



Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Computersysteme

Jürg Gutknecht

The Oberon Guide
System Release 1.2

October 1990

Операционная система Oberon 1.2

Руководство пользователя

Юрг. Гуткнехт

Октябрь 1990г.

Оглавление

Аннотация.....	6
Введение.....	6
Принципы проектирования.....	6
Благодарность.....	7
Команды и текст.....	7
Интеркликинг	9
Пакет инструментов редактирования Edit Tool Package.....	9
Команды мыши.....	10
Позиционирование текста	10

Размещение каретки.....	10
Выделение текста.....	10
Копирование текста.....	11
Копирование атрибутов.....	11
Удаление текста.....	11
Активация команды.....	11
Вьювер-ловушек TRAP.....	12
Команды редактирования.....	12
Edit.Open.....	12
Edit.Show M.X.....	12
Edit.Store.....	12
Edit.Recall.....	13
Edit.CopyFont.....	13
Edit.ChangeFont.....	13
Edit.ChangeColor.....	13
Edit.ChangeOffset.....	13
Edit.Search.....	13
Edit.Locate.....	13
Edit.PrintServerName ["%"] {(ИмяФайла "*") ["/" NofCopies]}..	13
Пакет инструментов рисования Draw Tool Package.....	14
Draw.Open	14
Создание надписи.....	14
Выбор	14
Выделенные линии	15
Перемещение	15
Копирование	15
Команды меню.....	15
Draw.Cleanup	15
Draw.Delete	15
Draw.Store	15
Макросы.....	16
Draw.Macro lib mac	16
Draw.Directory libfile	16
Draw.OpenMacro lib mac	16
Draw.MakeMacro lib mac	16
Дополнительные команды.....	17
Draw.Reset	17
Draw.Store name	17
Draw.Pint Serername *	17
Draw.Print Severname (Filename)	17
Draw.SetGrid n	17
Draw.CarPos	17
Draw.Makelib lib flname	17
Rectangles.New.....	18
Circles.New.....	18
Установка системы Draw.....	18
Пакет инструментов Paint.....	18
Paint.Open {/w h}.....	19
Paint.Zoom.....	19
Paint.Print.....	19
Paint.Scale.....	19
Paint.Drow	19
Paint.SetColor белый черный белый инверт.....	19
Paint.SetPen w h.....	20
Paint.Fill n.....	20
Paint.SetGrid n.....	20
Paint.SetFont FontName.....	20
Пакет системных инструментов System Tool Package.....	20
System.Open.....	20

System.OpenLog.....	20
System.Copy.....	20
System.Grow.....	21
System.Close.....	21
System.CloseTrack.....	21
System.Recall.....	21
System.Time.....	21
System.Watch.....	21
System.Collect.....	21
System.Free.....	21
System.ShowModules.....	21
System.ShowCommands ModName.....	21
System.State ModName.....	21
System.SetUser.....	22
System.Directory.....	22
System.CopyFiles.....	22
System.RenameFiles.....	22
System.DeleteFiles.....	22
System.SetFont.....	22
System.SetColor.....	22
System.SetOffset.....	22
Пакет инструментов компилятора Compiler Tool Package.....	22
Compiler.Compile.....	23
Пакет различных инструментов Miscellaneous Tool Package.....	23
Miscellaneous.BoatLoad FileName.....	23
Miscellaneous.ConvertBlanks.....	23
Miscellaneous.ConvertTabs.....	23
Miscellaneous.Cleanup.....	23
Miscellaneous.CountLines.....	23
Miscellaneous.GetObjSize.....	24
Miscellaneous.Snapshot.....	24
Miscellaneous.Trap.....	24
Пакет инструментов резервного копирования Backup Tool Package .	24
Backup.Format.....	24
Backup.Init.....	24
Backup.Directory.....	24
Backup.DeleteFiles.....	24
Backup.ReadAll.....	24
Backup.ReadFiles.....	25
Backup.WriteFiles.....	25
Backup.ConvertToMSDOS.....	25
Backup.ConvertFormMSDOS.....	25
Пакет инструментов Net Server.....	25
Net.Mailbox.....	25
Net.DeleteMail.....	25
Net.Receivetail.....	25
NetSendMail.....	25
NetSendFiles.....	26
Net.ReceiveFiles.....	26
Net.DeleteFies.....	26
Net.SendMsg <сообщение партнера>.....	26
Net.GetTime.....	26
Net.SetPassword server <пароль>.....	26
Net.StartServer.....	26
Net.StopServer.....	27
Net.Unprotect.....	27
Ne.WProtect.....	27
Пакет инструментов ColorSystem.....	27
ColorSystem.Init.....	27

ColorSystem.Colors.....	27
ColorSystem.ShowColors.....	27
ColorSystem.LoadColors <Имя>.....	27
ColorSystem.StoreColors Имя.....	27
Colorsystem.SetCursor x.....	27
Иерархия модулей Oberon.....	28
Определения модулей.....	29
Система отображения Display System.....	29
DEFINITION Display; (* драйвер дисплея *) ..	29
DEFINITION Viewers; (* менеджер просмотра *).....	31
DEFINITION MenuViewers;.....	32
Текстовая система.....	33
DEFINITION Fonts; (* загрузчик шрифта *).....	33
DEFINITION Texts; (* менеджер текстов *).....	34
DEFINITION TextFrames; (* отображение текста *).....	37
Ядро Oberon.....	40
DEFINITION Math; (* библиотека для вещественных чисел *).....	40
DEFINITION MathL; (* библиотека для longreals *).....	40
DEFINITION Files; (* менеджер файлов *).....	40
DEFINITION Diskette; (* менеджер дискет *).....	41
DEFINITION Input; (* драйверы клавиатуры и мыши *).....	42
DEFINITION SCC; (* драйвер SCC *).....	42
DEFINITION V24; (* драйвер V24*).....	43
DEFINITION Printer; (* интерфейс принтера *).....	44
DEFINITION Oberon; (* менеджер системы *).....	44
Примеры.....	48
Запись метки времени в системный журнал.....	48
Обработка выделенного текста.....	48
Открыть вьювер в системном треке, генерирование и отображение текстовых данных.....	49
Открыть вьювер в пользовательском треке и отобразить существующий текст.....	50
Grow-отображение.....	51
Обработка текста вьювера или последовательности текстов в зависимости от контекста.....	51
Удаление выделенной части текста в помеченном вьювере.....	52
Копирование последнего выделенного фрагмента текста в позицию каретки.....	52
Копирование шрифта из видимой отмеченной позиции в выделенный текст.....	53
Переместить каретку на следующий символ выделенный курсивом. .	53
Программирование новых классов фремов и типов вьюверов.....	54
Фреймы как активные объекты.....	54
Каноническое разложение приложения.....	55
Учебный пример 1: Вьюверы текста.....	56
Учебный пример 2: Операции, ориентированные на фрейм.....	63
Отображение текстовой строки в текстовом фрейме.....	63
Каретка трека.....	64
Учебный пример 3: Операции, ориентированные на работу с текстом	65
Сохранение текста в буфере.....	65
Вставка содержимого буфера в текст.....	66
Литература.....	67
Рабочая станция Ceres.....	67
Язык Oberon.....	68
Система Oberon.....	68
Список номеров ошибок	69
Неправильное использование языка Oberon-2.....	69
Ограничения реализации.....	72

Аннотация

Это руководство содержит краткое описание системы Oberon на трех уровнях: уровень пользователя, уровень программирования инструментов и уровень реализации новых классов вьюверов. Руководство содержит полную документацию стандартных команд, прокомментированы определения интерфейса и обучающую коллекцию программ Oberon, демонстрирующих типичный стиль программирования Oberon.

Введение

История

Oberon – это одновременно название языка программирования и операционной системы. Проект был начат Никлаусом Виртом в 1985 году с целью разработки новой переносимой операционной системы для рабочих станций. Результатом проекта стала реализация системы и языка для компьютера Ceres (см. раздел Литература).

Расширение языка Oberon стало неизбежно, поскольку система имеющихся типов оказалась слишком ограниченной, чтобы выразить желаемую модель данных естественным и безопасным способом. Для знакомства с новым языком мы отсылаем Вас к отчету, а также к третьей и четвертой главе этого текста за некоторыми примерами его применения.

Принципы проектирования

Основополагающая динамическая модель чрезвычайно проста. Существует один процесс, в качестве общего носителя множества задач. Этот процесс многократно интерпретирует команды, которые являются официальными объектами выполнения в Oberon. Команды – это атомарные (неделимые) действия, влияющие на глобальное состояние системы. В отличие от обычных интерактивных программ, они строго избегают прямых диалогов с пользователем системы.

Следующие типичные примеры охватывают понятие "команда": Помещение каретки, вставка символа в текст, выделение фрагмента текста, удаление выделенного фрагмента текста, применение нового шрифта к фрагменту текста, поиск шаблона в тексте, компиляция программного модуля, открытие окна просмотра (вьюера), резервное копирование файлов на дискету и отображение каталога. Мы подчеркиваем, что выполнение команды всегда приводит к получению энергонезависимой информации. Например, в последнем примере это означает, что отображаемый каталог – это текст, который может сразу же подвергнуться дальнейшей обработке. Команды сообщают о результатах своего выполнения в виде записи в системном журнале (System log). Журнал предоставляет полный протокол текущего сеанса работы.

Команды инициируются входными действиями. За исключением нескольких универсальных операций, каждое входное действие связано с отображаемым вьювером, которому передается его дальнейшая обработка. Вьювер в Oberon –

это прямоугольная область на экране, которая может отображать любые данные. Большинство вьюеров имеют тонкую рамку и заголовок, содержащий меню. Мышь, обрабатывается вьюером, на который она указывает. Данные с клавиатуры передаются в текущий так называемый "фокус-вьюер". Мы заметили, что интерпретация команд является высоко децентрализованной деятельностью в Oberon и является существенным вкладом в то, что мы считаем самым важным качеством Oberon, а именно неограниченную расширяемость.

Реализация нового типа вьюера – это очень мощный и довольно далеко идущий метод расширения системы Oberon. Он уместен только в сочетании с установкой нового класса отображаемых объектов (например, графики или таблиц). Более подробно эта тема будет рассмотрена в главе "Программирование новых классов фремов и типов вьюеров".

Более умеренный способ расширения функциональности системы заключается в добавлении новых команд, действующих над объектами уже существующего класса (например, компилятор языка, работающий с текстом). В третьей главе мы увидим, что открытая и согласованная модульная архитектура Oberon обеспечивает эффективную поддержку для этого. Все компоненты и ресурсы системы доступны и могут быть использованы через модульные интерфейсы на таком высоком уровне абстракции, насколько это возможно.

Из вышесказанного мы должны сделать вывод, что в Oberon нет стены, отделяющей пользователя от разработчика. Пользователей поощряют настраивать систему и приспособлять ее к своим индивидуальным потребностям путем разработки и внедрения своих команд и средств. В системе мало что "жестко запрограммировано". Тем не менее, существует несколько общих соглашений и существующих инструментов. Они представлены в следующей главе.

Благодарность

Мы с благодарностью признаем инициативу и готовность Никлауса Вирта пройти через приключения, связанные с проектированием и создания новой рабочей станции, нового языка и новой операционной системы, особенно в условиях жестких ограничений личных ресурсов. Без его компетентности и чрезвычайной приверженности этот грандиозный проект не был бы успешно завершен. Я также очень благодарен за возможность включить выдержки Никлауса Вирта из руководства по Draw. Моя благодарность также выражается сообществу пользователей Oberon, в частности Мартину Райзеру, Хансу-Петеру Мссенбеку и Беверли Сандерс за их конструктивную критику и ценные предложения по улучшению.

Команды и текст

Среди классов возможных объектов, с которыми может работать система Oberon, класс текстов играет ключевую роль. Не только входные и выходные данные часто представляются в виде текста, но и объекты и команды часто идентифицируются по имени. Таким образом, текст является предопределенным классом объектов в Oberon. Это видно сразу, после запуска системы, когда, помимо журнала System Log, на экране открывается так называемый Tool-вьюер (Инструменты). Он содержит список команд, за которыми следуют параметры (или не следуют). Имена команд в Oberon имеют вид M.P, где M обозначает модуль а P процедуру, которую экспортирует модуль. Пользователь активирует команду, с помощью мыши, нажав среднюю клавишу мыши (СКМ). Например, активация команды Edit.Open приведет к появлению нового вьюера для работы с текстом.

Часто команда имеет вводные параметры. Например, для открытия другого инструмента требуется указание его имени, как System.Open Edit.Tool или System.Open Net.Tool. Хотя это далеко не самый общий случай спецификации

параметров. Некоторые команды принимают целый список имен, следующих за именем команды, и выполняется каждый член списка. Текст, подчиняющийся произвольному синтаксису (понимаемому командой) может быть передан одинаково хорошо. Команды могут даже ожидать в качестве параметров объекты любого типа, существующие в системе, такие как вьюеры, выделение текста, каретка и глобальный указатель "звезда". Некоторые команды даже допускают различные способы спецификации параметров. Например, если команда `System.Open` или `Edit.Open` вызывается с символом "^", а не с именем, следующим за командой, то имя берется из последнего выделения. В общем случае символ "^" следующий за именем команды, всегда относится к текущему выделенному объекту (тексту и т.д.).

Примечательно, что инструменты (Tool) – это обычные тексты, отличающиеся от обычных текстов только своей структурой и содержанием. Инструменты поддаются редактированию. Но мы понимаем, что команды, вполне могут проскользнуть в прозаический текст и могут быть там активированы. Очевидно, что нет никаких ограничений для фантазии, эксплуатирующему эту универсальную систему интерпретации команд.

Одним из умеренных применений универсальной схемы, рассмотренной выше, является построение взаимосвязанных текстов. Собственно говоря, набор стандартных инструментов структурирован в виде дерева с `System.Tool` в качестве предка и инструментами в `System.Tool` в качестве потомков. Напомним, что иерархическая система инструментов может быть легко настроена "на лету" путем корректировки списков команд (включая параметры) в соответствии с персональными требованиями, – изменения конфигурации иерархии инструментов, установки новых инструментов или даже предоставления он-лайн документации.

Теперь мы обсудим операции редактирования. Напомним, что большинство команд интерпретируются вьюерами по отдельности. Однако есть несколько более универсальных операций, которые обрабатываются непосредственно системой. Например, когда вы набираете `есаре`, все метки на дисплее удаляются, включая каретку и текстовые выделения. Или, когда вы набираете `Ctrl-Shift-Del`, система немедленно завершает выполнение текущей команды и открывается вьюер-ловушек (TRAP), отображающее состояние прерванного процесса. Обратите внимание, что мы только что идентифицировали вас (читателя этого руководства) с пользователем системы. Для упрощения формулировок мы впредь будем делать это негласно.

Далее мы перейдем к вьюерам и операциям, специфичным для конкретного дисплея. Вы можете перевести свой экран в один из трех различных режимов, нажав одну из функциональных клавиш `PF1`, `PF2`, `PF3` для Ceres и `F1`, `F2`, `F3` для остальных машин:

`F1` задает белый шрифт на черном фоне,
`F2` выключает дисплей,
`F3` задает черный шрифт на белом фоне.

В Oberon используется плиточная система просмотра. Дисплей разделен на вертикальные треки, и каждый трек далее подразделяется на вьюеры. В действительности структура вьюеров трехмерна. Новый трек может фактически накладываться на один или на несколько существующих треков. Первоначальная конфигурация будет восстановлена, когда наложенный трек впоследствии будет удален.

Хотя глобальная схема экрана может быть изменена, мы относим наши текущие объяснения к стандартной схеме: В ней показаны два трека, большой, пользовательский трек слева и более узкий, системный трек, справа. Треки распределяются автоматически соответствующими командами с использованием небольшой эвристики. Например, инструментальные средства, открываются в системном треке, а редактируемые в треке пользователя. Однако вы можете отменить любое автоматическое распределение, сначала поместив указатель в то место, где вы хотите разместить верхнюю часть нового вьюера. Указатель представляет собой маркер "звездочка", который помещается

путем перемещения мыши в нужное место и нажатием клавиши F1 (отмена - F2). Для того чтобы изменить размер вьювера, наведите указатель на строку заголовка, нажмите левую клавишу мыши и переместите мышь вверх или вниз. Вы также можете переместить вьювер в любое другое место на экране, начав, как было описано выше, затем зажмите среднюю клавишу мыши, перетащите мышь в новое место и отпустите все клавиши там.

Интеркликинг

Интеркликинг - означает нажатие и отпускание (клик) вторичной кнопки мыши, в то время как первичная кнопка мыши нажата. Первичной, либо вторичной кнопкой, могут быть любые из трех кнопок мыши. Интеркликинг - это эффективный инструмент, позволяющий многократно увеличить производительность пользователя. В Oberon интеркликинг применяется как основа работы с системой. Подробнее об этом вы узнаете в следующих главах. Нажатие ЛКМ и интерклик СКМ отменяет последнее действие.

По традиции, большинство вьюверов показывают в заголовке, меню со списком команд. Эти команды автоматически ссылаются на свой вьювер. В случае обычных текстовых вьюверов, команды включаются из пакета инструментов System и пакета инструментов Edit:

```
System.Close
    удаляет вьювер.
Edit.Copy
    открывает новый вьювер, отображающий тот же экземпляр текста.
Edit.Grow
    позволяет увеличить вьювер до полного размера или всего дисплея.
Edit.Locate
    определяет местоположение ошибки в исходном тексте.
Edit.Store
    сохраняет текст в файле.
```

Более подробно эти команды будут описаны в следующих разделах, посвященных инструментам.

Сначала мы определим некоторую терминологию и общие соглашения. Мы будем называть видимым или невидимым объект, если он помечен указателем. Видимость указателя не имеет значения. В качестве исключения это явное выделение вьювера, которое требует, чтобы указатель "звезда" был видимым. Обратите внимание, что явно выделенный вьювер автоматически помечается и указатель помещен в левый нижний угол дисплея и почти невидим.

По соглашению, заголовок вьювера обычно является либо именем отображаемого объекта, либо именем команды которая открыла этот вьювер. Кроме того, если команде в качестве параметра передается список имен, он должен быть завершен символом, отличным от имени, например, символом "~". По другому соглашению, правый верхний угол дисплея отводится для просмотра журнала, сообщающего о ходе и результатах выполнения команды. Наконец, существует несколько стандартных расширений имен файлов. Среди них .Text, .Graph и .Pict для текстов, графики и растровых изображений соответственно, и .Sen.Fnt для файлов экранных шрифтов:

В следующих разделах мы будем использовать термины параметр и список параметров в ограниченном смысле: "элемент следующий за именем команды" и "список элементов, следующий за именем команды" соответственно.

Пакет инструментов редактирования Edit Tool Package

Ранее мы уже говорили, что расширяемость была ключевой целью при разработке системы Oberon. Поэтому было заманчиво реализовать также

системно-ориентированные команды как расширения ядра системы на максимально высоком уровне модульной иерархии, тем самым достигая максимальной гибкости. Такая стратегия особенно подходит для редактирования текста. Она проявляется в существовании пакета инструментов редактирования, предоставляющего расширяемый набор команд редактирования. Тем не менее, несколько встроенных команд интерпретируются непосредственно текстовыми фреймами. К ним относятся позиционирование текста во вьювере, установка каретки, вставка вводимого символа, выделение части текста, удаление выделенной части текста, копирование выделенного фрагмента текста, копирование атрибутов и самое главное, выполнение произвольной команды, которая определяется ее именем. Следует отметить что все эти встроенные команды применимы и к меню-барам (которые, по сути, являются обычным текстом в рамках с инвертированным цветом фона).

Команды мыши

Позиционирование текста

Для того чтобы изменить положение текста во вьювере, переместите мышь в зону прокрутки. Это вертикальная полоса вдоль левой границы шириной около 0,5 см. Вы можете прокручивать текст вперед, нажав ЛКМ, перемещая мышь и строка текста, которую вы хотите сделать верхней, будет подчеркнута. В каждой полосе прокрутки отображается небольшая горизонтальная метка, указывающая текущее положение участка в пределах всего текста. Вы можете позиционировать текст нажав СКМ в том месте, где вы хотите видеть эту метку. Если вы хотите, чтобы отображалось начало текста, вы можете просто щелкнуть правой кнопкой мыши в любом месте полосы прокрутки.

Размещение каретки

Если вы хотите поставить каретку, подведите мышь к нужному тексту, нажмите левую кнопку мыши и, не отпуская ее, переместите каретку в нужную позицию. Все последующие набранные символы будут вставлены в эту позицию.

Обратите внимание, что немецкие умлауты вводятся нажатием следующих комбинаций клавиш:

Ctrl a	для а,
Ctrl Shift	для А,
Ctrl o	для о,
Ctrl Shift o	для О,
Ctrl u	для u
Ctrl Shift u	для U.

	A O U
-	a o u
Shift	A O U
Ctrl	a o u
Shift Ctrl	A O U

Выделение текста

Вы можете выделить любой участок текста перемещением мыши с нажатой ПКМ. Двойной щелчок ПКМ (не обязательно быстрый) перед концом строки, выделяет всю строку. Тоже самое можно сделать в любом месте строки, тогда будет выделение с этого места и до левого края строки. Если текст слишком

большой и уходит вниз за пределы вьювера, используйте Edit.Сору для открытия соседнего второго вьювера. Затем отдельно выберите начало участка текста в верхнем вьювере и его конец в нижнем вьювере. Обратите внимание, что отдельное выделение может быть активным для каждого отображаемого участка текста, включая заголовки вьюверов. Если на экране одновременно существует несколько выделений, команды обращаются к самому последнему.

Существуют следующие интересные интеркликовые варианты размещения каретки и выделения текста, которые эффективно сочетают эти операции маркировки с редактированием текста. Помните правило, гласящее, что любая операция, которая в данный момент выполняется мышью, может быть отменена нажатием всех оставшихся клавиш мыши.

Копирование текста

Если вы при размещении каретки левой кнопкой мыши, сделаете интерклик СКМ, то последнее выделение скопируется в это место. Это удобно при копировании одного выделения в разные места. Аналогично, если вы уже поставили каретку в нужном месте, то нажатие СКМ при выделении текста ПКМ скопирует текст в позицию каретки как только вы отпустите ПКМ. Чтобы отменить любой интерклик, просто нажмите оставшуюся третью кнопку, удерживая основную.

Копирование атрибутов

Если вы делаете интерклик ПКМ во время установки каретки под символом ЛКМ, атрибуты этого символа (шрифт, цвет, вертикальное смещение) будут применены к последнему выделению, как только вы отпустите ЛКМ.

Удаление текста

Если вы сделаете интерклик ЛКМ во время выделения текста ПКМ то выделенный текст будет удален. Здесь также, интерклик может быть отменен дополнительным интеркликом СКМ.

Заметьте что выше описанные операции редактирования применяются к заголовку вьювера только ограниченно. Заголовок не может быть изменен. Также каретка не может быть помещена в заголовок. Далее обратите внимание, что вариант копирования и вариант удаления при выборе применяются также в случае больших выборок с разделенными вьюверами.

Активация команды

Активация именованной команды из вьювера является самой распространенной операцией. Для этого просто наведите курсор на имя команды и нажмите СКМ (или на колесо мыши). Иногда (например, на этапе тестирования) важно, чтобы перед выполнением команды была загружена самая новая версия модуля, обеспечивающего нужную команду. Чтобы сделать это, просто нажмите ЛКМ удерживая СКМ и указывая на имя команды.

Следующая таблица обобщает основное значение клавиш мыши.

Первичная клавиша ->	Левая	Средняя	Правая
Вторичная клавиша			

нет	установка каретки	выполнить	выбрать и...
левая	–	Выгрузить модуль и выполнить	...удалить
средняя	копировать выделение отменить удаление	–	...копировать
правая	копировать атрибуты	–	–

Вьювер отображают текст стандартным способом. В частности, он не поддерживает никакого форматирования, например, автоматического переноса строк или выравнивания абзацев по правому краю. Однако, цвет шрифта и вертикальное смещение, возможны. Мы рекомендуем использовать следующие шрифты: Syntax10.Scن.Fnt (по умолчанию), Syntax10i.Scن.Fnt (Italic), Syntax10b.Scن.Fnt (Bolid), и Courier8.Scن.Fnt (моношир).

Вьювер-ловушек TRAP

Если выполнение команды не удастся, система выдает вьювер-ловушку (Trap). В Oberon не существует низкоуровневого отладчика. Однако обработчик ловушек вызывается всякий раз, когда произошла ошибка выполнения, это и есть отладчик но только высокоуровневый. Он отображает состояние прерванного процесса, (стек активаций процедур) состояние переменных. Он также связывает местоположение ловушки с позицией в исходном тексте программы. Чтобы найти сбойное утверждение, просто отметьте вьювер с исходным текстом программы, "звездочкой" (F1). Затем выделите номер позиции во вьювере-ловушек и вызовите команду Edit.Locate в заголовке. Указатель переместится в место где обнаружена ошибка. Список номеров ошибок смотрите в конце этого руководства. Вьювер-ловушек можно вызвать пренудительно, нажав: Ctrl-Shift-Del

Дополнительную функциональность обеспечивает пакет инструментов для редактирования текста Text Edit Tool Package. Он содержит следующие команды:

Команды редактирования

Edit.Open

Открывает вьювер отображающий текст в пользовательском треке. Имя файла текста должно следовать за именем команды Edit.Open или если следует символ "стрелка вверх" то открывается последнее выделенное имя. Если такого имени не существует, используется имя по умолчанию. Чтобы отменить автоматическое распределение, поместите указатель в любое место экрана.

Edit.Show M.X

Открывает вьювер отображающий определенный объект X модуля M в пользовательском треке. Если реализация модуля M доступна, показывается реализация X, если нет отображается только список определенных процедур модуля (Definition).

Edit.Store

Сохраняет текст в файл в помеченном вьювере, имя можно задать как параметр этой команды или ввести в начале заголовка вьювера.

Edit.Recall

Вставляет (восстанавливает) последний удаленный участок текста в позицию каретки.

Edit.CopyFont

Пересылает шрифт из помеченной области в последнее выделение текста. Шрифт выделения меняется.

Edit.ChangeFont

Применяет шрифт заданный как параметр этой команды к последнему выделению текста.

Edit.ChangeColor

Применяет цвет заданный как параметр этой команды к последнему выделению текста.

Edit.ChangeOffset

Применяет вертикальное смещение заданное как параметр этой команды к последнему выделению текста.

Edit.Search

Ищет шаблон текста в отмеченном тексте. Шаблон определяется последним выделением текста. Если такого нет, используется предыдущий шаблон. Поиск начинается с позиции каретки. Если такого нет в отмеченном тексте, поиск начинается с начала текста.

Edit.Locate

Эта команда всегда находится в заголовке системного журнала (System Log) и используется для поиска ошибок возникших при компиляции модуля. Она устанавливает каретку в исходном тексте модуля, согласно выделенному номеру позиции в системном журнале. Компилятор выдает в журнал номер позиции и номер ошибки, например:

```
pos 123 err 40
```

Если выделить всю эту строку то все символы кроме номера позиции игнорируются. Поэтому, можно просто выделить всю строку двойным нажатием ПКМ и нажать СКМ на Edit.Locate в заголовке. Это очень частое действие при программировании и поэтому все сделано для удобства и скорости.

Edit.PrintServerName [%] {(ИмяФайла|"*")} ["/" NofCopies]}

Отправляет все тексты, указанные в списке параметров, на сервер печати, имя которого взято из первой записи в списке параметров. Имена в списке параметров относятся к текстовым файлам, символ "*" к тексту в отмеченном выхвосте. Символ "%" указывает на опцию vanilla-print. Если она активна, текст печатается одним моноширинным мелким шрифтом (Gacha10L). Эта опция обычно используется для печати листингов исходных программ. NofCopies опционально задает желаемое количество копий. Это должно быть однозначное число. Эта команда предполагает, что правильная идентификация пользователя была установлена ранее (вызовом System.SetUser).

Пакет инструментов рисования Draw Tool Package

Система Draw служит для подготовки линейных чертежей. Они содержат линии, текстовые подписи и другие элементы и отображаются в графическом вьювере. Графический вьювер показывает фрагмент плоскости чертежа, а несколько вьюверов могут показывать разные части чертежа.

Наиболее часто используемые команды встроены в виде щелчков мыши и их комбинаций. Дополнительные команды можно выбрать из текста, либо в меню вьювера, либо в тексте под названием Draw.Tool.

Кнопки мыши имеют следующие функции:

Левая:	рисовать
Средняя:	перемещение/копирование
Правая:	выбор

Команда мыши идентифицируется (1) по кнопке k0, нажатой первоначально, (2) по начальному положению P0 курсора, (3) набором кнопок k1, нажатых до отпускания последней, и положением курсора P1 в момент отпускания.

Базовая система состоит из модулей Draw, GraphicFrames и Graphics. Эти модули содержат средства для создания и обработки горизонтальных и вертикальных линий, текстовых подписей и макросов. Дополнительные модули служат для введения других элементов, таких как прямоугольники и круги. Система является расширяемой, т.е. могут быть введены дополнительные модули для работы с другими элементами.

Draw.Open

(доступна в Draw.Tool) открывает новый вьювер и отображает график с именем, заданным в качестве параметра. Если команда не задана, то в качестве параметра берется текстовое выделение. Рекомендуется, чтобы имена файлов использовали расширение Graph.

Чтобы нарисовать горизонтальную или вертикальную линию от P0 до P1, нажмите кнопку с курсором в точке P0 и, удерживая кнопку, переместите мышь и курсор на P1. Затем отпустите кнопку. Если P0 и P1 совпадают по координатам x и y, конечная точка корректируется таким образом, чтобы линия была горизонтальной или вертикальной.

Создание надписи

Сначала установите курсор в то место, где должна появиться надпись. Затем нажмите левую кнопку, в результате чего появится перекрестие называемое "катетом". Затем наберите текст. Принимаются только вводимые тексты. Клавиша DEL можно использовать для удаления символов (backspace).

Выбор

Большинство команд требуют указания операндов, и многие из них неявно предполагают, что ранее выбранные элементы – выбор – являются их операндами. Выбор отдельного элемента осуществляется путем наведения на него курсором и нажатием правой кнопки мыши. Это также приводит к тому, что ранее выделения будут сняты. Если необходимо сохранить выделение, нажмите левую кнопку мыши, это действие называется интеркликом. Если вы хотите выделить сразу несколько элементов, переместите курсор с P0 на P1, удерживая правую клавишу. Затем будут выбраны все элементы, находящиеся в прямоугольнике с диагонально противоположными сторонами в точках 0 и P1.

Выделенные линии

отображаются в виде пунктирных линий, выделенные титры (и макросы) – в виде инверсного режима. Макрос выбирается путем наведения курсора на его нижний левый край.

Перемещение

Если вы хотите переместить (сместить) набор элементов, сначала выделите их, а затем переместите курсор из положения P00 P1, удерживая среднюю кнопку. Вектор от P0 до P1 задает перемещение и называется вектор перемещения. P0 и P1 могут совпадать в разных программах просмотра, отображающих один и тот же график.

Копирование

Аналогично, выбранные элементы могут быть скопированы (продублированы) . при указании вектора перемещения, нажмите левую кнопку.

Команда копирования также может быть использована для копирования элементов из одного графика в другой график путем перемещения курсора из одного вьюера в другой, отображающий целевой график.

Текстовая надпись может быть скопирована из текстового фрейма в графический фрейм и наоборот. Существует два способа достичь этого:

1. Сначала поместите каретку в конечную позицию, затем выделите текст и нажмите среднюю кнопку мыши.
2. Сначала выделить текст для копирования, затем поместить каретку в конечную точку, нажав левую кнопку мыши и щелкнув среднюю кнопку.

Следующая таблица показывает основное значение клавиш мыши.

Первичная клавиша ->	Левая	Средняя	Правая
Вторичная клавиша			
нет	рисовать линию установка каретки	переместить	выбрать
левая	-	копировать	выбрать
средняя	копировать текст	-	копировать текст
правая	-	сдвиг плана	-

Команды меню

Следующие команды из пакета Draw отображаются в меню (title bar) каждого графического вьюера. Они активируются при наведении на них курсора и нажатии средней кнопки мыши.

Draw.Cleanup

Весь рисунок стирается.

Draw.Delete

Выбранные элементы удаляются.

Draw.Store

Рисунок записывается в файл с указанным именем.

Оригинальный файл переименовывается путем изменения расширения на Bak.

Последующие команды относятся к атрибутам элементов рисунка, таким как ширина линии, шрифт текста и цвет. Set-команды определяют соответствующие атрибуты последующих создаваемых элементов, Change-команды – атрибуты текущего выделения в помеченном средстве просмотра.

Draw.SetWidth	w	Ширина по умолчанию = 1, $0 < w < 7$.
System.SetFont	fontname	По умолчанию = Syntax10.Scn.Fnt
System.SetColor	c	По умолчанию = белый
Draw.ChangeWidth	w	($0 < w < 7$)
Draw.ChangeFont	fontname	
Draw.ChangeColor	c	

Команды SetColor и ChangeColor принимают либо номер цвета в диапазоне 1..15, либо строку в качестве параметра. В последнем случае цвет выбирается из символа, следующего сразу за кареткой (см. раздел Draw.Tool).

Макросы

Макрос – это маленький рисунок, который может быть идентифицирован как единое целое и использоваться как элемент внутри большего рисунка. Макросы обычно хранятся в коллекциях, называемых библиотеками, откуда их можно выбрать и копировать по отдельности.

Draw.Macro lib mac

макрос mac выбирается из библиотеки lib и вставляется в позицию каретки. Он автоматически выбирается и показывается в инверсном виде.

Draw.Directory libfile

перечисляет имена макросов, содержащихся в libfile.

Следующие команды используются, когда необходимо создать новые макросы и вставить их в (возможно, новую) библиотеку.

Draw.OpenMacro lib mac

элементы макроса mac из библиотеки lib вставляются в позицию каретки. Теперь их можно выбрать по отдельности.

Draw.MakeMacro lib mac

1. Выделите все элементы, которые должны принадлежать новому макросу.
2. Установите каретку в место начала макроса.
3. Поместите вторичную каретку на противоположный край макроса с помощью левой кнопки и нажатием правой кнопки.
4. Активизируйте команду MakeMacro. Макрос будет добавлен в библиотеку lib и получает имя mac.

Дополнительные команды

Следующие команды перечислены в тексте Draw.Tool, но могут появляться в любом тексте.

Draw.Reset

Плоскость чертежа в отмеченном выювере (x) сбрасывается так, что плоскость находится в левом нижнем углу.

Draw.Store name

Рисунок в отмеченном выювере сохраняется как файл с указанным именем.

Draw.Pint Serername *

Рисунок в отмеченном выювере печатается указанным сервером.

Draw.Print Severname (Filename)

Печатается именованный файл.

Draw.SetGrid n

Внутри графических выюверов курсор перемещается дискретными шагами по невидимой сетке. Расстояние между точками сетками изначально равно 4 пикселям; Этой командой можно установить расстояние в 2-n пикселей ($0 \leq n < 4$).

Draw.CarPos

Отображает координаты каретки в окне просмотра журнала.

Draw.Makelib lib fname

Библиотека lib записывается в файл. Важное замечание: Записанная коллекция макросов содержит только те макросы, которые были ранее вставлены командами Draw.Macro и Draw.MakeMacro, Следовательно, если новые макросы должны быть добавлены к файлу библиотеки, весь файл должен быть сначала прочитан командой Draw.Macro Применяется к каждому существующему макроэлементу, это делается путем открытия рисунка, содержащего все макросы из библиотеки. Макросы обычно используются для рисования электронных схем. Основной файл библиотеки, содержащий часто используемые TTL-схемы, называется TTL0.lib, а чертеж, показывающий его элементы, называется TTL0.Graph.

Прямоугольники

Прямоугольники могут быть созданы как отдельные элементы и часто используются в качестве рамок вокруг набора других элементов. Они состоят из четырех линий, которые можно выбрать как единое целое. Команды атрибутов Setict, SetColor, ChangeWideh и ChangeColor также применяются к прямоугольникам. Прямоугольники выбираются путем указания на их нижний левый угол.

Rectangles.New

1. Поместите каретку туда, где находится край нового прямоугольника (левая кнопка).
2. Поместите вторую каретку в противоположную точку, нажав правую кнопку.
3. Активизируйте команду.

Прямоугольники могут быть заполнены тенью. Оттенок задается в виде числа (0#s#9).

`Rectangles.SetShades` по умолчанию = 0: без затенения.

Примечание: В текущей реализации все оттенки отображаются как один и тот же рисунок на дисплее, но различаются на принтере.

Круги

Дополнительные графические элементы – круги и эллипсы.

Circles.New

1. Установите каретку в том месте, где должен находиться центр.
2. Поместите вторую каретку справа от центра. Эта позиция будет лежать на окружности.
3. Активируйте команду.

Если над центром поместить еще одну вторичную каретку, вместо окружности будет создан эллипс.

Установка системы Draw

Для установки системы необходимы следующие модули:

`Draw.Obj GraphicFrames.Obj Graphics.Obj Draw.Tool`

Если необходимо использовать прямоугольники

`Rectangles.Obj`

и если будут использоваться круги

`Circles.Obj Display1.Obj`

Для рисования цифровых электрических схем необходимы следующие шрифты и библиотеки, также Прямоугольники
`Syntax8.Scen.Fnt Elektra.Scen.Fnt TTL0.Lib`

Кроме того, существуют следующие файлы: `Symbols.Graph TTL0.Graph`

Пакет инструментов Paint

Пакет `Paint` служит для отображения и редактирования оцифрованных

изображений. Его схема функциональности и пользовательский интерфейс максимально точно повторяют линии текстовых и графических пакетов.

Встроенные команды, управляемые мышью, следующие:

Первичная клавиша ->	Левая	Средняя	Правая
Вторичная клавиша			
нет	установка каретки	rep/выбрать переместить	выбрать
левая	-	rep/рисовать	удалить
средняя	копировать выбор	-	копировать текст
правая	многократная каретка	rep/очистить	-

Сопутствующий пакет инструментов дополняет набор встроенных операций редактирования изображений следующими командами.

Paint.Open {/w h}

открывает в пользовательском треке вьювер, отображающий указанный рисунок. В качестве альтернативы картинка задается параметром в командной строке или последним выбором имени. Если такого параметра не существует, то имя принимается по умолчанию.

Paint.Zoom

увеличивает изображение до такого размера, чтобы последние выборы были видны полностью, - если выбора нет, увеличивает изображение.

Paint.Print

печатает именованный файл. Предполагается, что ранее была установлена правильная идентификация пользователя.

Paint.Scale

открывает копию текущего изображения, масштабированную до размера последнего выделения, или уменьшенную до размера реального изображения.

Paint.Drow

нарисовать вертикальную/горизонтальную линию рисует вертикальную/горизонтальную линию, начинающуюся на первой каретке и заканчивающуюся на второй.

Paint.SetColor | белый | черный | белый инверт

применяет цвет, указанный параметром, к последующим выполняемым операциям.

Paint.SetPen w h

устанавливает ширину и высоту пера tow и h,

Paint.Fill n

закрашивает область с заданным рисунком.

Paint.SetGrid n

устанавливает n-расстояние между точками сетки. Каретка перемещается дискретными шагами по заданной сетке.

Paint.SetFont FontName

применяет шрифт, указанный параметром, к последующим набранным символам.

Последующие разделы завершают эту главу. В них дается краткое функциональное описание наиболее важных из доступных в настоящее время инструментов, не связанных с редактированием.

Пакет системных инструментов System Tool Package

Этот пакет содержит команды, связанные с системой. Среди них есть процедуры для отображения инструментов в виде вьюверов, закрытия вьюверов и фокусов, определения и отображения всех видов системных параметров, а также предоставлять базовые утилиты для работы с именованными файлами. Кроме того, пакет инструментов System Tool содержит обработчик ловушек, который неявно вызывается после возникновения ловушки отображая текущее состояние системного стека.

System.Open

открывает в системном треке вьювер, отображающий указанный инструмент. Инструмент альтернативно указывается как параметр или, в случае знака "^" (стрелка вверх) после имени команды - последним выбором имени. Если такого имени не существует, берется имя по умолчанию.

System.OpenLog

открывает вьювер в системном треке, отображающий системный журнал. Обратите внимание, что журнал обновляется независимо от его видимости. Поэтому в журнале всегда отображается полная история текущего сеанса.

System.Copy

открывает копию исходного окна просмотра, отображая тот же самый экземпляр содержимого.

System.Grow

позволяет увеличить вьювер до размера целой дорожки или, если применяется к вьюверу, уже заполняющему трек, до размера всего дисплея. Не факт, что оригинальная группировка будет восстановлена, когда увеличенный вьювер будет удален.

System.Close

удаляет отмеченный вьювер с дисплея или, если вызывается из строки меню текстового вьювера, восстанавливает собственный вьювер.

System.CloseTrack

закрывает отмеченный трек, т.е. удаляет все вьюверы на этом треке.

System.Recall

снова открывает недавно (возможно, ошибочно) закрытый вьювер.

System.Time

отображает текущую дату и время. Если параметры даты и времени дд мм. и чч.мм. (день, месяц, год и час, минута, секунда) непосредственно следуют за именем команды, System.Time сначала устанавливает дату и время соответственно.

System.Watch

отображает объем используемого в данный момент дискового пространства и ресурсов памяти.

System.Collect

инициирует последующую сборку мусора.

System.Free

выгружает каждый модуль, указанный параметром. Если за именем модуля сразу следует "^", импортированные модули также выгружаются.

System.ShowModules

отображает карту всех загруженных в данный момент модулей.

System.ShowCommands ModName

отображает список всех команд, экспортируемых данным модулем.

System.State ModName

отображает глобальные данные указанного модуля.

System.SetUser

принимает идентификацию пользователя в виде UserName "/" Password без отображения на дисплее. Имя пользователя имеет длину до восьми символов (в большинстве случаев инициалы). Пароль – произвольная строка.

System.Directory

отображает набор всех дисковых файлов, имя которых совпадает с шаблоном, заданным параметром. Параметр может содержать символ "*" в качестве маски. Если указан параметр 'd' (/d сразу после параметра) выводится дополнительная информация о размерах и датах файлов. При "^" после имени команды параметр берется из текущего выбора.

System.CopyFiles

обрабатывает список параметров пар A => B. Копирует каждый файл A в файл B. В случае "^", следующего за командой параметр берется из текущей выборки.

System.RenameFiles

обрабатывает список параметров пар A => B. Переименовывает каждый файл A в файл B. В случае "^" после имени команды параметр берется из текущей выборки.

System.DeleteFiles

удаляет каждый файл, указанный параметром. В случае "^", следующего за именем команды, параметр (имя одного файла) берется из текущей выборки.

System.SetFont

применяет шрифт, указанный параметром, к последующим набранным символам.

System.SetColor

применяет цвет, указанный параметром, к последующим набранным символам.

System.SetOffset

применяет вертикальное смещение, заданное параметром, к последующим набранным символам.

Пакет инструментов компилятора Compiler Tool Package

Этот пакет содержит компилятор Oberon. Результат компиляции показан в журнале. Номера ошибок отображаются в журнале вместе с номерами позиций в исходном тексте. Для того чтобы найти ошибку в исходном тексте, воспользуйтесь командой Edit.Locate в меню журнала, выделив перед этим номер позиции. Инструмент компилятора экспортирует одну команду:

Compiler.Compile

она компилирует все тексты, указанные в списке параметров, имена в списке параметров ссылаются на текстовые файлы, символ "*" обозначает исходный текст отмеченный "звездой". В случае, если после имени команды стоит символ "^", параметр берется из текущего выделения. Доступны следующие параметры: "/x" (отключение проверки индекса), "/v" (проверка целочисленного переполнения), "t*" (отключение защиты типов), "/s" (разрешить изменение символического файла) и "/d" (обеспечить отладку).

Пакет различных инструментов Miscellaneous Tool Package

Этот пакет предоставляет утилиты для преобразования файлов, определения статистических данных и демонстрации поведения системы.

Miscellaneous.BoatLoad FileName

загружает именованный загрузочный файл в загрузочные сектора на диске.

Miscellaneous.ConvertBlanks

преобразует все текстовые файлы, указанные в списке параметров, заменяя пары ведущих пробелов символами табуляции. Символ "^" после имени команды относится к текущему выделению (имя одного файла).

Miscellaneous.ConvertTabs

преобразует все текстовые файлы, указанные в списке параметров, заменяя ведущие символы табуляции парами пробелов. Символ "^" после имени команды относится к текущему выделению (имя одного файла).

Miscellaneous.Cleanup

преобразует все текстовые файлы, указанные в списке параметров, в обычные файлы Ascii, т.е. удаляет информацию о форматировании и непечатаемые символы. Символ "^" после имени команды относится к текущему выделению (имя одного файла).

Miscellaneous.CountLines

подсчитывает строки текстового файла, указанного в списке

параметров. Символ "^" после имени команды относится к текущему выделению (имя одного файла).

Miscellaneous.GetObjSize

извлекает размер кода и размер данных из всех объектных файлов, указанных в списке параметров. Символ "^" после имени команды относится к текущему выделению (имя одного файла).

Miscellaneous.Snapshot

делает снимок текущего состояния экрана и сохраняет его в двух файлах, указанных в списке параметров. Первый файл содержит нижнюю половину, а второй – верхнюю половину экрана.

Miscellaneous.Trap

вызывает ловушку, демонстрируя тем самым механизм отображения ловушек.

Пакет инструментов резервного копирования Backup Tool Package

Этот пакет обрабатывает передачу данных между основным диском и дискетой. Он поддерживает дискеты, соответствующие двустороннему формату DOS-подобный.

Backup.Format

форматирует произвольную двухстороннюю дискету. Эта команда защищена. Удалите символ "!" перед выполнением этой команды.

Backup.Init

инициализирует отформатированную дискету в MSDOS-подобный формат Oberon. Эта команда защищена. Удалите символ "!" перед выполнением этой команды.

Backup.Directory

отображает каталог загруженной в данный момент дискеты.

Backup.DeleteFiles

удаляет все файлы, указанные в списке параметров, с текущей загруженной дискеты.

Backup.ReadAll

считывает все файлы с текущей загруженной дискеты.

Backup.ReadFiles

считывает файл, указанные параметром файлы, с текущей загруженной дискеты.

Backup.WriteFiles

записывает все файлы, указанные параметром list, на загруженную в данный момент дискету.

Backup.ConvertToMSDOS

конвертирует текущую загруженную дискету в официальный DOS-формат P9 for 3.5 inch Diskette. DOS-ограничения для имен файлов (длина имени файла <=8, длина расширения <=3, только символы верхнего регистра).

Backup.ConvertFormMSDOS

преобразует текущую загруженную дискету из официального DOS-формата P9 для 3,5-дюймовых дискет в Oberon-формат.

Пакет инструментов Net Server

Этот пакет поддерживает электронную почту и передачу файлов по локальной сети. Он также позволяет рабочей станции стать файловым сервером. Предполагается, что правильная идентификация пользователя была установлена ранее.

Net.Mailbox

открывает вьювер Mailbox.Text, представляющий обзор всех сообщений, находящихся в данный момент в почтовом ящике.

Net.DeleteMail

появляется в заголовке вьювера Mailbox.Text почтового ящика и удаляет сообщение, запись которого выбрана.

Net.Receivetail

появляется в заголовке вьювера Mailbox.Text почтового ящика, открывает окно просмотра сообщения, запись которого выбрана.

NetSendMail

отправляет сообщение, отображаемое в отмеченном вьювере. Синтаксическое определение сообщения выглядит следующим образом:

сообщение	= получатель {получатель} [тема] текст
получатель	= "To" Список адресов "cc" Список адресов
список адресов	= адрес {" , " адрес}
тема	= "Re" текст

Адрес – это RFC-имя (@-нотация), идентифицирующее отдельного получателя или имя группы получателей, которое хранится на почтовом сервере. Вместо @ также принимаются ! и %. Никакая @-часть не нужна, если получатель находится в сети Ceres.

NetSendFiles

отправляет последовательность файлов на удаленный файловый сервер. Первое имя в списке параметров идентифицирует желаемый файловый сервер, а остальные имена определяют последовательность желаемых файлов. Путем префиксации имен файлы могут быть отправлены в определенные подкаталоги. Например, спецификация R.F.X подразумевает автоматическое преобразование имен с FX (на главном компьютере) на RX (на сервере). Если непосредственно следует за именем сервера, то параметр берется из текущего выделения.

Net.ReceiveFiles

получает последовательность файлов с удаленного файлового сервера. Первое имя в списке параметров идентифицирует желаемый файловый сервер, а остальные имена определяют последовательность желаемых файлов. Опять же, префикс и автоматическое преобразование имен позволяет получать файлы из определенных подкаталогов. Например, спецификация RX подразумевает автоматическое преобразование имен из REX (на сервере) в FX (на сервере) ведущем устройстве. Если символ "^" следует сразу за именем сервера, то список параметров берется из текущего выделения.

Net.DeleteFiles

удаляет список файлов с удаленного файлового сервера. Первой записью в списке параметров является имя сервера.

Net.SendMsg <сообщение партнера>

отправляет однострочное сообщение указанному партнеру.

Net.GetTime

получает время с сервера и настраивает часы на локальном компьютере.

Net.SetPassword server <пароль>

устанавливает новый пароль на сервере.

Net.StartServer

переводит рабочую станцию в режим сервера, т.е. разрешает доступ с других станций в сети.

Net.StopServer

выключает режим сервера.

Net.Unprotect

разрешает доступ на запись с других станций.

Ne.WProtect

запрещает доступ к записи с других станций (режим по умолчанию).

Пакет инструментов ColorSystem

Этот пакет предоставляет утилиты для проверки и определения параметров цветного дисплея. Его следует использовать только в том случае, если цветной дисплей физически установлен.

ColorSystem.Init

инициализирует экран цветного дисплея.

ColorSystem.Colors

инициализирует цветовую палитру.

ColorSystem.ShowColors

открывает вьювер, отображающий цветовую палитру. Вьювер, можно редактировать.

ColorSystem.LoadColors <Имя>

загружает указанную цветовую палитру.

ColorSystem.StoreColors Имя

сохраняет текущую палитру под указанным именем.

Colorsystem.SetCursor x

устанавливает цветовой курсор в режим x.

x = "^" только стрелка.

x = "+" только перекрестие.

x = "*" стрелка и перекрестие.

Иерархия модулей Oberon

В модульной иерархии Oberon мы различаем следующие структурные единицы: внутреннее ядро, внешнее ядро, текстовое ядро, графическая система, система рисунков, и коллекция инструментов.

Пакеты

инструментов

Net	Backup	Compiler	System	Miscellaneous	ColorSystem	
			Edit		Draw	Paint
			TextSystem		GraphicSystem	PictureSystem
			TextFrames		GraphicFrames	PictureFrames
					Graphic	Pictures
					MenuViewers	
			Внешнее ядро			
Внутреннее ядро		Принтер	Oberon			
	Модули			Тексты		
	Файлы			Шрифты		
	FileDir		Math	MathL	Reals	Viewers
Драйверы	Kenel	V24	SCC	Diskette	Input	Display

В обязанности внутреннего ядра входит управление памятью, файлами и программами. Внешнее ядро обеспечивает управление устройствами для сетевых портов, клавиатуры, мыши и дисплея. Другими частями внешнего ядра являются менеджер просмотра, управление текстом и поддержка удаленной печати. Модуль Oberon представляет собой основной интерфейс между внешним ядром и его клиентами. Он включает разделы, посвященные текущей конфигурации системы, стратегиям по умолчанию для треков распределения и размещения вьюверов, а также поддержке выполнения команд.

Модуль дисплея находится в самом низу иерархии системы отображения. Область отображения рассматривается как плоскость с координатами x и y . Она включает в себя как черно-белую, так и цветную области. Растровые операции используются для создания и копирования прямоугольных областей на плоскости дисплея. Участки плоскости можно сделать видимыми с помощью процедур управления отображением (DisplayControl). Видимые части плоскости дисплея структурированы как дорожки и вьюеры, и ими управляет менеджер вьюверов (Viewers). Модуль Oberon определяет стандартную компоновку с одним пользовательским треком и одним системным треком на экране дисплея. Наконец, модуль MenuViewers представляет собой высокоуровневый менеджер просмотра для стандартных вьюверов, состоящих из заголовка и основной области окруженной тонкой рамкой. И строка заголовка, и основная область являются так называемыми рамками (frames). В то время как строка заголовка почти всегда является текстовой рамкой (см. следующий параграф), тип основной рамки зависит от типа вьювера.

Текстовая система, графическая система и система изображений идентичны по структуре. Каждая из них состоит из тройки линейно зависимых модулей. В случае с текстами они называются Texts, TextFrames и Edit. Texts определяет объектный тип Text и экспортирует внутренние операции над текстами. TextFrames определяет тип объекта TextFrames.Frame и работает с

представлениями текстов в подфреймах вьюверов.

Модули в верхней части (например, Edit) – это пакеты инструментов. Пакет инструментов просто экспортирует набор команд в виде процедур без параметров. Инструментальные модули интенсивно используют средства, предоставляемые модулями более низкого уровня, в частности, системой просмотра, текстовой системой и модулем центральной системы Oberon. Важно, чтобы обычные команды строго оперировали текстами или графикой, а не клавиатурой или экраном напрямую.

Мы понимаем эту главу как учебник по реализации пакетов инструментов. Сначала мы дадим комментированный обзор определений наиболее важных модулей нижнего уровня. Затем мы проиллюстрируем их использование на примере некоторых типичных выдержек из существующих инструментов.

Определения модулей

Система отображения Display System

DEFINITION Display; (* драйвер дисплея *)

CONST black = 0; white = 15;

replace = 0; paint = 1; invert = 2; (*режимы работы*)

TYPE

Frame = **POINTER TO** FrameDesc;

FrameNhsg = **RECORD END**; (*базовый тип сообщений для фреймов*)

Pattern = LONGINT; (*указатель на дескрипторы шаблонов*)

(*PatternDesc = **RECORD**

w, h: SHORTINT;

raster: **ARRAY** (w + 7) DIV 8 x h **OF** BYTE

END*)

Font = **POINTER TO** Bytes;

Bytes = **RECORD END**;

Handler = **PROCEDURE** (Frame, **VAR** FrameMsg);

FrameDesc = **RECORD** (*базовый тип фреймов*)

dsc, next: Frame;

X, Y, W, H: INTEGER;

handl: Handler

END;

VAR

Unit: LONGINT; (*RasterUnit = Unit/36000 mm*)
left, (*left margin of black-and-white maps*)
Colleft, (*left margin of color maps*)
Bottom, (*Bottom of primary map*)
UBottom, (*Bottom of secondary map*)
Width, (*map width*)
Height: (*map height*)
INTEGER;

arrow, star, cross, downArrow, hook: Pattern;

PROCEDURE Map (X: INTEGER): LONGINT; (*address of map at X*)

PROCEDURE SetMode (x: INTEGER; 5: SET); (*set mode of map at X*)

(*black & white display: 0: display disable, 1: display secondary map, 2:
inverse video*)

(*цветной дисплей*)

PROCEDURE SetColor (col, red, green, blue: INTEGER); (*col < 0: overlay color*)

PROCEDURE GetColor (col: INTEGER; **VAR** red, geen, blue: INTEGER);

PROCEDURE SetCursor(mode: SET); (*color cursor; 0: crosshair, 1: arrows*)

PROCEDURE InitCC: (*initialize color crosshair to full screen*)

PROCEDURE InitCP: (*инициализировать цветовой шаблон для формы стрелки*)

PROCEDURE DefCC (X, Y, W, H: INTEGER); (*определить окно для цветового
перекрестия*)

PROCEDURE DefCP (**VAR** raster: **ARRAY OF** BYTE); (*определите растр 64 x 64 для
цветового маркера шаблона*)

PROCEDURE DrawCX (X, Y: INTEGER); (*нарисовать цветной курсор в точке X, Y*)

PROCEDURE FadeCX (X, Y: INTEGER); (*цвет курсора исчезает при X, Y*)

(*шрифты*)

PROCEDURE GetChar(f: Font; ch: CHAR; **VAR** dx, x, y, w, h: INTEGER; **VAR** p:
Pattern);

(*получаем поле x, y, w, h, ширину dx и растровые данные p символа ch в font f*)

(*операции с растром*)

PROCEDURE CopyBlock (SX, SY, W, H, DX, DY, mode: INTEGER);

(*копирует исходный блок SX, SY, W, H в пункт назначения DX, DY, используя
режим работы

блок задается его нижним левым углом X, Y и его размером W, H*)

PROCEDURE CopyPattern (col: INTEGER; pat: Pattern; X, Y, mode: INTEGER);

(*копирует шаблон p в цвете col в X, Y, используя режим работы col = 0:
черный; col = 15: белый*)

PROCEDURE RepIPattem (col: INTEGER; pat: Pattern; X, Y, W, H, mode: INTEGER);

(*копирует шаблон p в цвете col в блок X, Y, W, H, используя режим работы,

двигаясь слева направо и снизу вверх, начиная с левого нижнего угла*)

PROCEDURE Repl**CONST** (col: INTEGER; X, Y, W, H, mode: INTEGER);

(*поместить "единицы" цвета col в блок X, Y, W, H, используя режим работы*)

END Display.

Замечания:

1. Компьютер Ceres оснащен монохромным дисплеем, положение которого (левый нижний угол) задается переменными Left и Bottom, а ширина и высота задаются переменными Width и Height. На самом деле область рисования больше; ее координата y колеблется от -1248 до 799. Два раздела могут быть сделаны видимыми с помощью процедуры управления отображением, первая из которых характеризуется {yl -1024 <= y < -224}, а другая - fyl <= y < 800}.

2. Если установлен цветной дисплей, растровые процедуры модуля можно использовать для создания и копирования областей на цветном экране. Положение цветовой области (нижний левый угол) задается переменными Colleft и Bottom (Нижняя часть); ее ширина и высота такие же, как у монохромного дисплея.

3. Постулируемые предусловия для параметров процедуры не проверяются модулем; это возлагается на вызывающие модули, которые несут ответственность за устойчивость.

4. Обратите внимание, что существуют следующие ограничения на реализацию растровых операций:

ReplConst

Отображение цвета: режим закраски рассматривается как режим замены, из этого модуля

RepIPattern

Ширина шаблона w игнорируется и принимается равной 32 при монохромном и 16 при цветном отображении. $0 \leq h < 256$ в монохромном режиме, $0 \leq h < 16$ в цветном. Цветное отображение: x и x+w должны быть четными, иначе вычитается 1.

CopyPattern

Режим замены рассматривается как режим рисования.

$0 < w < -32, 0 < -h < 256$.

CopyCopy

Все режимы рассматриваются как режим замены.

DEFINITION Viewers; (* менеджер просмотра *)

IMPORT Display;

CONST

restore = 0; modify = 1 ; suspend = 2;

(*идентификаторы сообщений, относящиеся к следующему типу сообщений*)

TYPE

Message - **RECORD** (*сообщение, посылаемое зрителям по событиям зрителя*)

(Display.FrameMsg)

id: INTEGER;

X, Y, W, H: INTEGER;

state: INTEGER

END;

Viewer = **POINTER TO** ViewerDesc;

ViewerDesc = **RECORD** (*дескриптор зрителя расширяет Display.FrameDesc*)

(Display.FrameDesc)

state: INTEGER

END;

```
(*state > 1: displayed  
state=1: filler  
state=0: closed  
state<0: suspended*)
```

VAR curW, minH: INTEGER; (*текущая ширина логического дисплея, минимальные
высоты просмотра*)

PROCEDURE InitTrack (W, H: INTEGER; Filler: Viewer);

 (*добавление к текущему логическому дисплею и инициализация трека шириной W
и высотой H и установка Filler*)

PROCEDURE OpenTrack (X, W: INTEGER; Filler: Viewer);

 (*открыть новую дорожку, перекрывающую область [X, X + W]*)

PROCEDURE CloseTrack (X: INTEGER);

 (*закрывает стойку в X и восстанавливает наложенные дорожки*)

PROCEDURE Locate (X, H: INTEGER; **VAR** fil, bot, alt, max: Display.Frame);

 (*на дорожке в точке X найти следующие зрители:
filler fil,
bottom viewer bot,
an alternative viewer alt с высотой >= H,
зритель max максимальной высоты*)

PROCEDURE Open (V: Viewer; X, Y: INTEGER);

 (*откройте новый вид \ с вершиной в Y на пути в X*)

PROCEDURE Change (V: Viewer; Y: INTEGER);

 (*расширение или сужение окна просмотра V до новой вершины Y*)

PROCEDURE Close (V: Viewer);

 (*удалить средство просмотра с экрана*)

PROCEDURE Recall (**VAR** V: Viewer);

 (*вызов последних закрытых программ просмотра*)

PROCEDURE This (X,Y: INTEGER): Viewer;

 (*возвращает программу просмотра по адресу X, Y*)

PROCEDURE Next (V: Viewer): Viewer;

 (*возвращает следующего верхнего соседа V*)

PROCEDURE Broadcast (**VAR** M: Display.FrameMsg);

 (*отправить сообщение M всем видимым зрителям*)

END Viewers.

DEFINITION MenuViewers;

IMPORT Display, Viewers;

CONST extend = 0; reduce = 1; (*сообщение lds*)

TYPE

Viewer = **POINTER TO** ViewerDesc;

ViewerDesc = **RECORD** (Viewers.ViewerDesc)

 menuH: INTEGER (*количество кадров меню*)

END;

ModifyMsg = **RECORD** (Display.FrameMsg)

id: INTEGER; (*расширяет или уменьшает*)

dY, Y, H: INTEGER (*вектор трансляции dY; новые Y и H*)

END;

VAR Ancestor: Viewer; (*просмотр текущего меню*)

PROCEDURE Handle (V: Display.Frame; **VAR** M: Display.FrameMsg);

(*стандартный обработчик для вьюверов меню*)

PROCEDURE New (Menu, Main: Display.Frame; menu, X, Y: INTEGER): Viewer;

(*создать и открыть по адресу X, Y новое средство просмотра меню, содержащее кадры Menu и Main*)

END MenuViewers.

Замечание:

Сообщения вьюверам меню, не влияющие на размер и положение, передаются в подфреймы. Предок становится доступным для обработчиков подфреймов через переменную Ancestor. MenuViewers также создает новые сообщения типа ModifyMsg, запрашивающие подфреймы об изменении размера или вертикального положения (или обоих). dY представляет собой вектор вертикального перевода, а Y и H задают новое положение и высоту соответственно.

Текстовая система

DEFINITION Fonts; (* загрузчик шрифта *)

IMPORT Display;

TYPE

Name = **ARRAY 32 OF** CHAR;

Font = **POINTER TO** FontDesc;

FontDesc = **RECORD**

name: Name; (*имя файла*)

height, minX, maxX, minY, maxY: INTEGER; (*данные символа*)

raster: Display.Font (*растровые данные*)

END;

(*height = минимальное расстояние между строками текста, minX, maxX, minY, maxY - минимумы и максимумы X и Y, если все поля для символов шрифта расположены в начале координат 0, 0*)

VAR Default: Font; (*шрифт по умолчанию*)

PROCEDURE This (name: **ARRAY OF** CHAR): Font; (*шрифт с заданным именем*)

END Fonts.

DEFINITION Texts; (* менеджер текстов *)

IMPORT Files, Fonts;

CONST

(*классы символов, см. определение типа Scanner*)
Inval =0; (*недопустимые символы*)
Name=1; (*имя s (длина len)*)
String=2 (*литерная строка s (длина len)*)
int=3; (*integer i (десятичная или шестнадцатеричная)*)
Real=4; (*real число x*)
LongReal=5; (*long вещественное число y*)
Char=6; (*специальный символ с*)
replace = 0; insert = 1; delete = 2; (*оп-коды*)

TYPE

Text = **POINTER TO** TextDesc;
Notifier = **PROCEDURE** (T: Text; op: INTEGER; beg, end: LONGINT);
TextDesc = **RECORD**
len: LONGINT; (*длина текста*)
notify: Notifier (*операций редактирования*)
END;

Reader = **RECORD** (Files.Rider)
eot: BOOLEAN;
fnt: Fonts.Font; (*шрифт текущего символа*)
col: SHORTINT; (*цвет текущих символов*)
voff: SHORTINT (*вертикальное смещение*)
END;

Scanner = **RECORD** (Reader)
nextCh: CHAR;
line: INTEGER;
class: INTEGER;
i: LONGINT;
x: REAL;
y: LONGREAL;
c: CHAR;
len: SHORTINT;
s: **ARRAY** 32 **OF** CHAR
END;

(*используется для преобразования текста в поток символов.
Классы символов определены в разделе **CONST***)

Buffer = **POINTER TO** BufDesc;

BufDesc = **RECORD**
len: LONGINT (*длина буфера*)

END;

(*используется для записи потока текстовых данных в буфер*)

(*используется для хранения фрагмента текста*)

Writer = **RECORD**(Files.Reader)

buf: Буфер; (*ассоциированные буферы*)

fnt: Fonts.Font; (*текущий font*)

col: SHORTINT; (*цвет текущих символов*)

voff: SHORTINT (*вертикальное смещение*)

END;

PROCEDURE Load (T: Txt; f: Files.File; pos: LONGINT; **VAR** len: LONGINT);

(*загрузка текстового блока из позиции pos в тексте T*)

PROCEDURE Open (T: Text; name: **ARRAY OF** CHAR);

(* открыть текст T из дискового файла, указанного именем; открыть новый текст, если имя = "" *)

PROCEDURE OpenBuf (8: Buffer);

(*открыть новый текстовый буфер B*)

PROCEDURE OpenReader (**VAR** R: Reader; : Text; pos: LONGINT);

(*открыть текстовый ридер R и установить его в позицию pos в тексте T*)

PROCEDURE Read (**VAR** R: Reader; **VAR** ch: CHAR);

(*прочитать следующий символ в ch*)

PROCEDURE Pos (**VAR** R: Reader): LONGINT;

(*положение читателя в тексте*)

PROCEDURE Store (: Text f: Files. File; pos: LONGINT; **VAR** len: LONGINT);

(*сохранить текст T в дисковый файл f в позиции pos*)

PROCEDURE Save (T: Txt; beg, end: LONGINT;

(*добавление отрезка [beg, end] текста T в буфер B*)

PROCEDURE Copy (SB, DB: Buffer);

(*добавляет копию исходного буфера SB в буфер назначения DB*)

PROCEDURE CharigeLooks (T: Text; beg, end: LONGINT; sel: SET; fnt: Fonts.Font; col, voff; SHORTINT);

(* изменение атрибутов символов в пределах отрезка [beg, end] текста T. sel выбирает атрибуты для изменения. 0,1,2 IN sel = fnt, col, voff выбраны *)

PROCEDURE Insert (T: Text; pos: LONGINT, B: Buffer);

(*вставка буфера B в текст T в позиции pos*)

```

PROCEDURE Append (T: Text; B: Buffer);
    (*добавление буфера B в текст T*)

PROCEDURE Delete (T: Text; beg, end: LONGINT);
    (*удалить участок [beg, end] текста T*)

PROCEDURE Recall (VAR B: Buffer);
    (*повторное удаление ранее удаленного текста*)

PROCEDURE OpenScanner (VAR S: Scanner; T: Text; pos: LONGINT);
    (*открыть текстовый сканер S и установить его в позицию pos в тексте T*)

PROCEDURE Scan (VAR S: Scanner);
    (*считывание следующего символа*)

PROCEDURE OpenWiiter (VAR W: Writer);
    (*открыть новый писатель W*)

PROCEDURE Setfont (VAR W: Wiiter; fnt: Fonts.Font);
    (*установить писатель W в шрифт fnt*)

PROCEDURE SetColor (VAR W: Writer; col: SHORTINT);
    (*установить для писателя W цвет col*)

PROCEDURE SetOffset (VAR W: Writer; voff: SHORTINT);
    (*установка писателя W на вертикальное смещение voff*)

PROCEDURE Write (VAR W: Writer; ch CHAR);
    (*запись символа ch в буфер W*)

PROCEDURE WriteLn (VAR W: Writer);
    (*запись конца строки в буферы W*)

PROCEDURE WriteInt (VAR W: Writer; x, n: LONGINT);
    (*запись целого числа x в буфер W. Правая корректировка на n позиций*)

PROCEDURE WriteHex (VAR W: Writer, x: LONGINT);
    (*запись целого числа x в буфер W в шестнадцатеричной форме*)

PROCEDURE WriteString (VAR W: Writer; s: ARRAY OF CHAR);
    (*запись строки в буфер W*)

PROCEDURE WriteReal (VAR W: Writer; x: REAL n: INTEGER);
    (*запись вещественного числа x в буфер W. Используется n позиций*)

PROCEDURE WriteRealFix (VAR W: Writer; x: REAL; n, k: INTEGER);
    (*запись вещественного числа x в буфер W в форме с фиксированной точкой,

```

используя k позиций для десятичных дробей и n позиций для итогов*)

PROCEDURE WriteRealHex (**VAR** W: Writer; x: REAL);

(*передает вещественное число x в буфер W в шестнадцатеричной форме*)

PROCEDURE WriteLongReal (**VAR** W: Writer; x: LONGREAL; n: INTEGER);

(*запись длинного действительного числа x в буфер Ws. Используется n позиций*)

PROCEDURE WriteLongRealHex (**VAR** W: Writer; x: LONGREAL);

(*запись вещественного числа x в буфер W в шестнадцатеричной форме*)

END Texts,

Замечание:

Орен не создает текстовый объект и не устанавливает процедуру нотификатора. Оба действия оставлены на усмотрение вызывающего модуль. Как правило, вызывающий модуль сначала создает текстовый объект или его расширение с помощью **NEW**, а затем устанавливает процедуру-нотификатор. Основное назначение процедур нотификатора – запрос дисплея на восстановление согласованности после изменения текста.

DEFINITION TextFrames; (* отображение текста *)

IMPORT Display, Texts;

TYPE

Location = **RECORD**

org, pos: LONGINT, (*положение начала строки*)

dx, x, y: INTEGER (*ширина и положение расположенного символа*)

END;

Frame = **POINTER TO** FrameDesc;

FrameDesc = **RECORD** (Display.FrameDesc)

text: Texts.Text; (*отображает текст*)

org: LONGINT; (*позиция в тексте первого отображаемого символа*)

col: INTEGER; (*цвет фона*)

Isp, asr, dsr: INTEGER; (*межстрочный интервал, ascender, descender*)

left, right, top, bot: INTEGER; (*поля*)

markH: INTEGER; (*ширина полей, положение полей*)

time: LONGINT; (*время последнего выбора*)

mark, car, sel: INTEGER; (*состояние метки, каретки, выбора*)

carloc: Location; (*местоположение каретки*)

selbeg, selend: Location (*расположение начала и конца выделения*)

END;

(* mark < 0: метка стрелки

mark = 0: нет метки

mark > 0: позиция метки

```
car = 0: каретка не установлена
car > 0: каретка установлена
sel = 0: выбор не активен
sel > 0: выбор активен *)
```

UpdateMsg* = **RECORD**

(Display.FrameMsg)

id: INTEGER;

text: Texts.Text;

beg, end: LONGINT

END;

VAR menu, barW, left, right, top, bot, asr, dsr, lsp: INTEGER; (*стандартные размеры*)

PROCEDURE Restore (F: Frame);

(*восстановление кадра F*)

PROCEDURE Suspend(F: Frame);

(*приостановить кадр F*)

PROCEDURE Extend (F: Frame; newY: INTEGER);

(*расширить кадр F до нижнего значения newY*)

PROCEDURE Reduce (F: Frame; new: INTEGER);

(*уменьшить кадр F до нижнего значения new*)

PROCEDURE Mark (F: Frame; mark: INTEGER);

(*пометить рамку F меткой*)

PROCEDURE Show (F: Frame; pos: LONGINT);

(*показать часть текста, содержащую позицию pos в кадре F*)

PROCEDURE Pos (F: Frame; X, Y: INTEGER): LONGINT;

(*преобразование координат X,Y в позицию текста*)

PROCEDURE SetCaret (F: Frame; pos: LONGINT);

(*установить каретку во фрейме Fat в позицию pos*)

PROCEDURE TrackCaret (F: Frame; X, Y: INTEGER; **VAR** keysum: SET);

(*отслеживание каретки в кадре, начиная с X, Y, и возврат нажатых клавиш мыши*)

PROCEDURE RemoveCaret (F: Frame);

(*удалить каретку из кадра F*)

PROCEDURE SetSelection (F: Frame; beg, end: LONGINT);

(*выбрать текст, растянутый на [beg, end] в F*)

PROCEDURE TrackSelection (F: Frame; X,Y: INTEGER; **VAR** keysum: SET);

(*выбор дорожки в кадре F, начиная с X, Y и возвращая нажатые клавиши мыши*)

PROCEDURE RemoveSelection (F: Frame);

(*удаление выделения из кадра F*)

PROCEDURE TrackLine (F: Frame; X, Y: INTEGER; **VAR** org: LONGINT; **VAR** keysum: SET);

(*расстановка текстовой строки в кадре F, начиная с X,Y, и возвращает начало строки и нажатые клавиши мыши*)

PROCEDURE TrackWard (F: Frame; X, Y: INTEGER; **VAR** pos: LONGINT,; **VAR** keysums: SET);

(*vtrack текстовое слово в кадре F, с указанием X,Y, и возвращает данные о позиции и нажатых клавишах мыши*)

PROCEDURE Replace (F: Frame; beg, end: LONGINT);

(*текстовый отрезок [начало, конец] был заменен; обновить кадр F*)

PROCEDURE Insert (F: Frame; beg, end: LONGINT);

(*растяжение текста [beg, end] было вставлено; обновить рамку F*)

PROCEDURE Delete (F: Frame; beg, end: LONGINT);

(*текстовый отрезок [beg, end] был удален; обновить рамку F*)

(*----- обработка сообщений -----*)

PROCEDURE NotifyDisplay (T: Texts.Text; op: INTEGER; beg, end: LONGINT);

(*уведомление менеджера дисплея об изменении состояния текста*)

PROCEDURE Call* (F: Frame; pos: LONGINT; new: BOOLEAN);

(*вызов команды, указанной в pos в кадре F. new заставляет загрузить самую новую версию*)

PROCEDURE Write* (F: Frame; ch: CHAR; fnt: Fonts.Font; col, voff: SHORTINT);

(*записывает символ ch с заданными атрибутами в позиции каретки*)

PROCEDURE Defocus* (F: Frame); (F: Frame; ch: CHAR; fnt: Fonts.Font; col, voff: SHORTINT);

(*remove caret*)

PROCEDURE Neutralize* (F: Frame);

(*удалить метки*)

PROCEDURE Modify* (F: Frame; id, dY, Y, H: INTEGER);

(*верткали перевод и расширение уменьшенного кадра F. d указывает тип (расширение уменьшения), dY - вектор перевода, Y, H - новое местоположение и высота соответственно*)

PROCEDURE Open* (F: Frame; H: Display.Handler; T: Texts.Text; org: LONGINT;

col, left, right, top, bot, asr, dsr, lsp: INTEGER);

(*открыть новый текстовый фрейм F, отображающий текст T, начиная с позиции org, с цветом фона col, полями left, right, top, bot, и геометрией линий asr, dsr, lsp = интервалы между строками ascender, descender. Установить нотификатор H*)

PROCEDURE Copy* (F: Frame; VAR F1: Frame);

(*создание копии F1 фрейма F. Инициализация на пустой фрейм*)

PROCEDURE CopyOver* (F: Frame; text: Texts.Text; beg, end: LONGINT);

(*копирование растянутого текста [beg, end] в позицию каретки в кадре F*)

PROCEDURE GetSelection* (F: Frame; VAR text: Texts.Text; VAR beg, end, time: LONGINT);

(*получить текущее выделение текста в кадре F (если есть)*)

PROCEDURE Update* (F: Frame; VAR M: UpdateMsg);

(*обновление отображения после операции редактирования*)

PROCEDURE Edit* (F: Frame; X,Y: INTEGER; Keys: SET);

(*отслеживание мыши и интерпретация команд редактирования*)

PROCEDURE Handle* (F: Display.Frame; VAR M: Display.FrameMsg);

(*стандартный обработчик для текстовых фреймов*)

PROCEDURE Text* (name: ARRAY OF CHAR): Texts.Text;

(*создание нового отображаемого текста из именованного файла. Пустое имя файла означает пустой текст*)

```

PROCEDURE NewMenu* (name, commands: ARRAY OF CHAR): Frame;
    (*создание нового фрейма меню, содержащего перечисленные команды*)
PROCEDURE NewText* (text: Taxis.Text; pos: LONGINT): Frame;
    (*создание нового стандартного текстового фрейма*)
END TextFrames.

```

Ядро Oberon

```

DEFINITION Math;  (* библиотека для вещественных чисел *)
CONST pi = 3.14159265; e = 2,71828182;
PROCEDURE sqrt(x: REAL): REAL;
PROCEDURE exp(x: REAL): REAL;
PROCEDURE Infx (x: REAL): REAL): REAL;
PROCEDURE sin(x: REAL): PEAL;
PROCEDURE cos(x: REAL): PEAL;
PROCEDURE arctan(x: REAL): REAL;
END Math.

```

```

DEFINITION MathL;  (* библиотека для longreals *)
CONST
pi = 3.141592653589793D0;
e = 2.718281828459045D0;
PROCEDURE sqrt(x: LONGREAL): LONGREAL;
PROCEDURE exp(x: LONGREAL): LONGREAL;
PROCEDURE ln(x: LONGREAL): LONGREAL;
PROCEDURE sin(x: LONGREAL): LONGREAL;
PROCEDURE cos(x: LONGREAL): LONGREAL;
PROCEDURE arctan(x: LONGREAL): LONGREAL;
END MathL.

```

```

DEFINITION Files;  (* менеджер файлов *)
TYPE Handle = RECORD END;
File = POINTER TO Handle;

    (* File - это последовательность байтов, доступ к которой осуществляется
    через хэндл (указатель на него). Файлы хранятся на диске и на них можно
    ссылаться через имя, введенное в каталог файлов *)

```

```

Rider = RECORD
    res: INTEGER;
    eof: BOOLEAN;
    file: File
END;

```

(*Доступ к элементам файлов осуществляется через rider, который имеет позицию, выдвигаемую при чтении или записи данных. Позиция - это целое число от 0 до длины элемента, к которому прикреплен райдер. Поля eof и res служат

параметрами результата файловых процедур*)

PROCEDURE Old(name: **ARRAY OF** CHAR): File

(*файл с заданным именем. NIL, если имя отсутствует в каталоге*)

PROCEDURE New(name: **ARRAY OF** CHAR): File;

(*новый файл с заданным именем*)

PROCEDURE Register(f: File);

(*Закрывает файл f и регистрирует его под своим именем в каталоге.

Если имя уже существует, то соответствующий старый файл снимается с регистрации*)

PROCEDURE Close(f: File);

PROCEDURE Purge(f: File);

PROCEDURE Length(: File): LONGINT; (*количество байт в файле*)

PROCEDURE Set(**VAR** r: Rider; f: File; pos: LONGINT)

(*ассоциировать райдер r с файлом f в позиции pos. r.eof := FALSE*)

PROCEDURE Read(**VAR** r: Rider; **VAR** x: BYTE);

(*Считывание байта и продвижение райдера на одну позицию. Если в end, r.eof := TRUE и x := 0X*)

PROCEDURE ReadBytes(**VAR** r: Rider; **VAR** x: **ARRAY OF** BYTE; n: INTEGER);

(*считывание n байт и продвижение райдера на n позиций.

Если в end, eof = TRUE и r.res := количество запрошенных, но не прочитанных байт*)

PROCEDURE Write(**VAR** r: Rider; x: BYTE);

(*запись байта и продвижение райдера на одну позицию*)

PROCEDURE WriteBytes(**VAR** r: Rider; **VAR** x: **ARRAY OF** BYTE; n: INTEGER):

(*запись n байт и продвижение райдера на n позиций*)

PROCEDURE Pos(**VAR** r: Rider): LONGINT;

PROCEDURE Base(**VAR** r: Rider): File;

PROCEDURE Rename(old, new: **ARRAY OF** CHAR; **VAR** res: INTEGER);

(*res = 0: переименовано; res = 1: новое имя уже существовало и теперь обозначает переименованный файл; res = 2: старое имя отсутствует в дитектории; res = 3: имя недопустимо; res = 4: имя слишком длинное*)

PROCEDURE Delete(name: **ARRAY OF** CHAR; **VAR** res: INTEGER);

(* res = 0: удалено; res = 2: имя отсутствует в дитектории; res = 3: имя недопустимо; res = 4 имя слишком длинное *)

END Files.

DEFINITION Diskette; (* менеджер дискет *)

TYPE EntryHandler* = **PROCEDURE** (name: **ARRAY OF** CHAR; date, time: INTEGER; size: LONGINT);

VAR res: INTEGER; (*произошла ошибка в результате файлово-ориентированной операции = (res # 0)*)

err: SHORTINT; sect: LONGINT;; busy: BOOLEAN; (*состояние драйверов устройств*)

(*драйверы устройств*)

PROCEDURE Reset;

PROCEDURE GetSector (sec: INTEGER; **VAR** buf: **ARRAY OF** BYTE; off: INTEGER);

PROCEDURE PutSector (sec: INTEGER; **VAR** buf: **ARRAY OF** BYTE; off: INTEGER);

PROCEDURE Format;

```

(*обработчик каталога*)
PROCEDURE InitDir (format: CHAR); (*формат для будущего расширения*)
PROCEDURE ReadDir;
PROCEDURE WriteDir;
PROCEDURE GetData (VAR data, time, nofFiles, nofClusters: INTEGER); (*получение
данных тома*)
PROCEDURE Enumerate (proc: EntryHandler)

(*обработчик файла*)
PROCEDURE ReadAll;
PROCEDURE ReadFile (name: ARRAY OF CHAR);
PROCEDURE WriteFile (name: ARRAY OF CHAR);
PROCEDURE Delete (имя: ARRAY OF CHAR);
END Diskette.

```

DEFINITION Input; (* драйверы клавиатуры и мыши *)

```

PROCEDURE Available(): INTEGER;
    (*количество символов, доступных с клавиатуры*)
PROCEDURE Read (VAR ch: CHAR);
    (*следующий символ с клавиатуры*)
PROCEDURE Mouse (VAR keys: SET; VAR x,y: INTEGER);
    (*текущие координаты и настройки клавиш мыши*)
    keys = 0   нажата правая клавиша
    keys = 1   нажата средняя клавиша
    keys = 2   нажата левая клавиша*)
PROCEDURE SetMouseLimits (w, h: INTEGER);
    (*определяют ширину и высоту прямоугольника, в котором движется мышь*)
PROCEDURE Time(): LONGINT;
    (*текущее системное время в единицах 1/300 секунды*)
END Input.

```

DEFINITION SCC; (* драйвер SCC *)

(*Модуль драйвера контроллера последовательной связи (zlog 28530). Данные передаются блоками. Каждый блок содержит две части: заголовок и данные*)

```

TYPE Header = RECORD
    valid: BOOLEAN;
    dadr, sadr, typ: SHORTINT;
    len: INTEGER (*данных, следующих за заголовками*)
    destLink, srclink: INTEGER (*номера ссылок*)
END;

```

(*dadr - номер машины приемника, len - длина (количество байт) части данных.
typ, destLink и srclink не интерпретируются SCC*)

PROCEDURE Start(filter; BOOLEAN);

(*инициализация SCC*)

PROCEDURE Send(VAR head, buf: **ARRAY OF** BYTE);

(*отправка buf[0] ... buf[head.len-1] в head.adr*)

PROCEDURE Available(): INTEGER;

(*количество байтов, доступных из буфера приемника. Buffer содержит поток
полученных байтов, включая заголовки и части данных*)

PROCEDURE ReceiveHead(VAR head: **ARRAY OF** BYTE);

(*считывание заголовка из буфера приемника*)

PROCEDURE Receive (VAR x: BYTE);

(*считывание байта из буфера приемника*)

PROCEDURE Skip(m: INTEGER)

(*пропустить m байт в буфере приемника*)

PROCEDURE Stop;

(*выключить SCC*)

END SCC.

DEFINITION V24; (* драйвер V24*)

(* управляемый прерываниями UART-канал В *)

PROCEDURE Start(CSR, MR2: CHAR);

(* Регистр выбора тактовой частоты:

66X: 1200 бит/с

88X: 2400 бит/с

0VBX: 9600 бит/с

Регистр режима 2:

7%: стоп-бит

OFX: 2 стоповых бита *)

PROCEDURE SetOP(s: SET); (*выходной порт*)

PROCEDURE ClearOP(s: SET);

(*0:DTR, 1:RTS *)

PROCEDURE IP(n: INTEGER): BOOLEAN, (*входной порт*)

PROCEDURE SR(r: INTEGER): BOOLEAN;

(*Status Register. 0: Rx rdy, 2: Tx rdy, 4: превышение*)

PROCEDURE Avalable(): INTEGER;

PROCEDURE Receive(VAR x: BYTE);

```

PROCEDURE Send(x: BYTE);
PROCEDURE Break;
PROCEDURE Stop;
END V24.

```

DEFINITION Printer; (* интерфейс принтера *)

```

VAR es: INTEGER, (*результат*)

```

```

PROCEDURE Open(VAR name, user: ARRAY OF CHAR; password: LONGINT);
(*tes = 0: открыто, 1: нет принтера, 2: нет связи, 3: плохой ответ, 4: нет
разрешения*)

```

```

PROCEDURE Font (nfo: SHORTINT; VAR name: ARRAY OF CHAR); (*установить шрифт*)

```

```

PROCEDURE String (x,y: INTEGER; VAR s: ARRAY OF CHAR; nfo: SHORTINT);
(*разместить строку*)

```

```

PROCEDURE ContString (VAR s: ARRAY OF CHAR; fno: SHORTINT); (*разместить строку
продолжения*)

```

```

PROCEDURE Line (x,y, w, h: INTEGER); (*размещение горизонтальной или
вертикальной линии*)

```

```

PROCEDURE XLine (x,y, dx, dy: INTEGER); (*разместить линию общих направлений*)

```

```

PROCEDURE Circe (x,y, a, b: INTEGER); (*круг места или эллипсис*)

```

```

PROCEDURE Shade (x,y, w, h, col: INTEGER); (*заштриховать площадь*)

```

```

PROCEDURE Picture (x,y, w, h, mode: INTEGER; adr: LONGINT); (*разместить
картинку*)

```

```

PROCEDURE Page(nofcopies: INTEGER); (*распечатать текущую страницу*)

```

```

PROCEDURE Close; (*заккрыть соединение*)

```

```

END Printer.

```

DEFINITION Oberon; (* менеджер системы *)

```

IMPORT Display, Viewers, Texts;

```

```

CONST

```

```

consume = 0; track = 1; (*идентификаторы для входящих сообщений*)

```

```

defocus = 0; neutralize = 1; mark = 2; (*идентификаторы для управляющих
сообщений*)

```

```

TYPE

```

```

Painter = PROCEDURE (x,y: INTEGER);

```

```

Marker = RECORD

```

```

    Fade, Draw: Painter

```

```

END;

```

```

Cursor = RECORD

```

```

        on: BOOLEAN; m: Marker; X,Y: INTEGER
    END;

ParList = POINTER TO ParRec;

ParRec = RECORD
    vwr: Viewers.Viewer;    (*вьювер вызывающего*)
    frame: Display.Frame;    (*подкадр вызывающего*)
    text: Texts.Text;        (*список параметров*)
    pos: LONGINT              (*начальная позиция списка параметров*)
END;

InputMsg = RECORD(Display.FrameMsg)
    id: INTEGER;              (*id сообщения*)
    modes, keys: SET;         (*текущие режимы и клавиши мыши*)
    X,Y: INTEGER;             (*текущее положение мыши*)
    ch: CHAR                   (*текущий символ*)
END;

ControlMsg = RECORD(Display.FrameMsg)
    id: INTEGER,              (*Id сообщения*)
    X,Y: INTEGER              (*текущее положение мыши*)
END;

SelectionMsg = RECORD(Display.FrameMsg)
    time: LONGINT;
    text: Texts.Text;
    beg, end: LONGINT
END;

CopyOverMsg* = RECORD(Display.FrameMsg)
    text: Texts.Text;
    beg, ends: LONGINT
END;

CopyMsg* = RECORD(Display.FrameMsg)
    F*: Display.Frame
END;

Task = POINTER TO TaskDesc; (*устанавливаемые задачи*)
Handler = PROCEDURE;

TaskDesc = RECORD
    safe: BOOLEAN;            (*безопасные задачи не удаляются после ловушек*)
    handle: Handler
END;

```

VAR

(*конфигурация*)

FocusViewer: Viewers.Viewer; (*просмотрщик текущего фокуса*)

Log: Texts.Text; (*текст системного лога*)

Par: ParList; (*фактические параметры для следующей команды*)

User: ARRAY 8 OF CHAR; Password: LONGINT; (*текущий пользователь*)

CurFnt, CurCol:, CurOff SHORTINT, (*текущий шрифт, цвет, вертикальное смещение*)

Arrow, Star: Marker;

Mouse, Pointer: Cursor;

(*идентификация пользователей*)

PROCEDURE SetUser (VAR user, password: **ARRAY OF** CHAR);

(*часы*)

PROCEDURE GetClock (VAR, d: LONGINT);

PROCEDURE SetClock (t, d: LONGINT);

PROCEDURE Time (): LONGINT;; (*в единицах измерения 1/300 сек*)

(*работа с курсором*)

PROCEDURE OpenCursor (VAR c: Cursor);

PROCEDURE FadeCursor (VAR c: Cursor);

PROCEDURE DrawCursor (VAI c: Cursor; VAR m: Marker; X, Y: INTEGER);

(*управление дисплеем*)

PROCEDURE OpenDisplay (UW, SW, H: INTEGER);

(*инициализация нового дисплея с шириной пользовательской дорожки UW, шириной системной дорожки SW и высотой H*)

PROCEDURE DisplayWidth (X: INTEGER): INTEGER;

(*получить ширину дисплея в X*)

PROCEDURE DisplayHeight (X: INTEGER): INTEGER;

(*получить высоту дисплея в X*)

PROCEDURE OpenTrack (X, W: INTEGER);

(*открыть новый трек шириной W в X*)

PROCEDURE UserTrack (X: INTEGER): INTEGER;

(*получить левое поле пользовательской дорожки в точке X*)

PROCEDURE SystemTrack (X. INTEGER): INTEGER;

(*получить левое поле системной дорожки в X*)

PROCEDURE AllocateUserViewer (DX: INTEGER; VAR X, Y: INTEGER);

(*выделить новый пользовательский вьювер в дисплее по адресу DX*)

```

PROCEDURE AllocateSystemViewer (DX: INTEGER; VAR X, Y : INTEGER);
    (*выделение нового системного вьювера на дисплее по адресу DX*)

PROCEDURE PassFocus (V: Viewers.Viewer);
    (*передать фокус на вьювер V*)

PROCEDURE RemoveMarks (X,Y, W, H: INTEGER);
    (*удалить метки в пределах заданного прямоугольника*)

PROCEDURE MarkedViewer (): Viewers.Viewer;
    (*отметьте места просмотра, помеченные звездообразными указателями*)

(*интерпретация команд*)

PROCEDURE ShowMenu (VAR cmd: INTEGER; X, Y: INTEGER; menu: ARRAY OF CHAR);
    (* menu = {команда "|" } команда.
    Допускается шесть команд, 6 > cmd >= -1.
    cmd =5: выбрана первая команда
    cmd =0: выбрана последняя команда
    cmd = -1: выбор отсутствует *)

PROCEDURE Call (VAR name: ARRAY OF CHAR; par: ParLst; new: BOOLEAN; VAR res:
INTEGER);
    (* вызов команды name и передача параметра lst par. Опция new запрашивает
загрузку модуля. Done = (res = 0) *)

PROCEDURE GetSelection (VAR text: Texts.Text; VAR beg, end, time: LONGINT);
    (*получить последнее выделение текста, Существование выбранного текста =
(time = 0) *)

PROCEDURE Install (T: Task);
    (*установить новую задачу T*)

PROCEDURE Remove (T: Task);
    (*удалить установленную задачу T*)

PROCEDURE Collect;
    (*затребовать сборщики мусора*)

PROCEDURE SetFonts* (fnt: Fonts.Font);
    (*установить текущий шрифт*)

PROCEDURE SetColor* (col: SHORTINT);
    (*установить текущий цвет*)

PROCEDURE SetOffset* (voff: SHORTIN);
    (*установка текущих вертикальных смещений*)

```

END Oberon.

Примечание;

Установленные задачи считаются фоновыми активностями. Они активируются центральным циклом, когда не обнаружено никаких входных событий. Например, сборщик мусора реализован как установленная задача.

Обратите внимание, что установленные задачи могут быть аннулированы после выгрузки (или замены) их главного модуля. Небезопасные задачи автоматически удаляются после системной ловушки, чтобы избежать бесконечного повторения одной и той же ошибки.

Примеры

Запись метки времени в системный журнал

PROCEDURE TimeStamp:

BEGIN

Texts WriteString(W, Timestamp); Texts WriteInt(W, Oberon Time(), 1) Texts WriteLn(W);

Texts Append(Oberon Log, W.buf)

END TimeStamp;

где

VAR W: Texts.Writer

глобально определенный инициализированный посредством Texts.OpenWriter(W).

Примечания:

1. Обычно достаточно одного (глобального) писателя на модуль.
2. Если вы хотите, чтобы определенная часть выходного текста отображалась новым шрифтом, например, курсивом Syntax10i.Scn.Fnt, вызовите Texts.SetFont(W, Fonts.This("Syntax10i.Scn.Fnt")) – перед написанием этой части и Texts.SetFont(W, Fonts.Default) перед продолжением написания обычного текста.

Обработка выделенного текста

PROCEDURE CountWords;

VAR

T: Texts.Text;

R: Texts.Reader;

beg, end, pos, time: LONGINT;

words: INTEGER;

ch: CHAR;

BEGIN

words := 0;

Oberon.GetSelection(T, beg, end, time); (*получение последних выборок*)

IF time >= 0 **THEN** (*если он существует*)

Texts.OpenReader(R, T, beg); pos := beg; (*настройка считывающего устройства и инициализация pos*)

Texts Read(R, ch); INC(pos); (*читаем следующий символ*)


```

IF (pos # end) & (ch >"") THEN
    REPEAT Texts.Read(R, ch); INC(pos) UNTIL (pos = end) OR (ch <="");
INC(words)
END;

WHILE pos # end DO
    (* (pos # end) & (ch <=" ") *)
    REPEAT Texts.Read(R, ch); INC(pos) UNTIL (pos = end) OR (ch >" ");
    IF pos # end THEN
        REPEAT Texts.Read(R, ch); INC(pos) UNTIL (pos = end) OR (ch <=" ");
        INC(words)
    END
END
END;

Texts.WiiteString(W, "WordCount = "); Texts.WriteLine(W, words, 1);
Texts.WriteLine(W);

Texts.Append(Oberon Log, W.buf) (*добавление в системный журнал*)
END CountWords;

где
VAR W: Texts.Writer;

глобально определяется и инициализируется посредством Texts.OpenWriter(W).

```

Открыть вьювер в системном треке, генерирование и отображение текстовых данных

```

PROCEDURE Directory;

VAR Menu, Main: TextFrames.Frame; T: Texts.Text; V: Viewers.Viewer; X,Y:
INTEGER;

BEGIN

T := TextFrames.Text(""); (*генерирует новый пустой текст для отображения во
фрейме*)

Menu := TextFrames.NewMenu("Directory", StandardMenu); (*генерирует стандартный
фрейм меню*)

Main := TextFrames.NewText(T, 0); (*генерирует стандартный текстовый фрейм*)

Oberon.AllocateSystemViewer(Oberon.Par.vwr.X, X, Y);

V := MenuViewers.New(Menu, Main, TextFrames.menuH, X, Y); (*открывает
стандартный просмотрщик меню*)

TextFrames.Mark(Main, -1); (*установить вертикальную метку со стрелкой*)

Diskette.Enumerate(Lister); (*передаем Lister-проседуру в перечислитель*)

Texts.Append(T, W.buf); (*добавить писателя в T и отображение записанного
текста*)

TextFrames.Mark(Main, 1) (*восстановить метку позиции*)

END Directory;

```

```

где

CONST StandardMenu = "System.Close System.Copy System.Grow Edit.Search
Edit.Store";

VAR T: Texts.Text; W: Texts.Writer;

```

определены глобально, W глобально инициализируется посредством Texts.OpenWriter(W), а Lister – это (вызванная) процедура отображающая записи каталога:

```
PROCEDURE Lister (name: ARRAY OF CHAR; date, time: INTEGER; size: LONGINT);  
BEGIN  
  Texts.WriteString(W, name);  
  Texts.Write(W, " "); Texts.Writeln(W, size, 1);  
  Texts.Write(W, " "); Texts.WriteDate(W, time, date);  
  Texts.Writeln(W)  
END Lister;
```

Примечания:

1. Приведенная выше программа генерирует весь выходной текст перед его отображением. В качестве альтернативы, если вы переместите оператор Texts.Append(T, W.buf) в процедуру Lister, то каждая сгенерированная запись каталога будет выведена на экран немедленно.

2. Oberon. AllocateSystemViewer (Oberon.Par.vwr.X, X, Y) – это стандартное предложение для размещения нового системного вьюера в пределах трека, с которого была вызвана команда. Конечно, возможны и индивидуальные алгоритмы. Например, если требуется, чтобы новый вьюер закрывал самый нижний вьюер, за исключением случаев, когда указатель отменяет это, то алгоритм будет следующим

```
PROCEDURE AllocateSystemViewer (DX: INTEGER; VAR X, Y: INTEGER);  
  VAR bot: Viewers.Viewer;  
BEGIN  
  IF Oberon.Pointer.on THEN X := Oberon.Pointer.X; Y := Oberon.Pointer.Y  
  ELSE bot := Viewers.This(Oberon.SystemTrack(DX), 0); X:= bot.X; Y := bot.H -  
Viewers.minH  
  END  
END AllocateSystemViewer;
```

3. TextFrames.NewText генерирует стандартный текстовый фрейм. Следующая последовательность операторов создает текстовый кадр с индивидуальным обработчиком и настроенной геометрией.

```
NEW(F); Open(F, Handle, text, pos, col, left, right; top, bot, asr, dsr, lsp);  
где F – типа TextFrames.Frame.
```

Открыть вьюер в пользовательском треке и отобразить существующий текст

```
PROCEDURE OpenText;  
VAR par: Oberon.ParList; Text: TextFrames.Frame; S: Texts.Scanner;  
V: Viewers.Viewer; X, Y: INTEGER;  
BEGIN  
par := Oberon.Par; (* доступ к параметрам *)  
Text = par.frame(TextFrames.Frame); (* вызов фрейма *)  
TextFrames.Mark(Text, -1); (* метка со стрелкой *)  
Texts.OpenScanner(S, par.text, par.pos); (* открыть сканер в позиции списка  
параметров *)
```

```

Texts.Scan(S);                                (*получение символа *)
IF S.class = Texts.Name THEN
Oberon.AllocateUserViewer(par.vwr.X, X, Y);
V := MenuViewers.New(
TextFrames.NewMenu(S.s, StandardMenu);
TextFrames.NewText(TextFrames.Text(S.s), 0);
TextFrames.menuH, X, Y);
END;
TextFrames.Mark(Text, 1)                      (* восстановление метки положения *)
END OpenText;

```

Замечание:

Oberon AllocateUserViewer(par.vwr.X, X. Y) – это стандартное предложение для размещения нового вьюера в пользовательском треке вызывающего пользователя. Опять же, возможны и индивидуальные алгоритмы.

Grow-отображение

```

PROCEDURE Grow;
VARV, newV: Viewers.Viewer; M: Oberon.CopyMsg; N: Viewers ViewerMsg; DH:
INTEGER;
BEGIN
V:= Oberon.Par.vwr;                          (*получить вьювер оригинатора*)
DH := Oberon.DisplayHeight(V.X);              (*получить высоту данного трека*)
IF V.H < Oberon.DisplayHeight(V.X) THEN      (*если вьювер маленький*)
Oberon.OpenTrack(V.X, V.W);                  (*открыть наложенные треки*)
V.handle(V, M); newV := M.F(Viewers.Viewer); (*получить копию вьювера*)
Viewers.Open(newV, V.X, DH);                 (*открыть новые большие вьюеры*)
N.id := Viewers.restore; newV.handle(newV, N) (*попросить новый вьювер
нарисовать себя*)
END
END Grow;

```

Замечание:

Команда Grow является общей в том смысле, что она может работать с экземплярами вьюверов любого (текущего или будущего) класса. Обычно (и неизбежно) общие команды используют "передачу сообщений" вместо обычных вызовов процедур. Этот объектно-ориентированный стиль будет более подробно рассмотрен в следующей главе. Также обратите внимание, что фактически копия оригинального вьюера открывается в новом треке. Когда позже этот трек будет закрыт, оригинальный появится снова.

Обработка текста вьюера или последовательности текстов в зависимости от контекста

```

PROCEDURE ProcessText;
VAR par: Oberon.ParList; Main: TextFrames.Frame; S: Texts.Scanner; T:
Texts.Text;
BEGIN

```

```

par := Oberon.Par; (*параметр доступа*)
F par frame = par.ywr.dsc THEN (*команда во фрейме меню*)
  IF par.vwr.dsc.next IS TextFrames.Frame THEN
    Main = par.vwr.dsc.next(TextFrames.Frame); (* основной текст фрейма *)
    TextFrames.Mark(Main, -1) (* устанавливаем знак стрелки *)
    Process(Main.text); (* обработка отображаемого текста *)
    TextFrames.Mark(Main, 1) (* восстановить метку положения *)
  END
ELSE (* команда в главном текстовом фрейме *)
  Main := par.frame(TextFrames.Frame);
  TextFrames.Mark(Main, -1) (* установить метку стрелки *)
  Texts.OpenScanner(S, par.text, par.pos); (* открыть сканер в позиции списка параметров *)
  Texts.Scan(S); (* получить первый символ *)
  WHILE S.class = Texts.Name DO
    Texts.Open(T, S.s); (* открыть текст из файла *)
    Process(T); (* обработать этот текст *)
    Texts.Scan(S) (* получить следующий символ *)
  END;
  TextFrames.Mark(Main, 1) (* восстановить позиционную метку *)
END
END ProcessText;

```

Удаление выделенной части текста в помеченном вьювере

```

PROCEDURE Delete;
VAR Main: TextFrames.Frame; V: Viewers.Viewer;
BEGIN
  V := Oberon.MarkedViewer(); (* получить отмеченный вьювер *)
  Main := V.dsc.next(TextFrames.Frame); (* основной текстовый фрейм отмеченного вьювера *)
  IF Main.sel > 0 THEN (* если существует выделение *)
    Texts.Delete(Main.text, Main.selbeg.pos, Main.selend.pos) (*удаление текста*)
  END
END Delete;

```

Копирование последнего выделенного фрагмента текста в позицию каретки

```

PROCEDURE CopyText;
VAR Main: TextFrames.Frame; buf: Texts.Buffer; V: Viewers.Viewer; time: LONGINT;
BEGIN
  Oberon.GetSelection(T, beg, end, time); (* получаем последнюю выборку *)
  IF time >= 0 THEN (* если она существует *)
    Texts.OpenBuffer(buf);

```

```

    Texts.Save(T, beg, end, buf); (* сохранить текст в буфере *)
    V := Oberon.FocusViewer; (* получить средство просмотра фокуса *)
    IF (V.dsc # NIL) & (V.dsc.next IS TextFrames.Frame) THEN (* если просмотрщик
текста *)
        Main := V.dsc.next(TextFrames.Frame); (* основной текстовый фрейм *)
        IF Main.car > 0 THEN (* если установлен каретка *)
            Texts.Insert(Main.text, Main.carloc.pos, buf) (* вставка текста в позицию
каретки *)
        END
    END
END
END CopyText;

```

Копирование шрифта из видимой отмеченной позиции в выделенный текст

```

PROCEDURE CopyFont;
VAR F: TextFrames.Frame; T: Texts.Text; R: Texts.Reader; V: Viewers.Viewer;
beg, end, time: LONGINT; X, Y: INTEGER; ch: CHAR;
BEGIN
    Oberon.GetSelection(T, beg, end, time); (* получить последнюю выборку *)
    IF (time >= 0) & Oberon.Pointer.on THEN (* если указатель найден и виден *)
        X := Oberons.Pointer.X; Y := Oberon.Pointer.Y;
        V := Viewers.This(X, Y); (* отмеченный вьювер *)
        IF (V.dsc # NIL) & (V.dsc.next IS TextFrames.Frame) THEN
            F := V.dsc.next(TextFrames.Frame);
            IF (X >= F.X) & (X < F.X + F.W) & (Y >= F.Y) & (Y < F.Y + F.H) THEN
                Texts.OpenReader(R, F.text, TextFrames, Pos(F, X, Y)); (* позиционируем
читателя *)
                Texts.Read(R, ch); (* считывание отмеченных символов *)
                Texts.ChangeLooks(T, beg, end, {0}, R.fnt, 0, (*изменение только шрифта*)
            END
        END
    END
END CopyFont;

```

Переместить каретку на следующий символ выделенный курсивом

```

PROCEDURE Searchitalics;
VAR Main: TextFrames.Frame; R: Texts.Reader; italic: Fonts.Font; V:
Viewers.Viewer;
pos: LONGINT; ch: CHAR;
BEGIN
    italic := Fonts.This("Syntax1 0i.Scn.Fnt");
    V := Oberon.FocusViewer; (* получить фокус вьювера *)

```

```

IF (V.dsc # NIL) & (V.dsc.next IS TextFrames.Frame) THEN      (* если вьювер
текста *)
    Main := V.dsc.next(TextFrames.Frame);                        (* основные текстовые кадры *)
    IF Main.car > 0 THEN                                          (* если каретка установлена *)
        Texts.OpenReader(R, Main.text, Main.carloc.pos);        (* открыть читалку в
позиции каретки *)
        Texts.Read(R, ch); :
        WHILE ~R.eot & (R.fnt # italic) DO Texts.Read(R, ch) END; (* чтение потока
символов *)
        IF ~R.eot THEN                                          (* не конец текста *)
            pos := Texts.Pos(R);                                  (* позиция читателя *)
            TextFrames.RemoveSelection (Main);                   (* удалить все метки *)
            TextFrames.RemoveCaret(Main); .
            Oberon.RemoveMarks(Main.X, Main.Y, Main.W, Main.H);
            TextFrames.Show(Main, Max(0, pos - 200); (* показать текст в позиции *)
            TextFrames.SetCaret(Main, pos)                       (* каретку в новое положение *)
        END
    END
END
END SearchItalics;

```

где Max - функция максимума.

Программирование новых классов фремов и типов вьюверов

Фреймы как активные объекты

Фреймы - это основные объекты системы отображения Oberon. Они представляют собой более или менее автономные прямоугольные области на экране и могут быть вложенными. Экран иерархически организован следующим образом:

Дисплей > треки > вьюверы > фреймы данных.

Дисплей, треки и вьюверы - это объекты, которыми управляет модуль обработчика вьювера (Viewers).

Из-за плиточного подхода Oberon все эти фреймы полностью видимы или невидимы, т.е. на этом уровне нет частичного перекрытия. Существует класс стандартных "меню-вьюверов" (обрабатываемых модулем MenuViewers). Каждое меню-вьювер содержит ровно два вертикально смежных фрейма данных: фрейм заголовка (включающий заголовок и меню) и основной фрейм данных. Основной фрейм можно рассматривать как рабочий стол для приложения или задачи. Он может быть вложенным, т.е. содержать дополнительные подфреймы. Поскольку отдельный интерпретатор команд связан с каждым фреймом, реализация нового класса фреймов данных является наиболее мощным способом добавления функциональности в систему Oberon.

Мы намеренно использовали термин класс. На самом деле, фреймы реализованы в Oberon как "активные объекты" в смысле объектно-ориентированного программирования. Вызовы специфических для фреймов процедур обработки выполняются системными подпрограммами в виде передачи сообщений. Центральный цикл Oberon инициирует реакции на действия пользователя, посылая сообщения вьюерам. Мы подчеркиваем что использование объектно-ориентированной парадигмы в связи с обработкой фреймов является необходимостью для того, чтобы ядро могло вызывать командные интерпретаторы, идентичность которых ему известна.

Любой обработчик, связанный с определенным классом вьюеров, например, меню-вьюер, должен обрабатывать сообщения, по крайней мере, от трех источников: От менеджера вьюеров, от менеджера документов и от центрального цикла. Сообщения от менеджера вьюера уведомляют об изменении состояния вьюера. Например, менеджер вьюера Viewers посылает сообщение всякий раз, когда скрытый вьюер становится видимым (и поэтому должен восстановить его содержимое), или когда размер вьюера изменился из-за того, что его нижний соседний был открыт, изменен или закрыт. Сообщения от менеджера документов напоминают о необходимости обновления содержимого вьюера после операции редактирования.

Наконец, сообщения от модуля Oberon уведомляют вьюер об общесистемных событиях, такие как события ввода (например, "нажатие кнопки мыши в позиции X, Y) или запрос о последнем выборе текста (рассматривается в Oberon как стандартный ввод). Часть интерпретатора обрабатывающая события ввода, должна рассматриваться как редактор, связанный с классом вьюера. В случае вьюера, отображающего текст, это текстовый редактор, в случае графики – графический редактор, а в случае вьюера, представляющего почтовый ящик, это редактор электронной почты.

Типичный обработчик вьюера (например, меню-вьюера) обрабатывает сообщения, ориентированные на вьюера непосредственно и делегирует обработку более специфичных для данных сообщений, передавая их в подфреймы. Яркими примерами сообщений, ориентированных на вьюера, являются запросы на восстановление вьюера или изменение его размера. Примерами сообщений, специфичных для данных, являются выбор текста или вызов команды путем указания на ее имя. Обратите внимание, что в результате обработки сообщений, ориентированных на вьюеры, могут появиться новые и более тонкие запросы. Например, запрос на изменение размера меню-вьюера разрешается в запросы на изменение размера одного или обоих его подфреймов. Подводя итог, можно сказать, что обработчики вьюеров обычно выступают в роли посредников и отправителей сообщений, которые должны быть обработаны обработчиками их подфреймов.

Каноническое разложение приложения

Модулирование путем разделения задач является одной из наиболее эффективных техник, применяемых при проектировании больших программ. Наша концепция класса фреймов, отображающих данные определенного типа, предоставляет прекрасную возможность продемонстрировать эту технику. Можно выделить следующие отдельные части реализации приложения:

пакет инструментов, интерпретатор команд, обработчик отображения и менеджер данных.

На эти части возлагаются следующие задачи:

Набор команд, работающих с данными (Tool Package – пакет инструментов)

Реакция на входные действия в фрейме дисплея (Comm.Int – интерпретатор команд)

Отображение видимых объектов (Disp.Handler – обработчик отображения)

Инкапсуляция управления данными, не заботясь, как они отображаются (Data Manager – менеджер данных). Включает в себя набор операций редактирования, поддерживая внутреннюю структуру данных, описывая текущее состояние данных.

Естественно, назначен отдельный модуль для каждой из этих частей. Вскоре мы увидим, что интерпретатор команд и менеджер отображения предпочтительно объединить в один модуль, потому что, они оба должны работать с одним и тем же ассоциированным фреймом дисплея. Это приводит к следующей канонической конфигурации модулей, в случае стандартных текстов, графиков, картинок и MyData, например:

Tool Package	Edit	Draw	Paint	MyTool
Comm.Int & Disp.Handler	Textframes	GraphicFrames	PictureFrames	MyFrames
Data Manager	Texts	Graphics	Pictures	MyData

Обратите внимание, что один и тот же менеджер данных может использоваться разными обработчиками отображения и интерпретаторами команд. Менеджеры данных служат связующим звеном между различными приложениями и таким образом, обеспечивают оптимальную интеграцию. Например, если графическая система основана на модуле Texts для управления текстом титров, то текст может свободно обмениваться между фреймами этих классов: Графические фреймы и текстовые фреймы интегрированы.

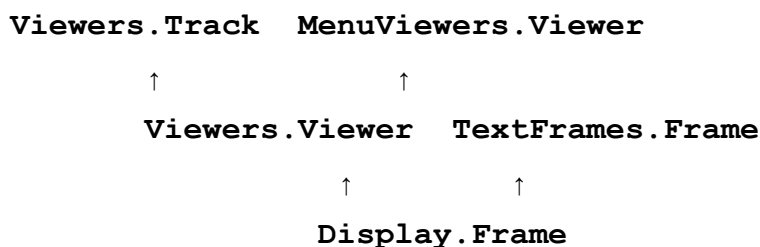
Учебный пример 1: Вьюеры текста

До сих пор мы не объяснили, как объекты и сообщения реализуются в языке Oberon. В отличие от типичных систем объектно-ориентированного программирования, полный набор сообщений, понимаемых объектом не обязательно указывать вместе с определением класса объекта. Вместо этого Oberon использует более гибкий двойной подход: Сообщения определяются в модулях отправителя. Например, сообщения первого из вышеперечисленных видов определяются в модуле Viewers, сообщения второго вида определяются в модуле редактора (TextFrames в случае с текстом), а сообщения третьего вида определяются в модуле Oberon, который обнаруживает события ввода.

Грубо говоря, язык Oberon поддерживает подклассификацию (с помощью средств расширения типов), но сообщения и обработчики сообщений не институционализированы в виде языковой конструкции. Мы реализовали описанную выше схему, используя процедурные переменные и применяя средство расширения записей Oberon.

Теперь мы проиллюстрируем эту модель с помощью стандартных

текстовых вьюверов, т.е. вьюверов меню, чьи подфреймы (меню и основной) являются текстовыми фреймами. Нижеследующее изложение может служить в качестве учебника и шаблона для разработчиков классов фреймов или типов вьюверов. Напомним, что структура данных дисплея представляет собой иерархию фреймов отображения. Ограничивая наши объяснения текстовыми вьюверами, мы имеем следующую иерархию типов:



Модуль `Display` определяет базовые типы фреймов и фреймовых сообщений, а также тип обработчика сообщений.

(Пустое) базовое сообщение служит корнем в (потенциально неограниченной) иерархии сообщений, которые могут быть принимаемых фреймами. Модуль `Viewers` предоставляет определение типа `Viewer`, который является расширением (вариантом) типа `Display.Frame`. Меню-вьюверы основаны на `Viewers.Viewer` определенном в модуле `MenuViewers`.

В определении `Display`:

TYPE

```

Frame = POINTER TO Frame.Desc;
FrameMsg = RECORD END;
Handler = PROCEDURE (Frame, VAR FrameMsg);
FrameDesc = RECORD
dsc, next: Frame; (* сын, брат *)
X,Y, W, H: INTEGER;
handle: Handler
END; :

```

В определении `Viewers`:

TYPE

```

Viewer = POINTER TO ViewerDesc;
ViewerDesc = RECORD (Display.FrameDesc)
state: INTEGER
END;

```

В определении `MenuViewers`:

TYPE

```

Viewer = POINTER TO ViewerDesc;

```

```
ViewerDesc = RECORD (Viewers.ViewerDesc)
```

```
menuH: INTEGER
```

```
END;
```

Ниже приведены объявления сообщений, которые, как ожидается, будут обрабатываться каждым вьювером. Обратите внимание, что они распределены по нескольким модулям, а не сосредоточены в одном месте (как это было бы в - "обычной" объектно-ориентированной модели) .

В определении Viewers:

```
ViewerMsg = RECORD (Display.FrameMsg)
```

```
id: INTEGER; :X,Y, W, H: INTEGER; (* новый прямоугольник *)
```

```
состояние: INTEGER (* новое состояние *)
```

```
END;
```

```
(* сигнализирует об изменении состояния вьювера
```

```
id = 0: восстановить вьювер
```

```
1: изменить размер в нижней части
```

```
2: приостановить просмотр *)
```

В определении TextFrames:

```
UpdateMsg = RECORD (Display.FrameMsg)
```

```
id: INTEGER; (* операция *)
```

```
text: Texts.Text; (* редактируемый текст *)
```

```
beg, end: LONGINT (* растяжение *)
```

```
END;
```

```
(* сигнализирует об изменении содержимого
```

```
id = 0: фрагмент [beg end] заменен
```

```
1: отрезок [beg, end] вставлен
```

```
2: отрезок [beg, end] удален *)
```

В определении Oberon:

```
InputMsg = RECORD (Display.FrameMsg)
```

```
id: INTEGER; (* операция *)
```

```
modes, keys: SET; (* мышь *)
```

```
X, Y: INTEGER; (* позиции *)
```

```
ch: CHAR (* чтение символов *)
```

```
END;
```

```
(* сигнализирует о событии ввода
```

```
id = 0: отслеживать мышь
```

```
1: потребление символов *)
```

```

ControlMsg = RECORD (Display.FrameMsg)
id: INTEGER; (* операция *)
X,Y: INTEGER (* текущее положение мыши *)
END;

(* сигналы управляющего воздействия
id = 0: удалить фокус
1: удалить все метки
2: метка *)

SelectionMsg = RECORD (Display.FrameMsg)
time: LONGINT;
text: Texts.Text;
beg, end: LONGINT
END;

(* сигнализирует о запросе на последнее выделение текста *)

CopyOverMsg = RECORD (Display.FrameMsg)
text: Texts.Text;
beg, end: LONGINT
END;

(* Получение отрезка текста [beg, end] *)

CopyMsgx = RECORD. (Display.FrameMsg)
F: Display.Frame
END;

(* запрос на создание копии текстового фрейма *)

```

Мы помним, что высокоуровневые менеджеры просмотра, такие как MenuViewers, обычно передают большинство полученных сообщений своим подкадрам. В процессе (предварительной) обработки запросов, ориентированных на зрителя, высокоуровневые менеджеры зрителей могут также создавать новые сообщения. В случае с MenuViewers это:

```

ModifyMsgx = RECORD (Display.FrameMsg)
id: INTEGER;          (* операция *)
dY, Y, H: INTEGER    (* вектор dY, новые Y и H *)
END;

(* вертикальное перемещение и расширение или уменьшение рамки внизу
id = 0: расширить рамку
id =1: уменьшить рамку *)

```

dY представляет собой вертикальный вектор перевода, показывающий вверх в случае расширения и вниз в случае уменьшения. По определению, dY всегда неотрицателен. Y и H задают новое положение и высоту соответственно.

В целом, вьювер и его подкадры должны обрабатывать одни и те же сообщения, за исключением того, что некоторые из сообщений обрабатываются или препроцессируются вьювером-предком, что, в свою

очередь, может привести к отправке новых и более тонких сообщений в его подкадры.

Сообщение посылается конкретному объекту простым вызовом установленного обработчика. Например, `F.handle(F, M)` имеет эффект отправки сообщения `M` фрейму `F`. В случае текстовых фреймов установленным обработчиком является `Handle`. Он подробно описано в модуле `TextFrames`. Сейчас мы приведем учебный набросок этой процедуры. Обратите внимание, что `Handle` широко использует возможности Oberon по проверке типов (и защите типов) для того, чтобы различать типы сообщений.

```
1 PROCEDURE Handle (F: Display.Frame; VAR M: Display. FrameMsg);
2 VAR F1: Frame;
3 BEGIN
4 WITH F: Frame DO
5 IF MIS Oberon.InputMsg THEN
6 WITH M: Oberon.InputMsg DO
7 IF M.id = Oberon.track THEN Edit(F, M.X, M.Y, M keys)
8 ELSIF M.id = Oberon.consume THEN =
9 IF F.car # 0 THEN Write(F, M.ch, M.fnt, M.col, M.off) END
12 END
13 END
14 ELSIF M IS Oberon.ControlMsg THEN
15 WITH M: Oberon.ControlMsg DO
16 IF M.id = Oberon.defocus THEN Defocus(F)
17 ELSIF M.id = Oberon.neutralize THEN Neutralize(F)
18 END
19 END
20 ELSIF M IS Oberon.SelectionMsg THEN
21 WITH M: Oberon.SelectionMsg DO GetSelection(F, M.text, M.beg, M.end, M.time)
    END
22 ELSIF M IS Oberon.CopyOverMsg THEN
23 WITH M: Oberon.CopyOverMsg DO CopyOver(F, M.text, M.beg, M.end) END
24 ELSIF M IS Oberon.CopyMsg THEN
25 WITH M: Oberon.CopyMsg DO Copy(F, 1); M.F:=F1 END
26 ELSIF M IS MenuViewers.ModifyMsg THEN
27 WITH M: MenuViewers.ModifyMsg DO Modify(F, M.id, M.dY, M.Y, M.H) END
28 ELSIF MIS UpdateMsg THEN
29 WITH M: UpdateMsg DO
30 IF F.text = M.text THEN Update(F, M) END
31 END
32 END
33 END
34 END Handle;
```

Пояснения:

1 процедура типа `Display`. Обработчик

```

2  вспомогательная переменная для копирования кадра (строка 25). Необходимый
    базовый тип potas M.Fis
4  защита типа; F должен быть текстовым фреймом
5  type test; является ли сообщение входным сообщением Oberon?
6  защита типа
7  если сообщение требует отслеживания мыши
8  если сообщение требует потребления, то потребляет символ
9  если каретка активна в данном текстовом фрейме
14 type test; является ли сообщение управляющим сообщением Oberon?
15 защита типа
16 если сообщение требует убрать каретку
17 если сообщение требует удаления каретки и выделения
20 проверка типа; является ли сообщение запросом выбора Oberon?
21 если да, то зарегистрируйте собственный выбор, если он новее кандидата.
22 проверка типа; является ли сообщение копией над сообщением?
23 если да, скопируйте текст, растягивая его на место каретки в этом кадре
24 тест типа; является ли сообщение копией сообщения?
25 если да, создайте копию этого фрейма (инициализированную пустой)
26 type test; is message a modify-message?
27 если да, измените размер или положение этого кадра (см. ниже)
28 type test; is message an update message?
30 если да, проверьте, представлен ли измененный текст в этом фрейме, а затем
    обновите его.

```

Мы также предоставляем следующее усовершенствование процедуры TextFrames.Modify, вызываемой в строке 27 в файле Textframes.Handle. Она хорошо показывает, как должны реагировать подфреймы меню-просмотрщиков на сообщение MenuViewers.ModifyMsg.

```

PROCEDURE Modify (F: Frame; id, dY, Y, H: INTEGER);
BEGIN
Mark(F, 0); RemoveMarks(F);      (* удаляет позицию-бар, каретку и выделение *)
IF id = MenuViewers.extend THEN
    IF dY > 0 THEN                (* если рамка должна быть перемещена *)
        Display.CopyBlock(F.X, EY, EW, FH, FX, FY + dY, 0); FY := FY + dY (*
        перемещение исходного блока *)
    END;
    Extend(F, Y)                  (* расширяем перемещенный кадр в нижней части *)
ELSIF id = MenuViewers.reduce THEN
    Reduce(F, Y + dY);            (* уменьшить исходный кадр в нижней части *)
    IF dY > 0 THEN                (* если кадр должен быть перемещен *)
        Display.CopyBlock(F.X, F.Y, FW, F.H, FX, Y, 0); F.Y := Y (* уменьшенный
        блок *)
    END
END;
IF F.H > 0 THEN Mark(F,1) END    (* восстановить позицию-бар *)
END Modify;

```

Помните, что `dY` всегда неотрицательно, и обратите внимание, что часть процедуры `reduce` является точной обратной части `extend`. Заметьте, кстати, что приведенный выше обработчик используется для обработки как данных-фреймов, так и стандартных заголовков. Эти два варианта текстовых фреймов отличаются только цветом фона. Этот факт свидетельствует о упрощенном дизайне системы и является примером "модного сегодня" понятия повторного использования кода.

Внимательный и опытный читатель мог заметить, что модуль `TextFrames` играет две очень разные роли. Первая его роль – обработчик сообщений, посылаемых экземплярам фреймов (с поздней привязкой). Вторая роль библиотеки процедур, работающих с текстовыми фреймами. Примерами являются `Edit`, `Write`, `CopyOverShow`; `SetCaret` и `TrackWord`. Две роли текстовых фреймов соответствуют двум различным способам обращения с текстовыми фреймами, а именно, как к активным объектам, индивидуально реагирующим на сообщения, и как к пассивным прямоугольникам, с которыми можно работать обычным образом, вызывая процедуры. Второй способ обращения с фреймами может быть важен в случаях, когда текстовые рамки представляют собой текстовые поля, относящиеся к содержимому какого-либо более сложного документа.

Модуль `TextFrames` предлагает еще один выбор, на этот раз потенциальным реализаторам обработчиков подклассов текстовых фреймов: Либо они реализуют свой собственный обработчик, просто добавляя инкременты, и затем ссылаются на `Handle`, либо они составляют индивидуальный обработчик из элементов. Например, разработчик `MailFrames` может применить первую стратегию. Он может просто добавить несколько методов, ориентированных на почту, перехватывающих ориентированные на почту сообщения, а затем делегировать всю дальнейшую обработку `TextFrames.Handle`. В отличие от этого, разработчик нового пользовательского интерфейса для текстовых фреймов могут предпочесть стратегию 2.

Мы завершаем этот раздел кратким объяснением потока управления после того, как, например, был набран символ. Мы предполагаем, что вьювер является программой просмотра текста и что каретка установлена в ее основном фрейме данных. Первое, центральный цикл Oberon обнаруживает новый символ в буфере клавиатуры. Затем он посылает входное `consume-message` в фокус-вьювер, который, в свою очередь, передает это сообщение своему главному подфрейму. После этого обработчик (интерпретатор команд) этого субфрейма определяет местоположение каретки в основном тексте и затем вызывает процедуру `ninsert` менеджера текстовых данных `Texts`. После этого `Insert` вставляет символ в текст и (`up-`)вызывает связанный с ним нотификатор, который в нашем случае находится в `TextFrames`. Нотификатор затем посылает сообщение типа `TextFrames.UpdateIsig` всем видимым вьюверам. Каждый из этих вьюверов передает это сообщение своим подфреймам. Если подрамка понимает сообщение и считает себя вовлеченным в это обновление, он восстанавливает свое содержимое, обращаясь к новым данным, опять же из менеджера данных `Texts`.

Теперь мы представим несколько типичных примеров реализации.

Учебный пример 2: Операции, ориентированные на фрейм

Отображение текстовой строки в текстовом фрейме

```
PROCEDURE DisplayLine (F: Frame; L: Line; VAR R: Texts.Reader; X, Y: INTEGER;
len: LONGINT);

  VAR pat: Display.Pattern; NX, dx, x, y, w, h: INTEGER;
BEGIN NX := F.X + F.W;

  WHILE (nextCh # CarriageReturn) & (R.fnt # NIL) DO
    Display.GetChar(R.fnt.raster, nextCh, dx, x, y, w, h, pat);
    IF (X + x + w <= NX) & (h # 0) THEN
      Display.CopyPattern(R.col, pat, X + x, Y +, 2)
    END;
    X := X + dx; INC(len); Texts.Read(R, nextCh)
  END;

n+ 1; Lwid i= X + eolW - (FX + F.margW);
NIL; Texts.Read(R, nextCh)
```

где типы Line и Frame определены следующим образом. Обратите внимание, что line является частным типом, а TextFrames.Frame является публичной проекцией типа Frame.

```
Line = POINTER TO LineDesc;
LineDesc = RECORD
  len: LONGINT; (* количество символов в этой строке *)
  wid: INTEGER; (* ширина поля строки *)
  eot: BOOLEAN; (* флаг конца текста *)
  next: Line    (* следующий нижний сосед *)
END;

Location = RECORD
  org, pos: LONGINT; (* начало линии, позиция *)
  dx, x, y: INTEGER; (* ширина и положение расположенного символа *)
  lin: Line          (* ассоциированная строка *)
END;

Frame = POINTER TO FrameDesc;
FrameDesc = RECORD (Display.FrameDesc)
  text: Texts.Text;          (* отображаемый текст *)
  org: LONGINT;              (* позиция в тексте первого отображаемого символа *)
  col: INTEGER;              (* цвет фона *)
  Isp, asr, dsr: INTEGER;    (* межстрочный интервал, восходящий, нисходящий *)
  left, right, top, bot: INTEGER; (* поля *)
  markH: INTEGER;            (* ширина поля, положение метки *)
  time: LONGINT;             (* время последнего выбора *)
  mark, car, sel: INTEGER;   (* состояние метки, каретки, выделения *)
  carloc: Location;          (* местоположение каретки *)
  selbeg, selend:            (* местоположение начала и конца выделения *)
  trailer: Line              (* указатель на связанную последовательность *)
```

дескриптора строки *)

END;

Каретка трека

PROCEDURE TrackCaret (F: Frame; X, Y: INTEGER; **VAR** keysum: SET);

VAR loc: Location; keys: Si

BEGIN

IF F.trailer.next # F.trailer **THEN**

 LocateChar(F, X - FX, Y - FY, F.carloc);

 FlipCaret(F);

 keysum :={};

LOOP

 Input.Mouse (keys, X, Y);

IF keys = {} **THEN EXIT END;**

 keysum := keysum + keys;

 Oberon.DrawCursor(Oberon.Mouse, Oberon.Mouse.marker, X, Y);

 LocateChar(F, X - FX, Y - F.Y, loc);

IF loc.pos # F.carloc.pos **THEN** FlipCaret(F); F.carloc := loc; FlipCaret(F) **END**

END;

Fcar:=1

END

END TrackCaret;

где используются следующие вспомогательные процедуры:

PROCEDURE LocateChar (F: Frame; x, y: INTEGER; **VAR** loc: Location);

VAR R: Texts.Reader;

 pat: Display.Pattern;

 pos, lim: LONGINT;

 ox, dx, u,v, w, h: INTEGER;

BEGIN LocateLine(F, y, loc);

 lim := loc.org + loc.lin.len - 1;

 pos := loc.org; ox := F.left;

 Texts.OpenReader(R, F.text, loc.org); Texts.Read(R, nextCh);

LOOP

IF pos = lim **THEN** dx := eolW; **EXIT END;**

 Display.GetChar(R.fnt raster, nextCh, dx, u,v, w, h; pat);

IF ox + dx > x **THEN EXIT END;**

 INC(pos); ox := ox + dx; Texts.Read(R, nextCh)

END;

loc.pos := pos; loc.dx := dx; loc.x := ox

END LocateChar;

PROCEDURE Locateline (F: Frame; y: INTEGER; **VAR** loc: Location);

VAR T: Texts.Text; L: Line; org: LONGINT; cury: INTEGER;


```

BEGIN F.text;
    org = F.org; L:= F.trailer.next; cy» i= F.H - F.top - Fas;
    WHILE (L.next # F.trailer) & (cury > y + F.dsr) DO
        org := org + Llen; L:= L.next; cury := cury - F.lsp
    END;
loc.org := org; loclin := L; locy = cury
END Locateline;

PROCEDURE FlipCaret (F: Frame);
BEGIN
IF F.carloc.x < FW THEN
IF (F.carlocy >= 10) & (F.carloc.x + 12 < FW) THEN
    Display.CopyPattern(white, BigCaret; F.X + F.carlocx, FY + Farlocy - 10, 2)
ELSIF (F.carlocy >= 4) & (F.carloc.x + 6 < -F.W) THEN
    Display.CopyPattern(white, SmallCaret, F.X + F.carloc.x, FY + F.carlocy - 4, 2)
END
END
END FlipCaret;

```

Учебный пример 3: Операции, ориентированные на работу с текстом

Сохранение текста в буфере

```

PROCEDURE Save (T: Text; beg, end: LONGINT; B: Buffer);
    VAR p, q, gb, qe: Piece; org: LONGINT;
BEGIN #
    IFend > T.len THEN end := T.len END;
    FindPiece(T, beg, org, p);
    NEW(gb); gb := pt;
    gb.len.:=gb.len - (beg - org);
    qb.off + (beg - org);
    qe :=qb;
    WHILE end > org + p.len DO
        org = org + p.len; p := p.next;
    NEW(q); q+ := p#; qe.next :
    END;
    genext = NIL; qe len := ge.len - (org + p.len - end);
    B.last.next := qb; qb.prev := B.last; B.last := qe;
    B.len := B.len + (end - beg)
END Save;

```

где FindPiece - следующая вспомогательная процедура:

```

PROCEDURE FindPiece (T: Text; pos: LONGINT; VAR org: LONGINT; VAR p: Piece);
    VAR n: INTEGER;
BEGIN
    IF pos < T.org THEN T.org := 1; T.pce i= T trailer END;
    org i= T.org; p := T.pce; (*из кэша*)
    n:=0;
    WHILE pos >= org + p.len DO org := org + p.len; p := p.next; INC(n) END;
    IF n > 50 THEN T.org := org; T.pce := p END (*кэшировать*)
END FindPiece;

```

и где типы Piece, Text и Buffer определены следующим образом. Обратите внимание, что Piece является частным типом, Texts.Text. Buffer является публичной частью типа Buffer.

```

Piece = POINTER TO PieceDesc;
PieceDesc = RECORD

```

```

f: Files.File;      (* файлы носителей *)
off: LONGINT;       (* смещение в файле *)
len: LONGINT;       (* длина фрагмента *)
fnt: Fonts.Font;    (* шрифт *)
col: SHORTINT;      (* цвета *)
voff: SHORTINT;     (* вертикальное смещение *)
prev, next: Piece   (* ссылки на соседей *)
END;

```

```

Text = POINTER TO TextDesc;
Notifier = PROCEDURE (Text, INTEGER, LONGINT, LONGINT);
TextDesc = RECORD,
len: LONGINT;      (* длина текста *)
notify: Notifier;   (* вызывается после изменения текста *)
trailer: Piece;     (* дозорный *)
org: LONGINT;       (* кэшированное начало *)
pce: Piece          (* кэшированный фрагмент *)
END;

```

```

Buffer = POINTER TO BufDesc;
BufDesc = RECORD
len: LONGINT;      (* длина буфера *)
header, last: Piece (* буферизованный список фрагментов *)
END;

```

Вставка содержимого буфера в текст

```

PROCEDURE Insert (T: Text; pos: LONGINT; B: Buffer);
    VAR pl, pr, p, gb, qe: Piece; org: LONGINT;
BEGIN

```

```

FindPiece(T, pos, org, p); SplitPiece(p, pos - org, pr);
IF T.org >= org THEN                                (* настроить кэш *)
T.org := org - p.prev.len; T.pce i= p.prev
END;

pl := pr.prev; qb := B.header.next;
IF (gb # NIL) & (gb.f= pl.f) & (gb.off = pl.off + pl.len)
& (gb.fut = pl.fnt) THEN pl.len := pl.len + gb.len; gb := gb.next
END;

IF gb # NIL THEN ge := Blast;
qb.prev := pl; pl.next := gb; qe.next :=pr; pr.prev := ge
END;

T.len := T.len + B.len;
T.notify(T, insert, pos, pos + B.len);                (* вызов постпроцессора *)
B.last := B.header; B.last.next := NIL; B.len := 0
END Insert;

```

где SplitPiece это

```

PROCEDURE SplitPiece (p: Piece; off: LONGINT; VAR pr: Piece);
VAR q: Piece;
BEGIN
IF off > 0 THEN NEW(q);
q.col := p.col; q.fnt := p.fnt;
q.len := p.len-off;
q.f := p.f; q.off := p.off+off;
p.len := off;
q.next:= p.next; p.next := q;
q.prev := p; q.next.prev :=q;
pr := q
ELSE pr := p
END
END SplitPiece;

```

Литература

Рабочая станция Ceres

Н. Эберле. Разработка и анализ рабочей станции.
Дисс. ЕТН № 8431, 1987.

В. Нееб. Проектирование процессорной платы для рабочей станции

Ceres-2,

Bericht 93, Inst. fiir Informatik, ETH Ziirich, ноябрь 1988.

Язык Oberon

N. Вирт, Расширения типов.

ACM Trans. on Prog. Languages and Systems, 10, 2 (апрель 1988), 204-214.

N. Wirth. От Модуля к Оберон.

Программное обеспечение - практика и опыт, 18, 7, (июль 1988), 661-670.

N. Wirth. Язык программирования Oberon.

Программное обеспечение - практика и опыт, 18, 7, (июль 1988), 671-690.

J. Гуткнехт. Вариации на тему роли интерфейсов модулей.

Структурированное программирование, 10,1, (январь 1989), 40-46.

Система Oberon

N: Вирт. Расширяемая система и инструмент программирования для рабочих станций.

Proc. IVth South African Computer Science Symposium. Претория, Южная Африка.

N. Wirth. Oberon: Расширяемая операционная система для рабочих станций.

Proc. Euromicro Conf, Ziirich, 29.8. - 19.1988.

N. Wirth and J. Gutknecht. Система Oberon.

Bericht 88, Inst. fiir Informatik, ETH Ziirich, July 1988;
и Программное обеспечение - практика и опыт, 19(1989).

N. Wirth. Проектирование системы с нуля.

Структурированное программирование, 10,1 (январь 1989), 10-18.

Список номеров ошибок

Неправильное использование языка Oberon-2

- 0 не объявленный идентификатор
- 1 многократно определенный идентификатор
- 2 недопустимый символ в числе
- 3 недопустимый символ в строке
- 4 идентификатор не соответствует имени процедуры
- 5 комментарий не закрыт
- 6
- 7
- 8
- 9 "=" ожидается
- 10 идентификатор ожидается
- 11
- 12 определение типа начинается с неправильного символа
- 13 фактор начинается с неправильного символа
- 14 инструкция начинается с неправильного символа
- 15 объявление, за которым следует неправильный символ
- 16 **MODULE** ожидается
- 17 число ожидается
- 18 "." отсутствует
- 19 "," отсутствует
- 20 ":" отсутствует
- 21
- 22 ")" отсутствует
- 23 "]" отсутствует
- 24 "}" отсутствует
- 25 **OF** отсутствует
- 26 **THEN** отсутствует
- 27 **DO** отсутствует
- 28 **TO** отсутствует
- 29 "(" отсутствует
- 30
- 31
- 32
- 33 "!=" отсутствует
- 34 ", " или **OF** ожидается
- 35
- 36
- 37 идентификатор ожидается
- 38 ";" отсутствует

39
40 **END** отсутствует
41
42
43 **UNTIL** отсутствует
44
45 **EXIT** не внутри цикла
46
47 недопустимо помеченный идентификатор
48 неудовлетворенная forward-ссылка
49 рекурсивный импорт запрещен
50 выражение должно быть константой
51 константа, а не целое число
52 идентификатор не обозначает тип
53 идентификатор не обозначает тип записи
54 тип результата процедуры не является базовым типом
55 процедура вызова функции
56 присвоение не переменным
57 указатель, не привязанный к типу записи или массива
58 определение рекурсивного типа
59 недопустимый параметр открытого массива
60 неправильный тип метки
61 недопустимый тип метки
62 метка определенная более одного раза
63 индекс выходит за границы
64 больше фактических, чем формальных параметров
65 фактических параметров меньше, чем формальных
66 типы элементов фактического массива и формального открытого массива различаются
67 фактический параметр, соответствующий открытому массиву, не является массивом
68
69
70
71
72
73
74
75
76
77
78

79

80 индексное выражение не целое число

81 индекс выходит за указанные границы

82 индексируемая переменная не является массивом

83 неопределенное поле записи

84 разыменованная переменная не является указателем

85 тип защиты или теста не является расширением типа переменной

86 тип защиты или теста не являются указателями

87 охраняемая или проверяемая переменная не является ни указателем, ни записью **VAR**-параметра

88 тип теста не является типом расширения

89 тип теста не является указателем

90 проверяемая переменная не является ни указателем, ни записью **VAR**-параметра

91

92 операнд IN не является INTEGER или не является SET

93 тип элемента SET не является целым числом

94 операнд & не имеет типа BOOLEAN

95 операнд OR не имеет типа BOOLEAN

96 операнд неприменимый к (унарному) +

97 операнд неприменимый к (унарному) -

98 операнд ~ не имеет типа BOOLEAN

99

100 несовместимые операнды двухадресного оператора

101 тип операнда, неприменимый к *

102 тип операнда, неприменимый к /

103 тип операнда неприменим к DIV

104 тип операнда неприменим к MOD

105 тип операнда неприменим к +

106 тип операнда неприменим к -

107 тип операнда неприменим к = или #

108 тип операнда неприменимый к отношению

109

110 операнд не является типом

111 операнд неприменим к (этой) функции

112 операнд не является переменной

113 несовместимое назначение

114 строка слишком длинная

115 несоответствие параметра между типом переменной (или forward-процедурой) и данной процедурой

116 тип переменной (или forward процедуры) имеет больше параметров, чем эта процедура

117 тип переменной (или forward процедуры) имеет меньше

- параметров, чем эта процедура
- 118 тип результата переменной процедуры (или forward-объявления) отличается от типа результата данной процедуры
- 119 назначенная процедура не является глобальной
- 120 тип выражения, следующего за **IF**, **WHILE**, **UNTIL** или **ASSERT**, не является **BOOLEAN**
- 121 вызываемый объект не является процедурой
- 122 фактический **VAR**-параметр не является переменной
- 123 тип фактического параметра не идентичен типу формального **VAR**-параметра
- 124 тип выражения результата отличается от типа процедуры
- 125 тип выражения **case** не является ни **INTEGER**, ни **CHAR**
- 126 недопустимый режим операнда
- 127 недопустимый режим адресованного операнда
- 128
- 129 тело объявленной процедуры отсутствует
- 130 **WITH** предложение не указывает переменную
- 131 **LEN** не применяется к массиву
- 132 размер в **LEN** слишком большой или отрицательный
-
- 150 несоответствие ключа импортированного модуля
- 151 неверный файл символов
- 152 символьный файл импортированного модуля не найден
- 153 объектный или символьный файл не открыт (диск заполнен?)
- 154 экспортируемый тип относится к не экспортируемому компоненту
- 155 создание нового файла символов не разрешено

Ограничения реализации

- 200 еще не реализовано
- 201
- 202 заданный элемент больше **MAX(SET)** или меньше 0
- 203 слишком большое число
- 204
- 205 деление на ноль
- 206
- 207
- 208 слишком много переменных в процедуре
- 209 требуется слишком много пространства переменной
- 210

- 211 слишком большое расстояние перехода
- 212 буфер для идентификаторов и строк полный
- 213 слишком много case-ов в операторе case
- 214 слишком много операторов выхода
- 215 недостаточно регистров: упрощение выражения
- 216 недостаточно регистров с плавающей запятой: упрощение выражения
- 217 параметр должен быть целой константой
- 218 недопустимое значение параметра ($20 \leq p \leq 256$)
- 219 недопустимое значение параметра ($0 \leq p < 16$)
- 220 слишком много указателей в записи
- 221 строка не может быть экспортирована
- 222 слишком много указателей (либо глобально, либо в записи)
- 223 слишком много типов записей
- 224 слишком много типов указателей
- 225 адрес переменной указателя слишком велик (перемещение вперед по тексту)
- 226 слишком много экспортированных процедур (>40)
- 227 слишком много импортированных модулей
- 228 слишком много экспортируемых структур
- 229 слишком много вложенных записей для импорта
- 230 слишком много констант (строк) в модуле
- 231 слишком много ссылок таблицы записей (внешние процедуры)
- 232 слишком много команд в модуле
- 233 иерархия расширений записей слишком высока

Номера ловушек времени выполнения

- 1 ошибка четности (NMI)
- 2 недопустимая ссылка (NIL-ссылка, inspect EIA)
- 3 ошибка FPU (inspect FSR)
- 4 недопустимая инструкция
- 5 недопустимый номер SVC
- 6 деление на ноль
- 7 ловушка флага, недопустимый индекс, целочисленное переполнение
- 8
- 9 трассирующая ловушка
- 10 неопределенная инструкция
- 11 ошибка шины с возможностью перезапуска
- 12 ошибка шины без возможности перезапуска
- 13 ловушка целочисленного переполнения

14 ловушка отладчика

15

16 недопустимая инструкция в CASE-операторе

17 функциональная процедура без инструкции RETURN

18 проверка типа guard

19 проверка типа guard в назначении записи

20 ошибка диска (не читаемый сектор)

21 неверный адрес сектора

22 разрыв с клавиатурой

23 файл слишком большой (>2.5 МБ)

24 диск полный

25 файл отсутствует для отложенной загрузки

26

27 недопустимый аргумент функции (Math или MathL)

28

29

30-255 запрограммированный HALT