстрее разрабатывать код, проверяя на каждом шаге созданный код. Например, Вы можете определить новый класс Python, создать экземпляр этого класса и посмотреть, как он себя ведет. Затем Вы можете вернуться к определению класса, добавить дополнительные методы или редактировать существующие методы и создать новый экземпляр. Повторяя это столько, сколько Вам нужно, Вы сможете развить определение класса до такой степени, что оно будет достаточно хорошим, чтобы

его вырезать и вставлять в исходном коде программы. И это приведет нас к нашей следующей возможности.

4.2.6 Вырезание (Cut) и вставка (Paste)

Иногда Вам может потребоваться использовать разработанный в оболочке код без необходимости его повторного ввода. В других случаях, Вы можете найти образец кода, возможно в онлайновом руководстве и использовать его в Python-оболочке. PyCrust предоставляет несколько простых операций вырезания и вставки, перечисленных в таблице 4.2.

Таблица 4.2 Клавиатурные комбинации вырезания и вставки оболочки PyCrust

Комбинация клавиш	Результат
Ctrl+C	Копировать выделенный текст, удаляя приглашения (>>>)
Ctrl+Shift+C	Копировать выделенный текст, оставляя приглашения (>>>)
Ctrl+X	Вырезать выделенный текст
Ctrl+V	Вставить из буфера обмена
Ctrl+Shift+V	Вставить и запустить многострочную команду из буфера обмена

Другая особенность операции вставки состоит в том, что PyCrust распознает и автоматически удаляет из любого кода, вставляемого в PyCrust-оболочку, стандартные приглашения Python (>>>). Это облегчает копирование образцов кода из руководства или почтового сообщения, вставку и использование его в PyCrust без необходимости делать ручную чистку.

Иногда, когда Вы копируете код, Вам может понадобиться удалить приглашения PyCrust, как например, при копировании кода в ваши исходные файлы. В другой раз, Вы захотите сохранить приглашения, например, когда Вы копируете примеры в какой-либо документ или отправляете в группу новостей. PyCrust обеспечивает оба варианта при копировании текста из оболочки.

4.2.7 Стандартное окружение оболочки

В максимально возможной степени в пределах среды wxPython, PyCrust ведет себя так же, как и Python-оболочка командной строки. Имеются в виду некоторые необычные ситуации, как например, травление (pickling) экземпляров классов, которые определены в рамках сеанса оболочки. Единственная область, где PyCrust даёт сбой, кроется в его способности замещать функциональные возможности командной строки, являющиеся клавиатурными прерываниями. Как только код Python введен в PyCrust-оболочку, нет возможности прервать его выполнение. Например, предположим, Вы закодировали в PyCrust бесконечный цикл, как в следующем примере:

```
>>> while True:
...    print "Hello"
...
```

После того, как Вы нажмете Enter, и код отправится интерпретатору Python, PyCrust приостановит ответную реакцию. Чтобы прерывать бесконечный цикл, завершите программу PyCrust. Этот недостаток PyCrust - отличие от Python-оболочки командной строки, которая сохраняет способность обрабатывать клавиатурное прерывание (Ctrl+C). В Python-оболочке командной строки Вы увидели бы следующее поведение:

```
>>> while True:
...    print "Hello"
...
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
KeyboardInterrupt
>>>
```

Механизм обработки событий в GUI среде чрезвычайно затрудняет решение, которое позволило бы PyCrust преодолеть бесконечный цикл или прервать любую длительно исполняемую последовательность введенного в приглашение оболочки кода. Будущая версия PyCrust может обеспечить решение этого вопроса. Между тем, помните об этом поведении. К счастью, это единственное известное различие между PyCrust и стандартной командной оболочкой. Во всех других отношениях, оболочка PyCrust работает точно так же, как и оболочка командной строки Python.

4.2.8 *Динамическое обновление*

После запуска оболочки PyCrust все ее возможности динамически обновляются, и такие свойства как автозавершение и подсказка о вызове будут доступными даже для объектов, определяемых при непосредственном вводе команд и данных в строке приглашения оболочки. Например, посмотрите показанные на рисунках 4.6 и 4.7 сеансы, где мы определяем и используем класс.

На рисунке 4.6 PyCrust показывает доступные для нового класса варианты автозавершения.

На рисунке 4.7 PyCrust показывает подсказку о вызове для недавно определенного метода класса.

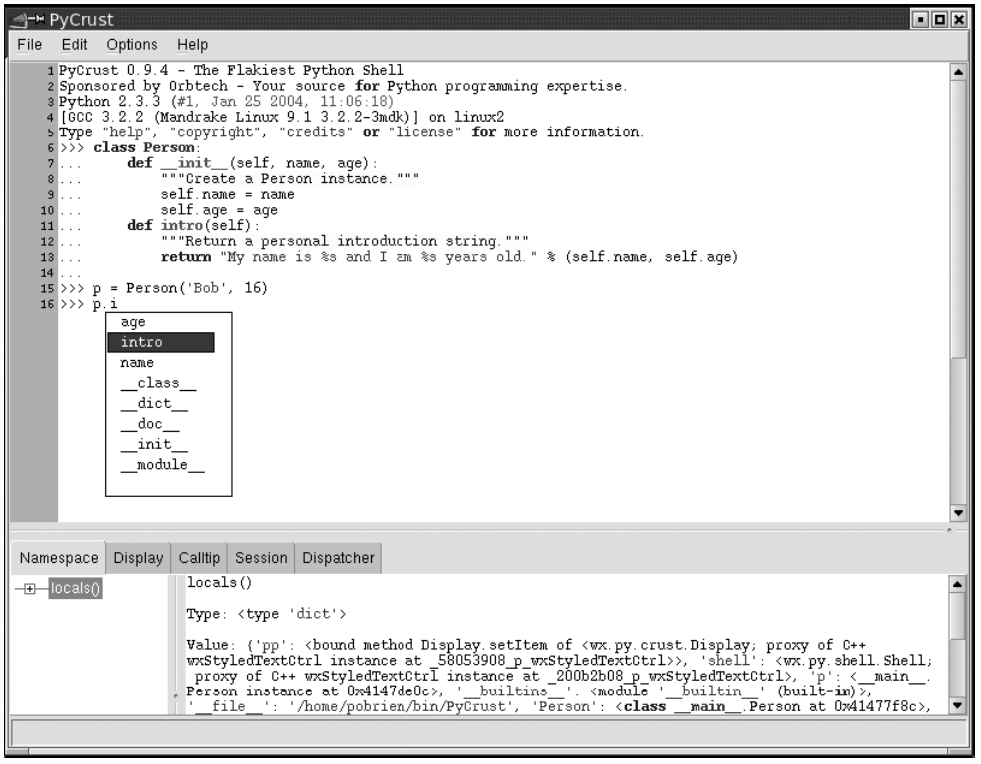

Рисунок 4.6 Динамически генерированная PyCrust информация автозавершения

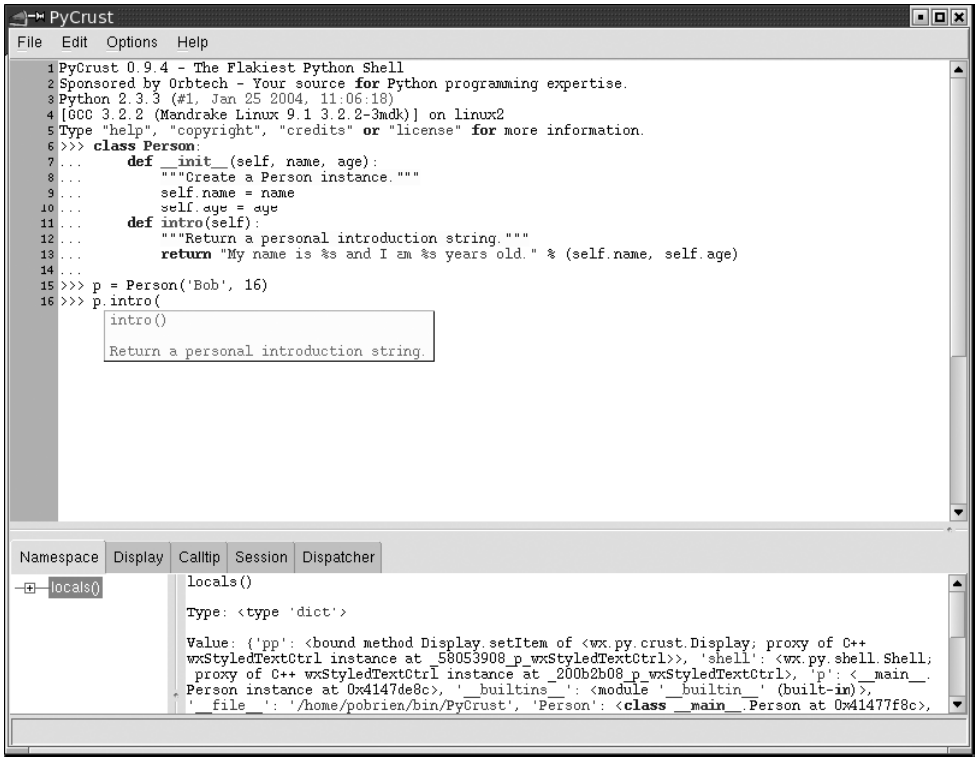

Рисунок 4.7 Динамически генерированная PyCrust информация подсказки о вызове

Это иллюстрирует реализуемый PyCrust способ динамического выполнения Python, который невозможен в других языках программирования со статическим вводом и компиляцией.

4.3 Каково назначение страниц PyCrust?

В нижней половине интерфейса PyCrust расположен элемент наподобие блокнота (notebook), который включает несколько страниц полезной информации. Появляющаяся при старте страница PyCrust – это Namespace (пространство имен).

4.3.1 Страница Namespace (Пространство имен)

Показанная на рисунке 4.8 страница Namespace, разделена на две части, где снова используется элемент `wx.SplitterWindow`. Левая сторона содержит элемент управления деревом, который отображает текущее пространство имен, а правая сторона отображает информацию о текущем объекте, выбранном в дереве пространства имен.

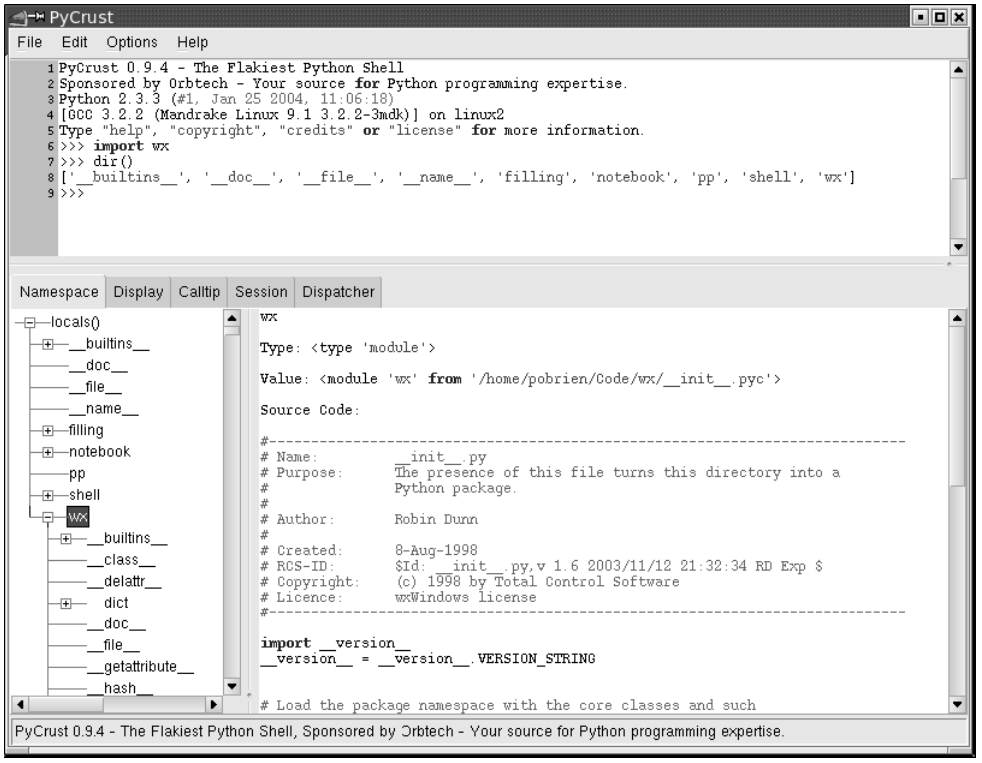

Рисунок 4.8 Дерево пространства имен PyCrust позволяет погружаться в объекты и инспектировать их атрибуты

Дерево Namespace представляет иерархический вид всех объектов в текущем пространстве имен. В нем содержатся пункты, которые возвращаются при работе встроенной функции Python `locals()`. На рисунке 4.8, мы импортировали пакет `wx` и выбрали его в дереве пространства имен. Правая сторона отображает имя выбранного элемента, его тип и текущее значение. Если объект имеет ассоциированный с ним исходный код, PyCrust также его отображает. В этом случае, `wx` является пакетом wxPython, поэтому PyCrust отображает исходный код из файла `__init__.py`, который находится в директории `wx`.

Первая строка экрана предоставляет имя объекта, полностью пригодное, для операций вырезания и вставки в оболочку PyCrust или в исходный код Вашего приложения. Например, если Вы импортируете и откроете модуль `locale`, Вы можете достичь элементов этого модуля, сохраненных внутри атрибута словаря `encoding_alias`. Как только Вы выберете один из этих элементов, Вы можете вырезать и вставлять его отображаемое имя непосредственно в оболочку PyCrust, например следующим образом:

```
>>> import locale
>>> locale.encoding_alias['en']
'ISO8859-1'
>>>
```

В этом случае, PyCrust снабдил нас полностью пригодным именем (`locale.encoding_alias['en']`), использующим индексную нотацию Python (`['en']`) для ссылки на определенный пункт словаря `encoding_alias`. Этот механизм также работает и для списков. Если Вы что-то указываете в дереве пространства имен для ссылки в своем коде, PyCrust даёт Вам точный синтаксис для выполнения этой задачи.

4.3.2 Страница Display (Экран)

Страница Display отображает красивый вид объекта. PyCrust имеет встроенную функцию `pp()`, которая использует модуль качественной печати Python (`pprint`) для выполнения хорошо отформатированного вида любого объекта wxPython. Информация на странице Display корректируется при каждом обновлении отображаемого объекта, не требуя от Вас явного импортирования и многократного использования `pprint`.

Например, чтобы увидеть, как изменяется содержимое списка при манипулировании им в оболочке, Вы можете выделить страницу Display, использовать функцию `pp()` в пределах оболочки для отображения Вашего объекта списка на странице Display, а затем запустить код, который модифицирует этот список. Всякий раз, когда список будет изменяться, изменения сразу же будут появляться на странице Display.

4.3.3 Страница Calltip (Подсказка о вызове)

Страница Calltip отображает содержание последней подсказки о вызове Python-оболочки. Выберите страницу Calltip, если Вы работаете с вызываемыми объектами, которые требуют большого количества передаваемых им параметров. При использовании пакета wxPython существует много классов, которые могут иметь много методов, принимающих множество параметров. Например, для создания `wx.Button`, Вы можете передать вплоть до восьми параметров, один из которых обязателен, тогда как другие семь имеют значение по умолчанию. Страница Calltip отображает следующую информацию о конструкторе `wx.Button`:

```
__init__(self, Window parent, int id=-1, String label=EmptyString,
    Point pos=DefaultPosition, Size size=DefaultSize,
    long style=0, Validator validator=DefaultValidator,
    String name=ButtonNameStr) -> Button

Create and show a button. The preferred way to create standard buttons
is to use a standard ID and an empty label. In this case wxWigets will
automatically use a stock label that corresponds to the ID given. In
addition, the button will be decorated with stock icons under GTK+2.
```

Поскольку классы wxPython в действительности надстроены над классами C++, информация для подсказок о вызове основывается только на документации класса (docstrings). Она сгенерирована, чтобы показать требуемые базовому классу C++ состав передаваемых параметров и их тип (int, string, point и т.д.). Вот почему подсказка о вызове конструктора `wx.Button` показывается таким способом.

PyCrust инспектирует объекты, полностью определенные на языке Python, полагаясь на характер их аргументов.

4.3.4 Страница Session (Сеанс)

Страница Session (в новых версиях – History (История)) является простым текстовым элементом, который перечисляет все команды, введенные в текущем сеансе оболочки. Это облегчает вырезание и вставку команд для использования где-нибудь еще, при этом нет необходимости удалять отклик, который возвращается интерпретатором wxPython.

4.3.5 Страница Dispatcher (Диспетчер)

PyCrust включает модуль, названный `dispatcher`, предоставляющий механизм для свободного связывания объектов внутри приложения. PyCrust использует диспетчер, чтобы держать аспекты своего интерфейса актуальными, когда команды посылаются из оболочки в интерпретатор Python. Страница Dispatcher (рисунок 4.9) содержит информацию о сигналах, проходящих через механизм диспетчеризации. Прежде всего это полезно при работе непосредственно с PyCrust.

Страница Dispatcher также иллюстрирует, как добавить другую страницу в элемент `wx.Notebook`. Приведенный ниже исходный код отображаемого на странице Dispatcher элемента управления текстом, показывает, как может быть использован модуль диспетчера:

```
class DispatcherListing(wx.TextCtrl):
    """Text control containing all dispatches for session."""

    def __init__(self, parent=None, id=-1):
       style = (wx.TE_MULTILINE | wx.TE_READONLY |
                wx.TE_RICH2 | wx.TE_DONTWRAP)
       wx.TextCtrl.__init__(self, parent, id, style=style)
       dispatcher.connect(receiver=self.spy)
    def spy(self, signal, sender):
       """Receiver for Any signal from Any sender."""
       text = '%r from %s' % (signal, sender)
       self.SetInsertionPointEnd()
       start, end = self.GetSelection()
       if start != end:
           self.SetSelection(0, 0)
       self.AppendText(text + '\n')
```

Теперь, когда Вы увидели, что может делать PyCrust как автономная оболочка Python и инспектор пространства имен, давайте взглянем на некоторые приемы использования PyCrust в Ваших собственных программах wxPython.

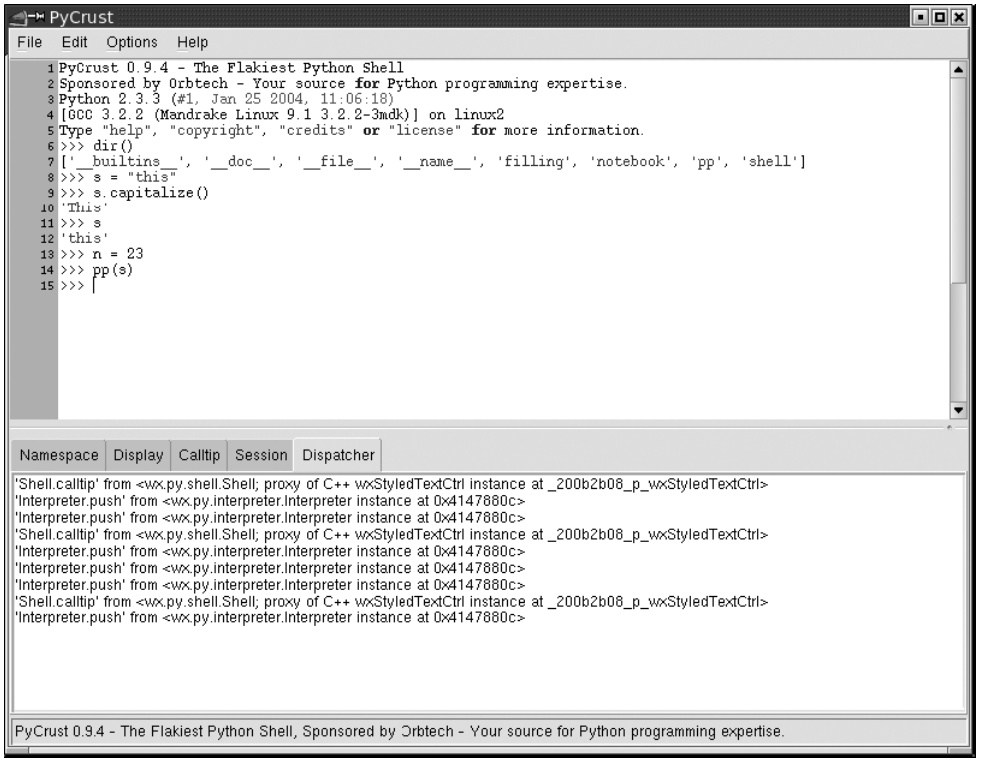

Рисунок 4.9 Диспетчер PyCrust информирует о том, каким образом команды передаются интерпретатору Python

4.4 *Как связать PyCrust с моим wxPython-приложением?*

Допустим, Вы использовали wxPython для создания программы, Ваша программа работает, и теперь Вы хотели бы лучше понять, как именно она работает. В этой главе Вы увидели возможности PyCrust и они могут оказаться очень полезными для понимания функционирования Вашей программы. Но Вам не хотелось бы изменять программу только для того, чтобы использовать ее с PyCrust. Что же делать в этом случае?

Передавая имя Вашей программы утилите PyWrap, Вы сможете без изменений ее запустить под оболочкой PyCrust. Листинг 4.2 показывает программу `spare.py`, которую мы подготовили для связи с PyCrust.

Листинг 4.2 Программа spare.py, подготовленная для связи с PyCrust

```
#!/usr/bin/env python

"""Spare.py is a starting point for simple wxPython programs."""

import wx
```

```
class Frame(wx.Frame):
    pass

class App(wx.App):

    def OnInit(self):
        self.frame = Frame(parent=None, id=-1, title='Spare')
        self.frame.Show()
        self.SetTopWindow(self.frame)
        return True

if __name__ == '__main__':
    app = App()
    app.MainLoop()
```

Для запуска этой программы в окружении PyCrust, передайте имя программы в PyWrap из каталога, где находится `spare.py`. Командная строка под Linux выглядит так:

```
$ pywrap spare.py
```

Когда PyWrap стартует, он попытается импортировать модуль, включенный в командную строку. Затем PyWrap ищет в этом модуле подкласс `wx.App` и создает экземпляр этого класса. После этого PyWrap создает окно `wx.py.crust.CrustFrame` с оболочкой, открывает прикладной объект в дереве пространства имен PyCrust и запускает цикл событий wxPython.

Полный исходный код для PyWrap приведен в листинге 4.3. Это пример того, как много функциональных возможностей можно добавить в Вашу программу, используя небольшое количество дополнительного кодекса.

Листинг 4.3 Исходный код PyWrap.py

```
"""PyWrap is a command line utility that runs a python
program with additional runtime tools, such as PyCrust."""

__author__ = "Patrick K. O'Brien <pobrien@orbtech.com>"
__cvsid__ = "$Id: PyCrust.txt,v 1.15 2005/03/29 23:39:27 robind Exp $"
__revision__ = "$Revision: 1.15 $"[11:-2]

import os
import sys
import wx
from wx.py.crust import CrustFrame

def wrap(app):
    wx.InitAllImageHandlers()
    frame = CrustFrame()
    frame.SetSize((750, 525))
    frame.Show(True)
    frame.shell.interp.locals['app'] = app
    app.MainLoop()
```

```
def main(modulename=None):
    sys.path.insert(0, os.curdir)
    if not modulename:
        if len(sys.argv) < 2:
            print "Please specify a module name."
            raise SystemExit
        modulename = sys.argv[1]
        if modulename.endswith('.py'):
            modulename = modulename[:-3]
    module = __import__(modulename)
    # Find the App class.
    App = None
    d = module.__dict__
    for item in d.keys():
        try:
            if issubclass(d[item], wx.App):
                App = d[item]
        except (NameError, TypeError):
            pass
    if App is None:
        print "No App class was found."
        raise SystemExit
    app = App()
    wrap(app)

if __name__ == '__main__':
    main()
```

После запуска команды PyWrap будут отображены как простой фрейм из `spare.py`, так и фрейм PyCrust.

PyCrust в действии

Теперь давайте посмотрим, что мы можем сделать с прикладным фреймом `spare.py` из оболочки PyCrust. Рисунок 4.10 отображает результат. Сначала мы импортируем `wx` и добавим в наш фрейм панель:

```
>>> import wx
>>> app.frame.panel = wx.Panel(parent=app.frame)
>>> app.frame.panel.SetBackgroundColour('White')
True
>>>
```

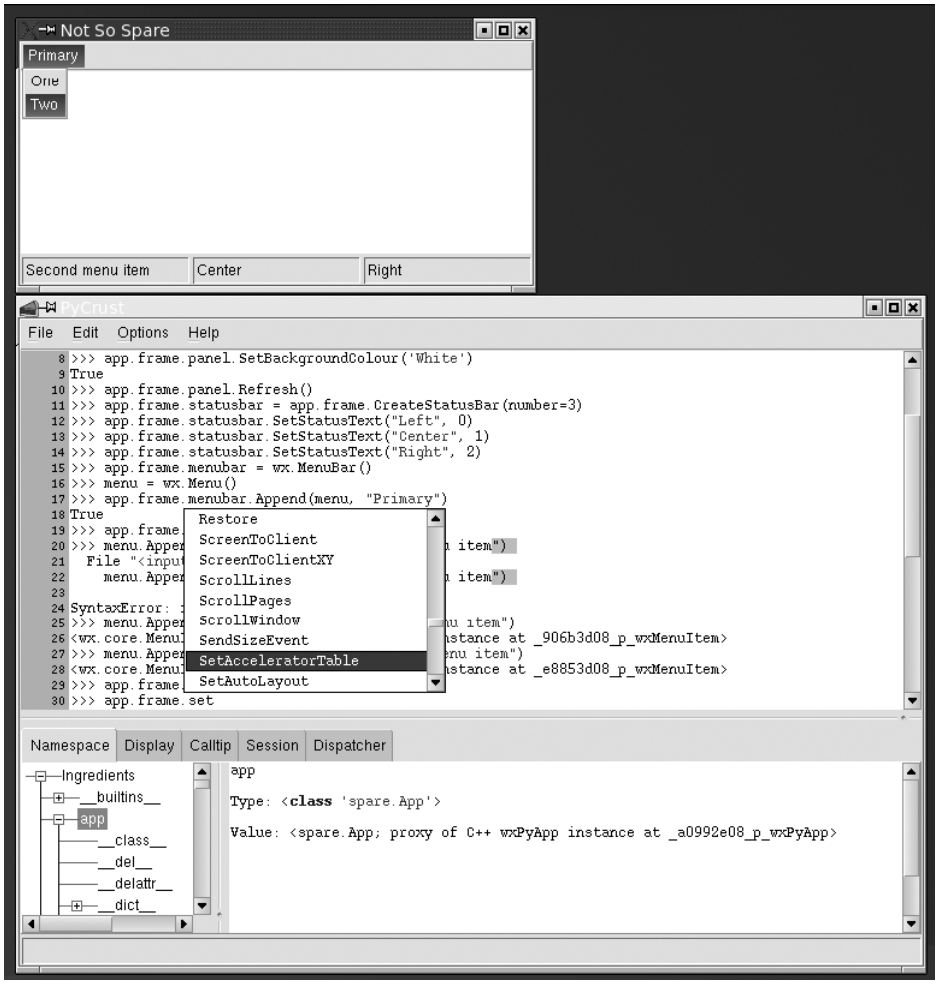

Рисунок 4.10 Использование PyWrap для оптимизации выполнения wxPython-программы

Добавленная к фрейму панель имеет по умолчанию серый цвет, затем он изменен на белый. Однако, установка цвета фона панели не проявляется немедленно. Для этого нужно инициировать событие, которое заставит панель перерисовать себя на основании свойства ее нового фонового цвета. Единственный способ инициировать это событие – потребовать, чтобы панель себя обновила:

```
>>> app.frame.panel.Refresh()
```

Теперь белая панель отображена, и мы на один шаг ближе к пониманию деталей того, как действительно работает wxPython.

Давайте добавим еще и строку состояния (status bar):

```
>>> app.frame.statusbar = app.frame.CreateStatusBar(number=3)
>>> app.frame.statusbar.SetStatusText("Left", 0)
```

```
>>> app.frame.statusbar.SetStatusText("Center", 1)
>>> app.frame.statusbar.SetStatusText("Right", 2)
```

Заметьте, как строка состояния появляется в пределах фрейма, не изменяя самые крайние размеры фрейма. Также, обратите внимание, что добавленный в каждую из трех секций текст строки состояния появляется немедленно и не требует обновления. Теперь давайте добавим меню и строку меню (menubar):

```
>>> app.frame.menubar = wx.MenuBar()
>>> menu = wx.Menu()
>>> app.frame.menubar.Append(menu, "Primary")
True
>>> app.frame.SetMenuBar(app.frame.menubar)
>>> menu.Append(wx.NewId(), "One", "First menu item")
<wx.core.MenuItem; proxy of C++ wxMenuItem instance at
  _d8043d08_p_wxMenuItem>
>>> menu.Append(wx.NewId(), "Two", "Second menu item")
<wx.core.MenuItem; proxy of C++ wxMenuItem instance at
  _40a83e08_p_wxMenuItem>
>>>
```

Когда Вы манипулируете своими wxPython-объектами в оболочке PyCrust, помните о влиянии этих изменений на Вашу выполняемую программу. Попытайтесь ответить на следующие вопросы. Когда меню действительно появляется в пределах фрейма? Какие атрибуты меню Вы можете изменить, пока выполняется программа? Можете ли Вы добавить дополнительные пункты меню? Можете ли Вы удалить их? Можете ли Вы их блокировать? Изучение всех этих возможностей должно помочь лучше понять wxPython и обеспечить Вас большей уверенностью, когда придет время записи фактического программного кода.

Теперь, когда мы потратили большую часть главы на обсуждение PyCrust, мы готовы пойти на прогулку через остальную часть компонентов пакета Py.

4.5 Что еще имеется в пакете Py?

Все программы PyCrust просто используют включенные в пакет Py модули Python, например, такие как `shell.py`, `crust.py`, `introspect.py` и `interpreter.py`. Эти программы являются строительными блоками, которые использованы для создания PyCrust и Вы можете использовать их раздельно или вместе.

Считайте, что PyCrust представляет одно направление трансляции битов и частей функционального наполнения, содержащегося в пакете Py. PyShell представляет другое направление, а PyAlaMode - третье. В каждом из этих случаев, бо́льшая часть основного кода является общей для всех них с некоторым отдаленным изменением контейнеров. Поэтому считайте пакет Py библиотекой модулей, которые Вы можете собрать, как Вам нравится, и где бы Вы хотели отобразить оболочку wxPython, в редакторе кода или в информации времени выполнения в пределах Вашей программы.

Внутри пакета Py существует четкое разделение между модулями, обеспечивающими функциональность интерфейса пользователя и другими

модулями. Такое разделение значительно облегчает использование этих модулей в Ваших собственных программах. Все модули, начинающиеся с Py, например, PyCrust, PyShell, PyAlaMode и PyAlaCarte – это GUI программы конечного пользователя. У вас не будет необходимости импортировать в Ваши программы какой-нибудь из этих модулей. Следующий параграф описывает модули конечного пользователя.

4.5.1 Работа с GUI-программами

Программы пользовательского уровня является пакетами, которые поддерживают модули различными путями. Таблица 4.3 приводит описание программ пользовательского уровня.

Таблица 4.3 Программы конечного пользователя, включенные в пакет Py

Программа	Описание
PyAlaCarte	Простой редактор кода. Редактирует одновременно только один файл.
PyAlaMode	Многофайловый редактор исходного кода. Каждый файл отображается на отдельной странице. Первая страница содержит окно разделителя PyCrust (splitter).
PyCrust	Комбинация оболочки wxPython и блокнотных страниц, отображающих различную информацию времени исполнения, включая вьювер дерева пространства имен.
PyFilling	Простой вьювер дерева пространства имен. Самостоятельно эта программа бесполезна. Она существует просто как еще один пример использования основной библиотеки.
PyShell	Простой интерфейс оболочки wxPython, без дополнительного блокнотного элемента, как в PyCrust. Оболочки wxPython в PyShell и PyCrust функционально идентичны.
PyWrap	Утилита командной строки, которая выполняет имеющуюся wxPython-программу параллельно с фреймом PyCrust и позволяет Вам манипулировать приложением в пределах оболочки PyCrust.

4.5.2 Работа с модулями поддержки

Модули поддержки обеспечивает базовую функциональность программ конечного пользователя, и могут быть импортированы в Ваши собственные программы. Эти модули являются по существу строительными блоками для создания Py-программ пользовательского уровня. Таблица 4.4 содержит перечень модулей поддержки, являющихся частью пакета Py, вместе с их кратким описанием.

Таблица 4.4 Модули поддержки пакета Py

Модуль	Описание
buffer	Поддерживает редактирование файлов
crust	Содержит GUI-элементы, относящиеся к программному приложению PyCrust
dispatcher	Поддерживает службы общей диспетчеризации сигналов (global signal dispatching)
document	Модуль `document` содержит очень простой класс `Document`, который является тонкой надстройкой вокруг файла. Документ хранит различные атрибуты файла, как например, его имя и путь, и обеспечивает методы `read()` и `write()`. Класс `Buffer` обеспечивает эти низкоуровневые операции чтения и записи в экземпляре `Document`.
editor	Содержит все GUI компоненты интерфейса пользователя, которые появляются в программах PyAlaCarte и PyAlaMode.
editwindow	Модуль `editwindow` содержит единственный класс `EditWindow`. Этот класс наследуется от `wx.stc.StyledTextCtrl` (STC) и обеспечивает всеми свойствами, общими для трех основных потребителей STC в пределах пакета Py: оболочки Python, редактора исходного кода и обозревателя исходного кода, используемого в режиме только для чтения.
filling	Содержит все GUI элементы, которые позволяют пользователю управлять пространством имен объектов и отображать информацию времени выполнения этих объектов
frame	Модуль `frame` определяет класс `Frame`, который является базовым классом для всех других фреймов в пределах пакета Py. Он имеет строку состояния (status bar), иконку и меню, которые используются всеми другими классами фреймов. Пункты меню непрерывно себя обновляют, основываясь на текущем статусе и контексте. Таким образом, меню могут быть одинаковыми во всех программах, а не соответствующие для текущей ситуации пункты меню просто отключаются.
images	Модуль `images` содержит использующийся различными Py-программами набор иконок
interpreter	Класс `Interpreter` отвечает за работу списков автозавершения, информацию подсказок о вызове и ключевых кодов, приводящих в действие свойство автозавершения.
introspect	Обеспечивает ряд функций поддержки самоуправляемого типа для вещей подобных подсказкам о вызове и команде автозавершения
pseudo	Модуль `pseudo` определяет файловые классы, позволяющие классу `Interpreter` перенаправлять stdin, stdout и stderr.

shell	Содержит GUI-элементы, определяющие интерфейс оболочки Python в PyCrust, `PyShell` и `PyAlaMode`.
version	Этот последний модуль проще остальных. Он содержит единственную строковую переменную VERSION, которая представляет текущую версию или релиз Py в целом.

Модуль buffer

Модуль `buffer` содержит класс `Buffer`, который поддерживает обычное редактирование файла. Буфер имеет методы такие, как `new()`, `open()`, `hasChanged()`, `save()` и `saveAs()`. Обрабатываемый посредством буфера файл представлен экземпляром класса `Document`, определенного в модуле `document`. Фактическое редактирование содержимого файла происходит посредством одного или более экземпляров класса `Editor`, определенного в модуле `editor`. Буфер действует как посредник между одним или более редакторами и фактически представляет физический файл.

Уникальная характерная особенность класса `Buffer` состоит в том, что каждому экземпляру буфера назначается отдельный экземпляр интерпретатора Python. Эта возможность позволяет использовать буферы в приложениях, которым нужно предоставить функции автозавершения, подсказки о вызове и другие вспомогательные вещи стадии выполнения при редактировании файлов исходного кода Python. Каждый буфер интерпретатора полностью независим и обновляется при вызове метода буфера `updateNamespace()`. Листинг 4.4 приводит исходный код этого метода.

Листинг 4.4 Метод буфера для обновления пространства имен

```
def updateNamespace(self):
    """Update the namespace for autocompletion and calltips.

    Return True if updated, False if there was an error."""

    if not self.interp or not hasattr(self.editor, 'getText'):
        return False
    syspath = sys.path
    sys.path = self.syspath
    text = self.editor.getText()
    text = text.replace('\r\n', '\n')
    text = text.replace('\r', '\n')
    name = self.modulename or self.name
    module = imp.new_module(name)
    newspace = module.__dict__.copy()
    try:
        try:
            code = compile(text, name, 'exec')
        except:
            raise
        try:
            exec code in newspace
```

```
except:
    raise
else:
    # No problems, so update the namespace.
    self.interp.locals.clear()
    self.interp.locals.update(newspace)
    return True
finally:
    sys.path = syspath
    for m in sys.modules.keys():
        if m not in self.modules:
            del sys.modules[m]
```

Данный метод формирует текст в редакторе при помощи встроенного метода Python `compile`, а затем выполняет его, используя ключевое слово `exec`. Если компиляция успешна, результат устанавливает количество переменных в пространстве имен `newspace`. В результате выполнения, при обновлении локального пространства имен интерпретатора, обеспечивается доступ интерпретатора к любым определенным в буфере редактора классам, методам или переменным.

Модуль crust

Модуль `crust` содержит шесть характерных для программного приложения PyCrust элементов GUI. Основной класс `CrustFrame` является подклассом `wx.Frame`. Если Вы просмотрите листинг 4.1, то можете увидеть, как программа PyWrap импортирует `CrustFrame` и создает его экземпляр. Это самый простой способ вставить фрейм PyCrust в Вашу собственную программу. Если Вы хотите что-то меньшее, чем целый фрейм, можете использовать один или несколько классов, указанных в таблице 4.5.

Таблица 4.5 Классы, определенные в модуле crust

Класс	Описание
Crust	Основан на `wx.SplitterWindow` и содержит оболочку и блокнотные страницы с runtime-информацией
Display	Стилизованный текстовый элемент управления для отображения объекта при помощи Pretty Print
Calltip	Текстовый элемент управления, содержащий последние подсказки о вызове оболочки
SessionListing	Текстовый элемент управления, содержащий все команды сеанса
DispatcherListing	Текстовый элемент управления, содержащий все сообщения сеанса
CrustFrame	Фрейм, содержащий окно разделителя `Crust`

Эти GUI-элементы интерфейса могут быть использованы в любой программе wxPython и обеспечивают полезными интроспективными визуальными возможностями.

Модуль dispatcher

Диспетчер обеспечивает услуги глобальной обработки сигналов. Это означает, что он действует как посредник, разрешая объектам посылать и получать сообщения, без необходимости что-либо знать друг о друге. Все что им нужно знать - это посылаемый сигнал (обычно простая строка). Один или несколько объектов могут запросить, чтобы диспетчер уведомлял их всякий раз, когда сигнал посылается, и один или несколько объектов могут указать диспетчеру послать некоторый конкретный сигнал.

Листинг 4.5 иллюстрирует, почему диспетчер такой полезный. Поскольку все Py-программы построены на одних и тех же основных модулях, оба, и PyCrust и PyShell используют почти идентичный код. Единственное различие в том, что PyCrust включает блокнотный элемент с дополнительными функциями, подобно дереву пространства имен, обновляемый всякий раз, когда интерпретатору посылаются команды. Интерпретатор использует диспетчера для посылки сообщения (сигнала) всякий раз, когда через него продвигается команда:

Листинг 4.5 Код для посылки команды с помощью модуля dispatcher

```
def push(self, command):
    """Send command to the interpreter to be executed.

    Because this may be called recursively, we append a new list
    onto the commandBuffer list and then append commands into
    that. If the passed in command is part of a multi-line
    command we keep appending the pieces to the last list in
    commandBuffer until we have a complete command. If not, we
    delete that last list."""
    command = str(command) # In case the command is unicode.
    if not self.more:
        try: del self.commandBuffer[-1]
        except IndexError: pass
    if not self.more: self.commandBuffer.append([])
    self.commandBuffer[-1].append(command)
    source = '\n'.join(self.commandBuffer[-1])
    more = self.more = self.runsource(source)
    dispatcher.send(signal='Interpreter.push', sender=self,
                    command=command, more=more, source=source)
    return more
```

Различные задействованные части модулей `crust` и `filling` самостоятельно настраиваются на получение этого сигнала, связываясь с диспетчером в своих конструкторах. Листинг 4.6 содержит полный исходный код элемента `SessionListing`, размещенного в PyCrust на странице Session:

Листинг 4.6 Код сеансовой страницы PyCrust

```
class SessionListing(wx.TextCtrl):
    """Text control containing all commands for session."""

    def __init__(self, parent=None, id=-1):
        style = (wx.TE_MULTILINE | wx.TE_READONLY |
                 wx.TE_RICH2 | wx.TE_DONTWRAP)
        wx.TextCtrl.__init__(self, parent, id, style=style)
        dispatcher.connect(receiver=self.push,
                           signal='Interpreter.push')

    def push(self, command, more):
        """Receiver for Interpreter.push signal."""
        if command and not more:
            self.SetInsertionPointEnd()
            start, end = self.GetSelection()
            if start != end:
                self.SetSelection(0, 0)
            self.AppendText(command + '\n')
```

Обратите внимание, как получатель `SessionListing` (его метод `push()`) игнорирует направленные интерпретатором параметры `sender` и `source`. Диспетчер в этом смысле очень гибкий и только посылает принимаемые получателями параметры.

Модуль editor (редактор)

Модуль `editor` содержит все компоненты редактирования GUI, которые используются в программах PyAlaCarte и PyAlaMode. Если Вам необходимо включить в свою программу редактор исходного кода Python, используйте классы, описанные в таблице 4.6.

Эти классы могут быть использованы в любой программе, где необходимы функциональные возможности практичного и стильного редактирования кода.

Таблица 4.6 Классы, определенные в модуле editor

Класс	Описание
EditorFrame	`EditorFrame` является подклассом более общего класса `Frame` из модуля `frame` и используется PyAlaCarte для поддержки редактирования одного файла за раз.
EditorNotebookFrame	Подкласс класса `EditorFrame`, расширяющий его интерфейсом блокнотных страниц и возможностью одновременного редактирования нескольких файлов. Этот фреймовый класс используется PyAlaMode.
EditorNotebook	Этот элемент используется фреймом `EditorNotebookFrame` для отображения каждого файла на отдельной странице.

Editor	Обслуживает взаимодействие между буфером и ассоциированным `EditWindow`.
EditWindow	Основанный на `StyledTextCtrl` элемент редактирования текста.

Модуль filling (инспектирование)

Модуль `filling` содержит все элементы управления GUI, которые позволяют пользователю передвигаться по пространству имен объектов и отображать информацию времени выполнения этих объектов. Четыре определенных в модуле `filling` класса описаны в таблице 4.7.

Таблица 4.7 Классы, определенные в модуле filling

Класс	Описание
FillingTree	Базируется на `wx.TreeCtrl` и представляет иерархическое дерево объектов пространства имен
FillingText	Подкласс `editwindow`. `EditWindow` используется для отображения информации о текущем выбранном в FillingTree объекте.
Filling	`wx.SplitterWindow`, содержащий слева FillingTree, а справа FillingText.
FillingFrame	Фрейм, содержащий окно инспектора объектов `Filling`. Двойной щелчок на пункте дерева filling открывает новый `FillingFrame`, с этим выбранным пунктом в корне дерева.

Использование этих классов в Вашей программе позволит легко создать иерархическое дерево пространства имен Python. Оно может быть использовано в качестве быстрого обозревателя данных, если Вы настраиваете эти данные, как объекты Python.

Модуль interpreter (интерпретатор)

Определенный в классе `Interpreter` модуль `interpreter` основан на классе `InteractiveInterpreter` модуля `code` из стандартной библиотеки Python. Кроме передачи исходного кода в Python, класс `Interpreter` также отвечает за обеспечение списков автозавершения, информации подсказок о вызове и даже за обработку кодов клавиш, инициирующих автозавершение (обычно это код клавиши точки ".").

Из-за такого явного разделения ответственности, Вы можете создать собственный подкласс интерпретатора и передать его экземпляр в оболочку PyCrust, а не в стандартный интерпретатор. Это сделано в нескольких программах, для поддержки пользовательских языковых изменений и получения преимуществ среды PyCrust. Например, одна программа подобного типа позволяет управлять лабораторным оборудованием из встроенной оболочки PyCrust. Эта программа

использует ведущий слэш (/), чтобы инициировать автозавершение всякий раз, когда после ссылки на одну из частей оборудования появляется ведущий слэш. Появляющаяся опция автозавершения характеризует эту часть оборудования, как оно настроено и в каком текущем состоянии находится.

Модуль introspect

Модуль `introspect` используется классами `Interpreter` и `FillingTree`. Он обеспечивает ряд функций поддержки интроспективного типа для подсказок о вызове и команд автозавершения. Далее показано использование `wx.py.introspect` для получения всех названий атрибутов объекта списка, исключая атрибуты с ведущими двойными подчеркиваниями:

```
>>> import wx
>>> L = [1, 2, 3]
>>> wx.py.introspect.getAttributeNames(L, includeDouble=False)
['append', 'count', 'extend', 'index', 'insert', 'pop',
'remove', 'reverse', 'sort']
>>>
```

Функция `getAttributeNames()` используется классом `FillingTree` для инспектирования иерархии пространства имен. Один из лучших способов понять модуль `introspect` состоит в том, чтобы взглянуть на тесты устройства, которые он успешно проходит. Просмотрите файл `test_introspect.py` в директории установки Python Lib/site-packages/wx/py/tests.

Модуль shell

Модуль `shell` содержит GUI элементы, которые определяют интерфейс оболочки Python внутри программ PyCrust, PyShell и PyAlaMode. Таблица 4.8 приводит описание каждого элемента. Основным классом является `ShellFrame`, подклассом - `frame.Frame`. Он содержит экземпляр класса `Shell`, выполняющий основной объем работы по обеспечению диалоговой среды Python.

Таблица 4.8 Классы, определенные в модуле shell

Класс	Описание
Shell	Основанная на `wx.stc.StyleTextCtrl` Python-оболочка. `Shell` наследует `editwindow`, а `EditWindow` в свою очередь преодолевает ряд трудностей, для того чтобы основной элемент управления текстом вел себя подобно оболочке Python, а не редактору файла исходного кода.
ShellFacade	Упрощенный интерфейс ко всем связанным с оболочкой функциональным возможностям. Полупрозрачный внешний вид, при котором остаются доступными атрибуты реальной оболочки, хотя пользователю из оболочки видны только некоторые из них.
ShellFrame	Фрейм, содержащий окно `Shell`

Класс `ShellFacade` был создан при разработке оболочки PyCrust для упрощения доступа из нее к объекту оболочки. Когда Вы запускаете PyCrust или PyShell, экземпляр класса `Shell` становится доступным в оболочке Python. Например, следующим образом Вы можете вызвать в приглашении оболочки метод оболочки `about()`:

```
>>> shell.about()
Author: "Patrick K. O'Brien <pobrien@orbtech.com>"
Py Version: 0.9.4
Py Shell Revision: 1.7
Py Interpreter Revision: 1.5
Python Version: 2.3.3
wxPython Version: 2.4.1.0p7
Platform: linux2
>>>
```

Поскольку оболочка Shell наследует `StyledTextCtrl`, она содержит свыше 600 атрибутов. Большинство атрибутов используются в приглашении оболочки, так что `ShellFacade` был создан, чтобы ограничивать количество атрибутов, появляющихся в списке автозавершения при вводе на запрос оболочки. Теперь объект оболочки отображает только около 25 наиболее полезных атрибутов оболочки. Если Вы хотите использовать некоторый атрибут, который не включен в список автозавершения, можете его ввести, и он будет направлен обычной оболочке, сохраненной в качестве атрибута внешнего вида.

4.6 Как использовать модули пакета Py в wxPython-программах?

Что делать, если в Вашем приложении Вам не нужен целый фрейм PyCrust? Что, если Вам просто нужен интерфейс оболочки в одном фрейме, и возможно вьювер пространства имён в другом? И что, если Вы хотите, чтобы они были постоянными дополнениями Вашей программы? Такие альтернативы не только возможны, они также довольно легко достигаются. Мы закончим эту главу примером того, как это может быть сделано. Мы вновь вернемся к созданной нами в главе 2 программе, которая имела строку меню, панель инструментов и строку состояния. Мы добавим другое меню с одним пунктом для отображения фрейма оболочки и другим пунктом для отображения фрейма инспектора объектов (filling frame). И, наконец, мы установим корень дерева объектов на объект фрейма из нашей основной программы. Результат показан на рисунке 4.11.

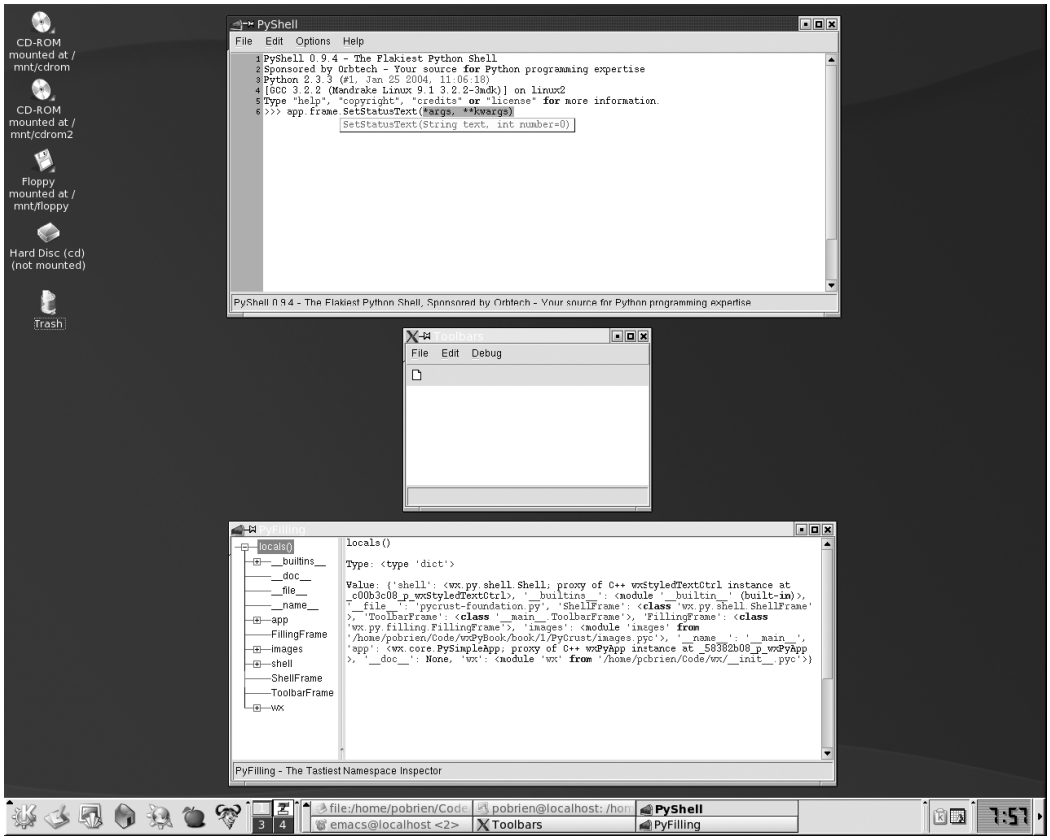

Рисунок 4.11 Базовая программа с фреймами оболочки и инспектирования

Листинг 4.7 показывает модифицированный исходный код (Для объяснения первоначальной программы обратитесь к главе 2). Как видите, чтобы добавить возможность запускать фрейм оболочки и фрейм инспектора объектов с каждым функционирующим фреймом основного приложения, использована всего пара дополнительных строк кода.

Листинг 4.7 Базовая программа с дополнительными работающими инструментами

```
#!/usr/bin/env python

import wx
from wx.py.shell import ShellFrame
from wx.py.filling import FillingFrame
import images

class ToolbarFrame(wx.Frame):

    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, 'Toolbars',
            size=(300, 200))
```

1 Импортирование фреймовых классов

```
        panel = wx.Panel(self, -1)
        panel.SetBackgroundColour('White')
        statusBar = self.CreateStatusBar()
        toolbar = self.CreateToolBar()
        toolbar.AddSimpleTool(wx.NewId(), images.getNewBitmap(),
            "New", "Long help for 'New'")
        toolbar.Realize()
        menuBar = wx.MenuBar()
        menu1 = wx.Menu()
        menuBar.Append(menu1, "&File")
        menu2 = wx.Menu()
        menu2.Append(wx.NewId(), "&Copy", "Copy in status bar")
        menu2.Append(wx.NewId(), "C&ut", "")
        menu2.Append(wx.NewId(), "Paste", "")
        menu2.AppendSeparator()
        menu2.Append(wx.NewId(), "&Options...", "Display Options")
        menuBar.Append(menu2, "&Edit")
```

Создание меню Debug и его элементов

```
        menu3 = wx.Menu()
        shell = menu3.Append(-1, "&wxPython shell",
                             "Open wxPython shell frame")
        filling = menu3.Append(-1, "&Namespace viewer",
                               "Open namespace viewer frame")
        menuBar.Append(menu3, "&Debug")
        self.Bind(wx.EVT_MENU, self.OnShell, shell)
        self.Bind(wx.EVT_MENU, self.OnFilling, filling)
        self.SetMenuBar(menuBar)

    def OnCloseMe(self, event):
        self.Close(True)

    def OnCloseWindow(self, event):
        self.Destroy()

    def OnShell(self, event):
        frame = ShellFrame(parent=self)
        frame.Show()

    def OnFilling(self, event):
        frame = FillingFrame(parent=self)
        frame.Show()

if __name__ == '__main__':
    app = wx.PySimpleApp()
    app.frame = ToolbarFrame(parent=None, id=-1)
    app.frame.Show()
    app.MainLoop()
```

2

3

Установка обработчико в событий меню

4

Обработчик пункта меню OnShell

Обработчик пункта меню OnFilling

1. Здесь мы импортируем классы `ShellFrame` и `FillingFrame`.
2. Мы добавляем пункты в наше третье меню Debug (Отладка) так же, как и в предыдущих двух меню и добавляем его в строку меню фрейма.

3. Связывание функции с `wx.EVT_MENU()` позволяет нам ассоциировать с пунктом меню обработчик, вызываемый при выборе этого пункта меню.

4. Когда пользователь выбирает оболочку Python в меню Debug, создается фрейм оболочки, наследуемый от фрейма панели инструментов (toolbar). Когда фрейм инструментальной панели закрывается, также закрываются любые открытые фреймы оболочки и инспектора объектов.

4.7 Резюме

- Пакеты разработчика подобные wxPython по своей природе очень велики и сложны. Взаимодействие между GUI-элементами не всегда интуитивно, ход этого процесса определен событиями и ответами на события, а не линейной последовательностью выполнения. Использование инструментальных средств подобных оболочке PyCrust может существенно повысить Ваше понимание этой событийно-управляемой среды.
- PyCrust - просто другая оболочка Python, подобная оболочкам, включенным в IDLE, Boa Constructor, PythonWin и другие инструментальные средства разработки. Тем не менее, PyCrust был создан с помощью wxPython, применение которого оправдывается, когда Вы разрабатываете программы на wxPython. В частности, у Вас не будет проблем с конфликтами событийных циклов, и Вы сможете манипулировать всеми аспектами Вашей программы при ее выполнении в рамках оболочки PyCrust и обозревателя пространства имен.
- Поскольку PyCrust - часть дистрибутива wxPython, он устанавливается вместе с wxPython, включая весь исходный код. Это облегчает использование PyCrust, нивелируя трудности изучения и обеспечивая интроспективную функциональность Ваших программ.
- Кроме того, модульный проект пакета Py облегчает выбор необходимых для Вашей программы модулей, например, для редактирования исходных текстов, просмотра пространства имен или для обеспечения функциональности оболочки.
- PyCrust помогает Вам контролировать ключевые моменты поведения Вашей программы при ее выполнении.

В следующей главе мы используем накопленные знания о wxPython, а также получим некоторую практическую помощь в том, как структурировать Ваши GUI-программы и при этом не запутаться.