MiniM Database Server
Расширенное руководство

Версия 1.29

Евгений Каратаев
mailto:support@minimdb.com
http://www.minimdb.com

20 мая 2018 г.

Оглавление

Глава 1

Администрирование и настройки

1.1 Описание сервиса MiniM

1.1.1 Windows версия

Сервис обеспечивает работу выбранной инсталляции (или экземпляра) MiniM.

Инсталляция

Запустить `mnmsvc.exe` с ключом `/install`. Ключ нечувствителен к регистру. При инсталляции появится диалоговое окно, индицирующее успешность инсталляции. Если его появления не требуется, то нужно добавить в командную строку ключ `/silent`. Этот ключ также нечувствителен к регистру.

Инсталляция сервиса `mnmsvc.exe /install /silent` рекомендуется для автоматической инсталляции. При инсталляции проверяется значение `InstallName` в файле `minim.ini` в секции `Server`. При конфигурировании сервера MiniM следует обращать внимание на то, чтобы имена инсталляций различались.

Пример инсталляции:

```
w:\MiniM\bin\mnmsvc.exe /install
w:\MiniM\bin\mnmsvc.exe /install /silent
```

Деинсталляция

Запустить `mnmsvc.exe` с ключом `/uninstall`. Ключ нечувствителен к регистру. При деинсталляции появится диалоговое окно, индицирующее успешность деинсталляции. Если его появления не требуется, то нужно добавить в командную строку ключ `/silent`. Этот ключ также нечувствителен к регистру. При попытке деинсталляции сервиса, если он не был установлен, ключ `/silent` игнорируется, и выводится сообщение об ошибке.

Деинсталляция сервиса `mnmsvc.exe /uninstall /silent` рекомендуется для автоматической деинсталляции.

Пример деинсталляции:

```
w:\MiniM\bin\mnmsvc.exe /uninstall
w:\MiniM\bin\mnmsvc.exe /uninstall /silent
```

Работа

После инсталляции сервиса он по умолчанию работает под аккаунтом LocalSystem, запускается вручную. Для того чтобы сервер MiniM запускался при запуске компьютера автоматически, следует в списке служб найти соответствующий экземпляр сервиса, имеющий вид «MiniM Service for INSTALLNAME» и установить этому сервису опцию старта автоматически. Если на компьютере установлено несколько экземпляров MiniM, то эта опция может быть установлена сервисом, обслуживающим несколько инсталляций MiniM.

При этом различные инсталляции могут быть различных версий. При конфигурировании инсталляций следует им в файле minim.ini указать различные порты для телнет сервера и для других значений, которые могут быть использованы этими инсталляциями одновременно.

Запуск и останов службы из командной строки.

Для запуска службы из командной строки используйте команду net с ключом start:

`net start "MiniM Service for MINIM00"`,

где `MINIM00` - имя инсталляции MiniM.

Для останова сервиса из командной строки используйте команду net с ключом stop:

`net stop "MiniM Service for MINIM00"`

Эти команды должны привести соответственно к запуску и останову сервиса. Если в рамках данной инсталляции работали какие-либо процессы, то они будут принудительно остановлены.

Вариант запуска и останова сервиса из командной строки приведён ниже:

```
W:\MiniM\bin>net start "MiniM Service for MINIM00"
Служба "MiniM Service for MINIM00" запускается.
Служба "MiniM Service for MINIM00" успешно запущена.

W:\MiniM\bin>net stop "MiniM Service for MINIM00"
Служба "MiniM Service for MINIM00" останавливается.
Служба "MiniM Service for MINIM00" успешно остановлена.
```

Для проверки, запущен ли сервис, нужно обратиться к менеджеру сервисов в административных утилитах. Другой способ - иметь запущенную программу minimti.exe, которая индицирует активность инсталляции иконкой в трее.

1.1.2 Linux версия

Инсталлятор сервера MiniM для Linux совмещен с деинсталлятором и один и тот же исполняемый файл используется как для инсталляции, так и для деинсталляции. Отличие режима заключается в параметре -u. Инсталлятор работает исключительно как консольная программа командной строки и не использует редкие клавишные комбинации, зависимые от терминала. Поэтому инсталляция и деинсталляция могут выполняться как локально, так и удаленно.

Архитектура MiniM Database Server допускает установку на один компьютер нескольких экземпляров MiniM, в том числе различных версий одновременно. Они работают независимо друг от друга.

Инсталляция

Установить флаг исполняемого файла для файла инсталлятора

```
chmod +x setupminim...
```

Имя исполняемого файла инсталлятора может отличаться в различных версиях номером версии и названием целевого процессора.

Запустить на выполнение инсталлятор

```
sudo ./setupminim...
```

Деинсталляция

Установить флаг исполняемого файла для файла инсталлятора той же версии сервера, которую необходимо удалить.

```
chmod +x setupminim...
```

Имя исполняемого файла инсталлятора может отличаться в различных версиях номером версии и названием целевого процессора.

Запустить на выполнение деинсталлятор с параметром

```
sudo ./setupminim... -u
```

В случае если текущий пользователь имеет недостаточно прав для запуска инсталлятора или деинсталлятора, инсталлятор сообщает об этом.

Запуск и останов демона из командной строки.

Для запуска, останова и перезапуска демона сервера MiniM используются скрипты в подкаталоге /bin инсталляции сервера:

```
start.sh
stop.sh
restart.sh
```

В случае каких-либо изменений в способе запуска в зависимости от версии MiniM, эти изменения должны ожидаться в этих файлах скриптов.

При необходимости автоматического запуска сервера при запуске компьютера необходимо настроить запуск сервера, используя скрипт

```
start.sh
```

Все внутренние логи сервер ведет в подкаталоге /bin, файл minim.log.

1.2 Описание файла minim.ini

1.2.1 Секция Server

Ключ **InstallName** - имя инсталляции. Каждая инсталляция сервера должна иметь уникальное на данном компьютере имя. Рекомендуется использовать имена, производные от MINIM и содержащие только латинские буквы или цифры. Это имя также будет использовано сервисом mnmsvc.exe, и по нему можно ориентироваться в списке имеющихся сервисов. Длина имени инсталляции - до 31 символов. Если указано бóльшее количество, будут использованы только первые 31 символ. При

автоматическом преобразовании имени инсталляции символы, не являющиеся латинскими буквами или цифрами, будут заменены на букву M. Пример:

InstallName = MINIM00

Ключ **LogFileName** - имя или полное имя файла лога с аварийными и диагностическими сообщениями сервера. Значение по умолчанию - minim.log, размещение в каталоге bin инсталляции сервера. При указании некорректного имени файла используется имя по умолчанию. Файл лога может быть удален, скопирован без нарушения работы сервера и его данных. В этом файле делаются регламентные записи, наличие которых не требуется для восстановления работы сервера, но которые могут оказаться полезны для выяснения причин возможных отказов. Имя ключа нечувствительно к регистру. Пример:

LogFileName = w:/minim/bin/minim.log

Ключ **LogFileChunk** - размер файла лога.

При превышении его размера файл лога minim.log сохраняется в файл minim.log.bak, и, последующие записи лога, ведутся в файле minim.log. Значение указывается в байтах. Если значение не указано, используется 100 килобайт. Имя ключа нечувствительно к регистру.

Файл лога может быть удален, скопирован без нарушения работы сервера и его данных. В этом файле делаются регламентные записи, наличие которых не требуется для восстановления работы сервера, но которые могут оказаться полезны для выяснения причин возможных отказов. При переключении лога на новый файл старый файл minim.log.bak замещается. Это сделано с целью недопущения неконтролируемого перерасхода дискового пространства. В случае если требуется сохранять логи сервера за более длительное время, следует периодически сохранять логи отдельно или увеличить величину LogFileChunk. Не рекомендуется указывать малые значения, поскольку это может привести к быстрой потере информации, которая может оказаться важной. Пример:

LogFileChunk = 100000

Ключ **BIJFileName** - имя или полное имя файла предзаписи. В нем сервер записывает страницы файлов данных перед записью в сами файлы с целью обеспечить защиту целостности файлов данных на физическом уровне. Файл предзаписи страниц используется при восстановлении файлов данных при аварийном останове сервера. Имя ключа нечувствительно к регистру. Пример:

BIJFileName = w:/minim/bin/minim.bij

Ключ **ProcessLimit** - количество процессов, которые может запустить

сервер одновременно. Если значение не указано, используется величина 2000. Величина ProcessLimit не относится к лицензионному ограничению - ее можно указать сколь угодно большой. Эту величину следует контролировать в целях предотвращения рекурсивных запусков процессов. Увеличение значения не приводит к реальному повышению расхода ресурсов. Пример:

ProcessLimit = 1000

Ключ **Locale** - имя локали, используемое инсталляцией. Имя указывает файл с таблицей сравнения символов, таблицей поднятия и таблицей опускания регистра. Используется файл с указанным именем из каталога /nat и имеющий расширение N. Значение по умолчанию - RUS, что соответствует кодировке Windows-1251. Эти таблицы используются для всех баз данных инсталляции в операциях сравнения ключей и в операторе «сортируется после». Пример:

Locale = RUS

MiniMono difference

Для MiniM Embedded Edition имя файла локали указывается хост програмой, файл может находиться в любом каталоге.

Ключ **LockAreaSize** - количество памяти в мегабайтах для таблицы блокировок. Объем памяти выделяется один раз при старте сервера. Для изменения величины требуется перезапуск сервера. Размер выбирается экспериментальным путем в зависимости от потребностей прикладных систем, работающих на сервере, в блокировках. Минимальное количество - 1, максимальное - 64. Если требуется применение еще большей величины, то понадобится спецверсия сервера. Ограничение введено из-за опасений, что при бóльшем количестве сервер может перейти в нестабильное состояние. Этот размер памяти выделяется в области свапа операционной системы. Поэтому, в случае большой величины, рекомендуется соответственно нарастить на сервере размер свапа. Бóльшие величины могут потребовать пересмотра архитектуры сервера. Пример:

LockAreaSize = 8

MiniM x64 difference

В версии MiniM Database Server для архитектуры x64 верхний предел значения не используется, могут быть использованы десятки гигабайт на усмотрение администратора.

Ключ **AutoStartScript** - имя файла, в котором содержатся однострочные команды языка M[UMPS]. Если значение ключа указано, то после

успешного старта сервера запускается процесс выоплняющий эти команды последовательно одна за другой, построчно, как если бы они вводились в консоли оператором. При окончании выполнения каждой строки текущим устройством вода-вывода исполняющего процесса становится устройство по умолчанию. Файл скрипта не является подпрограммой, это только последовательность операторов. Например, если в файле autostart.m указаны строки

```
zn "user"
s ^TRACE($I(^TRACE))="start at "_$h
```

То при указании ключа

```
AutoStartScript = autostart.m
```

после успешного старта сервера будут последовательно исполнены указанные строки. Несмотря на то, что в файле скрипта нельзя использовать точечный синтаксис и объявлять подпрограммы, их можно вызывать, если они предварительно созданы и готовы к выполнению. Имя файла не должно содержать пробелов.

1.2.2 Секция Telnet

Ключ **Start** - запускать или не запускать внутренний телнет-сервер при старте MiniM сервера. Значение для запуска - 1, для незапуска - 0. Если ключ отсутствует или имеет иное значение, то внутренний телнет-сервер не запускается. Пример:

Start = 1

Ключ **Port** - номер TCP/IP порта, который должен использоваться внутренним телнет-сервером данной инсталляции. Допустимые значения - число, разрешённое к использованию в качестве номера порта. Порт по умолчанию для телнет-серверов - 23. При конфигурировании инсталляции следует учесть, работает ли на этом порту внутренний Windows или иной телнет-сервер или внутренний телнет-сервер другой инсталляции MiniM. Следует указать неиспользуемый номер порта. Если ключ отсутствует или не является допустимым номером порта, то внутренний телнет-сервер не запускается. Пример:

Port = 2323

1.2.3 Секция Process

Ключ **DeviceTableSize** - число устройств ввода - вывода, которое может быть открыто одним процессом одновременно. Это ограничение не связано с лицензионными ограничениями и указывается только для предотвращения возможного бесконтрольного расходования ресурсов одним процессом. Увеличение этой величины не приводит к существенному расходованию ресурсов. Рекомендуемое значение - примерно удвоенное число от числа различных используемых процессом устройств. Минимальное значение - 4. Если указано не числовое значение или число, меньшее чем 4, то используется таблица в 4 устройства. Это ограничение относится только к одному процессу и никак не ограничивает общее число устройств, используемое всеми процессами сервера. Пример:

DeviceTableSize = 8

Ключ **DeviceNameSize** - максимальная длина строки, идентифицирующая устройство. Для выбора величины нужно определить характер имен устройств. Большой величины могут потребовать устройства типа запуска программ с передачей параметров командной строки. Обычно, достаточным значением может считаться величина 1024 символа. При создании устройства, которое требует бóльшей величины, оно будет создано и будет корректно работать, но команда use будет иметь неопределённое поведение, если имена устройств имеют одинаковое значение до DeviceNameSize символов. Не следует выбирать эту величину слишком малой. Пример:

DeviceNameSize = 1024

Ключ **Storage** - число мегабайт, резервируемых для локальных переменных процесса. Минимальное значение - 1, максимальное - 64. При указании меньше 1 используется 1, при указании больше 64 используется 64 мегабайт. Пример:

Storage = 8

Ключ **RoutineCache** - число мегабайт, резервируемых для кэша рутин. Минимальное значение - 1, максимальное - 64. При указании меньше 1 используется 1, при указании больше 64 используется 64 мегабайт. Пример:

RoutineCache = 8

Ключ **ReadLineRecallCount** - сколько строк повторного вызова будут запоминать устройства типа TNT (телнет) и CON (консоль). Минимальное число 10, максимальное 128. Память под буфера предыдущего ввода резервируется однократно на все время жизни процесса. Пример:

ReadLineRecallCount = 10

Ключ **ReadLineRecallBuffer** - сколько символов резервируется на строку повторного ввода для устройств типа TNT (телнет) и CON (консоль). Минимальное число 80, максимальное 2048. Память под буфера предыдущего ввода резервируется однократно на все время жизни процесса. Пример:

ReadLineRecallBuffer = 1024

Ключ **FrameCount** - число стековых фреймов которые отводятся процессу в штуках. Память под фреймы аллокируется однократно при старте процесса и не может быть изменена при работе сервера или процесса без перезапуска сервера. Для каждого вызова функции, команд xecute и команд new в области их действия используется по одному стековому фрейму. Минимальное число 16, максимальное 131072. Рекомендуется выставлять это значение с некоторым запасом относительно ожидаемой величины. Пример:

FrameCount = 1024

Ключ **Namespace** - имя области, в которой по умолчанию должен стартовать процесс, если не указано иное специально. Это указание действует на консольные и телнет-процессы, а также на фоновые процессы, запускаемые сервисом, если для них не было указано иное. Если значение не указано, то область по умолчанию выбирается сервером самостоятельно в следующем порядке: USER, %SYS. Если область USER не смонтирована или отсутствует, то используется область %SYS. Области описываются в файле minimdb.ini. Имя области нечувствительно к регистру. Пример:

Namespace = MAPSALES

Ключ **DBCacheSize** - количество байт отводимое на кеширование страниц для всего экземпляра сервера. Память используется всеми процессами. Рекомендуется указывать размер, не превышающий размер оперативной памяти, иначе эта величина теряет смысл. Минимальное значение - 1 мегабайт (1048576), максимальное - 1 гигабайт (1073741824). Пример:

DBCacheSize = 104857600

Здесь на кеширование страниц отводится 100 мегабайт. Следует помнить что 1 килобайт это 1024 байт, 1 мегабайт это 1048576 байт, 1 гигабайт это 1073741824 байт.

Кеширование страниц используется не только всеми процессами, но и для всех файлов данных, обслуживаемых текущим экземпляром сервера.

MiniM x64 difference

В версии MiniM Database Server для архитектуры x64 верхний предел значения не используется, могут быть использованы десятки гигабайт на усмотрение администратора.

Ключ **NullSubscripts** - разрешено ли использование пустой строки в качестве значений индексов. Не разрешено - 0 (по умолчанию), разрешено - 1. Пример:

NullSubscripts = 0

Настройка NullSubscripts действует на вновь создаваемые локальные и глобальные переменные. Имеющиеся в глобалах данные с пустым индексом читаются. Имя ключа нечувствительно к регистру.

Ключ **TrapOnEof** - определяет поведение процесса при попытке чтения за пределами файла, если текущее устройство дисковый файл. Значение 0 - не генерировать ошибку <ENDOFFILE>, а использовать системную переменную $zeof. Значение 1 - генерировать ошибку <ENDOFFILE>. Значение по умолчанию - 1. Пример:

TrapOnEof = 1

Ключ **OnHalt** - определяет действие процесса при завершении работы в отношении его открытых транзакций: откат транзакции, коммит транзакции или ничего не делать. Допустимые значения:

Commit	Подтвержить транзакции на каждом открытом уровне транзакций
Rollback	Откатить транзакции до самого верхнего уровня
Иное	Ничего не делать с откатом или подтверждением транзакций

Значения используются нечувствительно к регистру. Если ничего не указано, то процесс не откатывает и не завершает транзакции. Это поведение по умолчанию.

При планировании поведения процесса при halt следует учитывать, то в случае, если процесс заканчивается по каким-либо причинам (внешние, сбой разделяемых библиотек), то автоматически срабатывает гвардиан процесса, и вызывает откат незаконченной транзакции и снятие всех блокировок, установленных защищаемым им процессом.

1.2.4 Секция Journal

Ключ **JournalDir** - каталог, в котором сервер должен держать файлы журналов. Значение по умолчанию - подкаталог journal каталога инстал-

ляции. В каталоге должно быть достаточно места для роста журнала, за расходуемым объемом требуется следить. Имя ключа нечувствительно к регистру. Пример:

JournalDir = w:/minim/journal

Ключ **JournalFileLimit** - размер текущего файла журнала, после достижения которого сервер автоматически переключает файл журнала на новый с автоматически создаваемым именем. Значение задается как число, в мегабайтах. Минимальное значение - 1, максимальное - 1024. Если значение не указано, то считается по умолчанию 1024 мегабайт. Пример:

JournalFileLimit = 200

Ключ **JournalCache** - размер внутреннего кэша сервера для операций журналирования. Размер указывается в мегабайтах. Значение по умолчанию 8. Минимальная величина - 1, максимальная - 64 мегабайта. Имя ключа нечувствительно к регистру. Пример:

JournalCache = 8

MiniM x64 difference

В версии MiniM Database Server для архитектуры x64 верхний предел значения не используется, могут быть использованы десятки гигабайт на усмотрение администратора.

Ключ **TransactLevelLimit** - максимальное допустимое число вложенных транзакций. Минимальная величина - 1, максимальная - 32000. Если не указано, используется значение по умолчанию - 255. Пример:

TransactLevelLimit = 255

1.2.5 Секция Mnemonic

Секция описывает назначение рутин мнемоник для устройств ввода-вывода.

Ключ **CON** - значение указывает имя рутины, которая автоматически назначается рутиной мнемоник для устройства типа CON (консоль). Если значение не указано или указана пустая строка, то мнемоника автоматически не назначается. Существование рутины с этим именем не требуется и при назначении не проверяется. Но она требуется в скомпилированной форме при использовании мнемоники. Пример:

CON = %CONX364

Ключ **TNT** - значение указывает имя рутины, которая автоматически назначается рутиной мнемоник для устройства типа TNT (telnet).

Если значение не указано, или указана пустая строка, то мнемоника автоматически не назначается. Существование рутины с этим именем не требуется и при назначении не проверяется. Но она требуется в скомпилированной форме при использовании мнемоники. Пример:

TNT = %TNTX364

1.2.6 Секция Login

Секция Login указывает использовать ли при запуске процессов автоматически исполняемый код. Настройки применяются к процессам запускаемым для обслуживания telnet клиентов и при запуске консоли в интерактивном режиме. Если не указано ничего, то процессы сразу переходят в интерактивный режим ввода. Если указаны команды то они выполняются и после их выполнения процессы переходят в интерактивный режим ввода. Чтобы отказать пользователю в обслуживании, требуется, чтобы при исполнении команд была вызвана команда *halt*. Перед выполнением строки команд процесс переключается в базу данных, указанную в секции **[Process]**, ключ **Namespace**.

Ключ **TNT** указывает, какие команды требуется выполнить при обслуживании телнет-коннекта.

Ключ **CON** указывает, какие команды требуется выполнить при обслуживании входа через консоль.

Примеры:

```
[Login]
TNT = d login^%login()
CON = d login^%login()
```

В последовательности команд можно выполнить, например, запрос имени пользователя, пароля, определить хост, с которого он подключился, выполнить смену текущей базы данных, запустить соответствующую ему программу. Если по окончании этой программы не вызвана команда *halt*, то выполнение переходит к обычному интерактивному режиму.

Ключ **GUI** указывает выражение, которое следует выполнить для проверки пароля для указанного пользователем логина. Если строка пустая, то программы GUI (MiniM Control Center, MiniM Global Editor и MiniM Routine Editor) не запрашивают проверки логина и пароля и продолжают работу.

Если строка не пустая, то рассматривается как вычисляемое выражение, которое должно вернуть 0 или не 0. Перед вычислением выражения утилиты переключаются в базу данных указанную в параметрах соединения и записывают в локальные переменные %username и %password значения, введенные пользователем. После вычисления выражения локальная переменная %password удаляется. В комплект MiniM включается рутина %logon, которая может быть использована как один из вариантов. Изменять рутину %logon не рекомендуется, поскольку в последующих версиях она может быть изменена инсталлятором. Пример включения проверки пароля для GUI утилит:

```
[Login]
GUI = $$VALID^%logon(%username,%password)
```

Если выражение вычислено как 0, то утилиты сообщают о неуспехе проверки пароля. Если пользователь отказывается проверить пароль, то утилиты заканчивают работу. Если выражение вычислено как не 0, то утилиты продолжают работу.

Используя описанные соглашения вызова, администратор сервера MiniM может настроить проверку паролей пользователей в соответствии с применяемыми в прикладной системе правами пользователей.

Значение ключей **TNT**, **GUI** и **CON** после инсталляции - пустая строка. При инсталляции поверх имеющейся инсталляции инсталлятор не заменяет значения этих ключей. Значения ключей читаются при необходимости и могут быть изменены при работе сервера без его перезапуска.

1.3 Описание файла minimdb.ini

В файле настроек minimdb.ini хранится описание конфигурации баз данных. Имена секций файла рассматриваются как имена баз данных. Например, секции [app] и [doc] задают базы данных APP и DOC. Имена секций нечувствительны к регистру. Имена секций должны быть корректным именем базы данных, начинаться на латинскую букву или символ процент и далее содержать либо латинские буквы либо цифры. Предел имени базы данных 31 символ.

Сервер MiniM поддерживает внутренний механизм отображения глобалов и рутин между базами данных. Рутины с именами начинающимися на символ процент и глобалы начинающиеся на символ процент физически хранятся в базе данных %sys и логически доступны из всех баз

данных. Глобалы начинающиеся на символы mtemp физически хранятся в базе данных TEMP и логически доступны всем базам данных. Это следует учитывать при конфигурировании баз - базы данных %sys и temp всегда должны существовать и быть смонтированы.

В секции описания базы данных ключи задают параметры базы данных. При указании неподдерживаемого ключа вы файл minim.log выводится диагностическое сообщение. Ключи нечувствительны к регистру.

Ключ **root** задает имя корневого файла базы данных. База данных может состоять их корневого файла и необязательной последовательности дополнительных экстентов. Минимальным составом является один корневой файл. Имя файла должно быть указано таким образом, чтобы этот файл был доступен всем процессам входящим в сервер MiniM и работающим под всеми назначенными им аккаунтами. Администратор должен разместить файлы и настроить аккаунт сервиса так, чтобы при использовании сетевых дисков и дисков, полученных командой subst, эти файлы данных были доступны. Пример:

```
root = e:\minim\db\user.dat
```

Ключ **SizeLimit** задает предел роста корневого файла в мегабайтах. При достижении этого предела рост базы не выполняется, либо, если указаны экстенты, начинается рост экстентов. Пример ограничения корневого файла на 2000 мегабайт:

```
SizeLimit = 2000
```

Если ключ SizeLimit не указан, то корневой файл будет расти неограниченно.

Ключ **AutoExpand** задает выполнять ли расширение файлов базы данных автоматически (значение 1) или нет (значение 0). Если указано что не выполнять, то при работе база данных не будет расширяться. При необходимости можно выполнить расширение административно с использованием утилиты ^%DBSIZE. Расширение файлов баз данных в автоматическом режиме выполняет демон расширения minimed.exe. Расширение файлов данных производится пакетами блоков. Один блок имеет размер 8 килобайт, один пакет блоков - это 128 блоков, общий размер пакета 1 мегабайт.

Ключ **Mount** указывает, использовать (значение 1) эту базы при старте или нет (значение 0). База данных может быть сконфигурирована

но на какой-то период не использоваться. В случае если указан ключ Mount=0, база данных при работе недоступна.

Ключ **Readonly** задает является ли база данных доступной только на чтение (значение 1) или в нее можно вносить изменения (значение 0). При попытке записи в базу описанную только на чтение процесс генерирует ошибку <DBREADONLY>.

Ключ **AutoCreate** задает, необходимо ли (значение 1) или нет (значение 0) автоматически создавать базу данных при старте сервера. Этот ключ устанавливается при инсталляции для базы данных TEMP. И при старте сервера эта база всегда имеет размер 1 мегабайт и не содержит данных.

Ключ **Journal** задает, необходимо ли (значение 1) или нет (значение 0) выполнять журналирование изменений в этой базе данных. Если ключ не указан, то используется значение по умолчанию 1. При отключении журналирования сделанные в базе изменения не могут быть отменены командой trollback и при восстановлении из бекапа с накатом по журналу изменения не накатятся, поскольку в журнале их нет. При инсталляции сервера журналирование отключено для базы данных TEMP.

Ключ **extentNNN** задает имя файла экстента номер NNN. Номера экстентов должны начинаться на 1 и следовать по порядку (1, 2, 3, ...). Например:

```
extent1 = e:\minim\db\user1.ext
```

Ключ **SizeLimitNNN** задает предел роста экстента с номером NNN в мегабайтах. Ключи SizeLimitNNN должны соответствовать ключам extentNNN. Если ключ SizeLimitNNN не указан, то этот экстент будет расти неограниченно.

При конфигурировании экстентов вручную начальное состояние файла экстента должно быть нулевой длины. Его состояние автоматически изменяется и наращивается пакетами блоков сервером MiniM. Следует обращать внимание на то, что номера экстентов должны следовать по порядку. Последний файл в цепочке может не иметь ограничения на рост. В случае если это только один корневой, то он может не иметь ограничения. Иначе все файлы входящие в базу данных кроме последнего экстента должны иметь ограничение на рост. Рост и заполнение блоков сервером выполняется последовательно. Новые блоки не занимаются, если есть свободные среди уже имеющихся.

Основным назначением экстентов является использование дискового пространства разных логических дисков одной базой данных. При бекапе выполняется бекап блоков, заданных последовательностью файлов на момент бекапа, восстановление - в последовательность файлов на момент восстановления. Экстенты являются логическим продолжением корневого файла и предыдущих экстентов.

1.4 Описание файла minimti.ini

Файл minimti.ini и программа minimti.exe используются только в Windows версиях MiniM Database Server. Unix-like версии используют MiniM Launcher, оконную программу для запуска и останова сервера и для запуска служебных утилит.

В файле настроек minimti.ini хранятся настройки для программы minimti.exe. Это программа, которая показывает иконку в системном трее и при вызове нажатием правой кнопки мыши показывает меню запуска, останова и перезапуска сервера и меню дополнительных служебных программ.

Всплывающее из трея меню имеет фиксированную часть и настраиваемую для запуска служебных программ. Настраиваемая часть читается программой minimti.exe при старте из файла minimti.ini.

Используется только секция [Run]. Перечисленные в ней ключи и значения задают название пункта меню и командную строку для запуска. После инсталляции секция [Run] имеет примерно такой вид:

```
Run Telnet Client = telnet localhost 23
Run MiniM Console = minim.exe
MiniM Routine Editor = minimre.exe
MiniM Global Editor = minimge.exe
MiniM Control Center = minimcc.exe
```

Здесь ключ Run Telnet Client используется как название пункта меню, telnet localhost 23 как командная строка запуска.

Программист может удалить или изменить имеющиеся, добавить свои. Например, добавить запуск minim.exe в консольном режиме с запуском определенной программы, например:

```
My Application = minim.exe -x @mycmds.m
```

и в файле mycmds.m указать строки, которые надо последовательно выполнить.

Изменения в файле minimti.ini будут использованы после рестарта программы minimti.exe.

1.5 Записи в реестре, выполняемые при инсталляции

Реестр используется только в Windows версиях MiniM Database Server. Unix-like версии MiniM Database Server используют ini файлы, размещаемые в подкаталогах

```
/home/username/.minim
```

и в подкаталогах каталога инсталляции сервера.

В реестр вносятся записи, необходимые для работы сервиса, инсталлятора и клиентских программ.

Записи для сервиса

В качестве архитектурной составляющей сервера MiniM используется сервис, как служба компьютера, которая может быть запущена, остановлена, запущена автоматически при запуске компьютера, как локально так и дистанционно, при наличии соответствующих прав. Записи, необходимые для работы сервиса, производятся в разделе реестра

```
HKEY_LOCAL_MACHINE\SYSTEM\
  CurrentControlSet\Services
```

Его дочерние ключи имеют имя производное от "MiniM Service for ", и далее в имени ключа содержится имя инсталляции MiniM. Например, сервис для инсталляции с именем MINIM1 описывается в ключе

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
  Services\MiniM Service for MINIM1
```

Записи в реестре, необходимые для работы сервиса, выполняются автоматически при инсталляции и деинсталляции сервера. Изменение этих записей вручную не рекомендуется. Записи о сервисе в реестре используются операционной системой при запуске, останове и других операциях с сервисом. Другие дежурные записи реестра о сервисе находятся в подключах

```
HKEY_LOCAL_MACHINE\SYSTEM\
  ControlSet001\Enum\Root
HKEY_LOCAL_MACHINE\SYSTEM\
  ControlSet002\Enum\Root
HKEY_LOCAL_MACHINE\SYSTEM\
  ControlSet001\Services
HKEY_LOCAL_MACHINE\SYSTEM\
 ControlSet002\Services
HKEY_LOCAL_MACHINE\SYSTEM\
 CurrentControlSet\Enum\Root
```

и также обновляются только автоматически. Вмешательство в их значения может привести к нарушению работы сервера MiniM.

Записи инсталляции сервера

Для работы инсталлятора выполняются записи в ключ

```
HKEY_LOCAL_MACHINE\SOFTWARE\
  MiniM Group\MiniM\Instance
```

Дочерние подключи этого ключа имеют имя, равное имени инсталляции сервера. Например, инсталляция с именем MINIM1 описывается в ключе

```
HKEY_LOCAL_MACHINE\SOFTWARE\
  MiniM Group\MiniM\Instance\MINIM1
```

В его значениях указываются данные о каталоге инсталляции, версии, группе программ и другие, в зависимости от версии MiniM. На значения этих ключей опирается инсталлятор при обновлении версии (upgrade).

Информация для работы деинсталлятора хранится в ключах

```
HKEY_LOCAL_MACHINE\SOFTWARE\
  Microsoft\Windows\CurrentVersion\Uninstall
```

в подключах с именем, производным от MiniM, например подключ MiniM_is1. Деинсталлятор может быть вызван как программой из группы программ, так и через апплет панели управления "Установка / Удаление программ".

Записи клиентских утилит

Для клиентских утилит выполняются записи настроек подключения и настройки пользователя, сохраняющие внешний вид программ, например цвета подсветки синтаксиса.

Пользовательские настройки клиентских программ сохраняются в ключах

```
HKEY_CURRENT_USER\Software\
  MiniM Group\MiniM\ControlCenter
HKEY_CURRENT_USER\Software\
  MiniM Group\MiniM\GlobalViewer
HKEY_CURRENT_USER\Software\
  MiniM Group\MiniM\RouEditor
```

и для каждого пользователя они различны. В дочерних ключах сохраняемая информация зависит от версии используемых программ и может меняться. Удаление этих ключей не влияет на работу клиентских программ. В случае отсутствия настроек в реестре программы используют значения и поведение по умолчанию.

Настройки подключения хранятся в ключе

```
HKEY_CURRENT_USER\Software\
  MiniM Group\MiniM\Connections
```

Его дочерние ключи имеют имя, равное имени подключения, и значения, описывающие параметры подключения.

При удалении настроек подключения клиентские программы будут показывать пустой список, в этом случае нужно добавить подключение используя окно выбора подключения.

Записи о настройках могут быть экспортированы из реестра и импортированы на других клиентских компьютерах, чтобы не вводить их на каждом. Записи о настройках могут быть изменены также любой программой, записывающей в реестр.

1.6 Бекап и восстановление

Для построения системы, в существенной степени защищенной от сбоев, MiniM предоставляет средства бекапа и восстановления из бекапа баз данных, а также внутренние, предусмотренные архитектурой системы, средства.

Для полного понимания работы системы и для выработки стратегии бекапа администратор MiniM должен понимать устройство, построение и принципы работы сервера MiniM.

Сервер MiniM при работе состоит из процессов: сервис (mnmsvc.exe), демон записи (minimwd.exe), демон расширения (minimed.exe), демон журнала (minimjd.exe) и исполнительные процессы (minim.exe). Запуск и останов сервера выполняется запуском и остановом сервиса. Сервис MiniM управляет разделяемыми областями памяти, запускает демоны и обслуживает подключение по порту телнета с запуском обслуживающего телнет-соединение исполнительного процесса.

В целях защиты от сбоев процессов и от вмешательства в работу процессов сторонними средствами сервис поддерживает слежение за запущенными исполнительными процессами и демонами. В случае сбоя демона выполняется повторный запуск этого демона с индикатором очистки. Запущенный вместо сбойнувшего процесс выполняет очистку внутренних структур разделяемой памяти и продолжает свою работу прозрачно для сервера.

При сбое исполнительного процесса (minim.exe) запускается процесс очистки общих данных от данных сбойнувшего процесса, например снятие блокировок которые он наложил. После чего процесс очистки завершает работу. Этот механизм носит название Process Guardian.

Второй частью архитектуры MiniM, важной для понимания работы с целью выработки стратегии бекапа, является набор файлов для обслуживания баз данных.

Основные данные баз данных находятся в файлах данных, указанных в конфигурационном файле minimdb.ini. Используются как первичные файлы баз данных, так и их вторичные экстенты. Набор файлов состоящий из первичного файла данных и вторичных экстентов образует основное место хранения данных базы данных.

Кроме файлов баз данных в работе сервера используются файлы предварительной записи minim.bij и файлы журналов из каталога journal. Действительное местоположение файла предварительной записи и файлов журнала определяется в конфигурационном файле minim.ini.

Работа исполнительных процессов и демонов MiniM выполняется параллельно и асинхронно. Демоны накапливают изменения, которые необходимо произвести с файлами данных и при наступлении условий сброса выполняют запись. Исполнительные процессы не выполняют запись в файлы данных и журнал, они выполняют только чтение из файлов баз

данных в кеш. Работа с блоками выполняется в кеше. При одновременном доступе к блокам нескольких процессов для разрешения конфликтов поддерживается дисциплина блокирования блоков "один пишет, много читают".

При останове сервера MiniM, после того как завершился последний исполнительный процесс, часть данных может оказаться не в файлах данных. При старте сервер MiniM выполняет необходимые процедуры восстановления для того, чтобы запускаемым исполнительным процессам были предоставлены логически корректные данные глобалов.

Важно! Сервер MiniM предоставляет логически корректные данные для своих исполнительных процессов (minim.exe), а не для файлов файловой системы.

Процедура старта сервера предусматривает гарантированную контрольную точку полной целостности файлов данных, в которой файл предварительной записи не содержит актуальных данных. Процедура штатного останова также предусматривает такую гарантированную контрольную точку. Но контрольная точка полной целостности не гарантируется при сбое питания или при выключении компьютера при работающем сервере.

Методы бекапа для последующего восстановления делятся на две большие группы - холодный бекап и горячий бекап. Холодным называется сохранение файлов данных копированием их. Горячим называется построение специальных файлов бекапа без останова сервера. Холодный бекап разделяется на два - копирование файлов без останова сервера и с остановленным сервером. Для сервера MiniM, работающего асинхронно, холодный бекап без останова сервера не является гарантированным. Для выполнения холодного бекапа при остановленном сервере администратор может выбрать два варианта - копирование файлов данных и файла предзаписи после останова сервера или останов, запуск и останов сервера без изменений глобалов. В первом случае файл предзаписи необходим для корректного гарантированного старта, во втором случае после запуска сервера контрольная точка гарантирована и файлы данных могут быть скопированы без файла предзаписи. В случае, если сервер был остановлен штатно, а не в результате выключения компьютера или питания, то после его останова также можно копировать файлы без повторного старта сервера.

Еще одним важным элементом работы сервера является журнал. К журналу относятся все файлы, находящиеся в каталоге journal. Имя и размещение журнального каталога могут быть изменены администрато-

ром. Файлы журнала рассматриваются сервером MiniM только полностью все, весь набор.

В журнал сервер MiniM записывает действия по изменению глобалов. Эти записи используются в двух задачах: 1) откат транзакции и 2) восстановление баз данных.

Файлы журнала при изменении глобалов непрерывно нарастают, в отличие от файлов данных. Если процессы изменяют несколько узлов глобалов, то общий размер файлов данных может не увеличиваться, в то время как в журнал непрерывно добавляются записи о каждом таком изменении. Журналирование отдельных баз данных может быть отключено или включено в файле конфигурации баз minimdb.ini

Если файлы данных сохраняют текущее одномоментное состояние баз данных, то в журнале записывается информация об их изменении во времени. Эта возможность может быть использована администратором при выработке стратегии восстановления. После того, как выполнен бекап (далее рассматривается исключительно горячий бекап), базы данных продолжают изменяться и информация об изменении накапливается в журнале. При восстановлении из бекапа базы данных будут восстановлены в состояние на последний момент бекапа. Последующие изменения базы данных, записанные в журнале после точки бекапа, могут быть применены при восстановлении. Таким образом сочетание файлов бекапа и файлов журнала образует комплект, необходимый для восстановления баз данных. Администратору следует помнить, что при утрате журнала после времени бекапа восстановление данных после точки этого бекапа невозможно, данные можно будет восстановить только на последний момент бекапа.

Горячий бекап в системе MiniM поддерживается двух видов - полный и дифференциальный.

При полном бекапе выполняются три прохода: 1) сохранение в файл бекапа всех блоков баз данных, указанных для бекапа с отметкой об их сохранении, 2) сохранение в файл бекапа блоков имеющих отметку о несохраненности (измененные за время выполнения первого прохода) и 3) приостанов всех исполнительных процессов и сохранение в файл бекапа всех блоков, имеющих отметку о несохраненности (измененные за время выполнения второго прохода), после чего процессы отпускаются и продолжают работу.

При записи в файл бекапа блоки сжимаются, чтобы уменьшить занимаемый ими объем хранения, поэтому файл бекапа может быть намного меньше чем сохраняемые в него базы данных.

При дифференциальном бекапе выполняется только второй и третий проходы, таким образом что в файле бекапа накапливаются блоки, измененные со времени предыдущего бекапа, полного или дифференциального. При восстановлении баз данных следует указать файлы бекапа в том же порядке, в котором они были получены - полный и последующие дифференциальные, если выполнялись. После последнего из файлов бекапа может быть использована опция повтора изменений, записанных в журнале с тем чтобы привести базу данных в последнее состояние. В этом случае поверх данных, восстановленных из файлов бекапа, будут применены изменения из журнала начиная с точки бекапа. Для этой операции необходимо, чтобы файлы журнала были.

При восстановлении из файла бекапа нужно выбрать одну из трех альтернатив операций с журналом: 1) рассмотренное ранее применение последующих журнальных операций, 2) никаких операций с журнальными записями или 3) откат незакоммиченных транзакций на момент бекапа.

Вторая альтернатива должна использоваться, если производится восстановление из непоследнего бекап файла из набора файлов (полного и дифференциальных).

Третья альтернатива, с откатом незакоммиченных на момент бекапа транзакций, приводит к восстановлению логически корректного с транзакционной точки зрения состояния на момент бекапа. Эта альтернатива может использоваться, если последующие после бекапа журнальные записи не нужны. Для этой операции необходимо, чтобы файлы журнала были.

При необходимости уменьшить занимаемый журналом место администратор может выбрать при бекапе опцию усечения журнала. В этом случае из журнала будут удалены все записи ранее необходимых для выполнения отката открытых на текущий момент транзакций. Операция удаления записей может привести к удалению части журнальных файлов.

Также администратор может при необходимости выполнить усечение журнала безотносительно функции бекапа. Это может входить в выбранную им стратегию восстановления.

Файлы журнала образуют цепочку файлов, из которых, в основном, используется последний. Администратор может выполнить операцию переключения журнала. В этом случае сервер создаст очередной элемент цепочки файлов журнала и предыдущие (остальные кроме последнего)

файлы могут быть скопированы в иное место. Имена файлов, образующих журнал, содержат в имени дату его образования и в качестве расширения очередной номер на эту дату. Ориентируясь в именах этих файлов, администратор может в принципе переместить часть ненужных для выполнения отката транзакций журнальных файлов. Но эта операция может привести к ошибке и администратору следует точно понимать ее необходимость.

При восстановлении из бекапа система сверяет указанный тип восстановления с записанным в файле бекапа.

При восстановлении система проверяет отсутствие на сервере процессов, которые находятся в состоянии открытой транзакции. В этом случае система отказывается от восстановления, предохраняя базы данных от потери изменений, сделанных в транзакции.

В файле бекапа производятся отметки о размере баз данных. В размер входит общий размер всех входящих в базу экстентов, первичного и всех последующих вторичных. При восстановлении система выполняет изменение баз данных на этот размер.

Изменение размеров баз данных и изменение блоков файлов баз данных производится по текущей конфигурации, указанной в файле minimdb.ini. Администратор может выполнить сохранение данных в одной, а восстановление в другой конфигурации файлов баз данных. При необходимости базы будут расширены по правилам, указанным в конфигурации или уменьшены, возможно, до нулевого размера. Блоки баз данных рассматриваются как логическая последовательность, переходящая с одного экстента на другой.

Предоставляемые системой MiniM средства сохранения и восстановления достаточно гибки, чтобы администратор мог реализовать стратегию сохранения баз наиболее приближенную к потребностям администрируемой системы.

Приведем две возможные типовые стратегии сохранения баз.

Первая стратегия - "важны сделанные изменения и их надо передать". В этом случае в MiniM подготавливаются данные (глобалы, рутины, байткод) в состояние готовое к передаче целиком. Выполняется полный бекап отдельной базы данных с усечением журнала. Бекап без журнала передается и восстанавливается на целевой системе в ее конфигурации с опцией игнорировать операции с журналом. На целевой системе оказывается база данных в указанной там конфигурации, содержащая необходимые данные (глобалы, рутины и байткод). Те же самые

данные могут быть переданы путем экспорта данных и их последующего импорта. Во втором случае в целевой базе будут присутствовать данные которые там уже находились, а при восстановлении из бекапа данные будут логически в точности те же что были при сохранении в бекап.

Вторая стратегия - "важно все и по возможности на последний момент сбоя". Администратор выбирает периодичность полного бекапа, например раз в неделю. Полный бекап выполняется с усечением журнала. С меньшим интервалом администратор выполняет дифференциальные бекапы без усечения журнала. За выбранный период полного бекапа (в данном случае неделя) накапливается набор из полного и последующих дифференциальных бекапов. При сбое сервера или баз данных администратор может последовательно выполнить восстановление из полного бекапа и последующих дифференциальных. При этом все кроме последнего файла бекапа восстанавливать без операций с журналом, а последний (вероятно, дифференциальный) восстанавливать с накатом последующих операций по журналу.

Система MiniM не ведет в файлах данных учет выполненных с ними бекапов и файлы бекапа не содержат информацию друг о друге. Поэтому система MiniM не имеет средств проконтролировать очередность применений файлов бекапа при восстановлении из них. Эта задача ложится на администратора. Ему необходимо выработать правила именования файлов бекапа в дополнение к стратегии сохранения. Система MiniM не налагает никаких ограничений на именование файлов и их расширения. Информация о выполненном бекапе остается только в журнале, и эту точку система отыскивает при выполнении журнальной операции при восстановлении.

Операции сохранения и восстановления баз данных выполняются утилитами ^%BACKUP и ^%RESTORE. Кроме того, программисты прикладных систем могут изучить функции $view("db") и встроить функции сохранения и восстановления в свои прикладные системы или сделать собственные укороченные средства сохранения и восстановления, максимально точно отвечающие выбранной стратегии сохранения и восстановления.

Действия администратора при возникновении проблем с хранением данных сводятся комбинированию трех действий - устранение проблем с файловой системой, выполнение восстановления или выполнение сохранения.

1) Погибли один или больше экстентов базы данных или файл предзаписи, журнал цел.

Решение - восстановить из бекапа или их последовательности с накатом журнала.

2) Погиб журнал.

Решение - удалить журнал и сделать полный бекап.

3) Погибли файлы данных и журнал.

Решение - восстановить из бекапа без наката журнала.

4) Сбой сервера при восстановлении.

Решение - Устранить проблему с файловой системой или конфигурацией в зависимости от вида сбоя и повторить восстановление.

5) Погиб файл предзаписи и в логе minim.log запись о проблеме восстановления сервера при старте.

Решение - остановить сервер, удалить файл предзаписи minim.bij и стартовать сервер.

После действий по восстановлению рекомендуется проверить характер потери данных в базах данных с точки хрения работающих на них прикладных систем, поскольку некоторые виды восстановления не могут восстановить некоторую часть последних сделанных изменений. В любом случае рекомендуется иметь на сервере средства бесперебойного питания и периодически проверять файловую систему на предмет сбоев.

1.7 Нехватка места на дисках

При нехватке места на дисках сервер предпринимает действия по ожиданию освобождения места, записи диагностики в файл minim.log и процессы, которым требуется выполнение дисковых операций с базой данных, генерируют ошибки. Сервер распознает следующие ситуации ограничений:

Установленное ограничение роста базы данных.

При обнаружении, что выполняемая операция требует расширения базы данных, но текущий размер равен установленному административно в файле minimdb.ini ограничению роста процесс генерирует ошибку <DBLIMIT>.

К рекомендуемым действиям в этой ситуации можно отнести анализ выполняемой программы и соответствие потребностей установленным ограничениям.

Нехватка дискового пространства для файла предзаписи minim.bij.

При нехватке места для выполнения записи в файл предзаписи minim.bij демон записи сообщает об этом в файл minim.log и переводит все смонтированные базы данных в состояние "только чтение". При выполнении попытки модификации данных процесс генерирует ошибку <DBREADONLY>.

При обнаружении этой ситуации в кеше сервера находятся блоки данных, которые не записаны на диск. Периодически демон записи предпринимает попытки выполнить эту запись. К рекомендуемым действиям относится освобождение дискового пространства для роста файла minim.bij, после чего используя функции $v("db",23) или $v("db",24) выполнить восстановление состояния "только чтение" до того, которое было указано в файле minimdb.ini для базы данных. Это изменение возможно без перезапуска сервера и модификации файла конфигурации minimdb.ini. После того как рост файла minim.bij станет возможен, демон записи завершит операцию записи измененных блоков. После того как будет разрешена запись в базы данных, процессы сервера могут продолжить работу, транзакционный контекст процессы не утрачивают.

Нехватка места для расширения базы данных.

При обнаружении ситуации, что расширение базы данных невозможно, демон расширения сигнализирует запросившим расширение процессам о такой невозможности и процессы генерируют ошибку <DBEXTEND> и смонтированные базы данных переводятся в режим "только чтение".

К рекомендуемым действиям относится анализ потребностей программ, свободного места на дисках и освобождение пространства. После освобождения пространства для роста баз данных следует перевести с помощью функций $v("db",23) или $v("db",24) базы данных в состояние разрешения записи. Последующие операции, требующие расширения баз данных, будут выполняться в обычном режиме.

Нехватка места для журнала.

При обнаружении ситуации невозможности добавить записи в журнал демон журнала переводит базы данных в состояние "только чтение" и периодически предпринимает попытки записать очередь журнальных записей в файл. При этом в очереди записи могут находиться журнальные записи, которые не записаны в файл журнала. Останов сервера в этом состоянии может привести к утрате части журнала. Кроме того, демон журнала предпринимает попытки записать информацию об ошибке в файл minim.log.

К рекомендуемым действиям относится освобождение пространства на дисках для роста журнала и перевод баз данных с помощью функций $v("db",23) или $v("db",24) в состояние разрешения записи.

К общим рекомендациям по использованию дискового пространства можно отнести неиспользование ограничений на рост баз данных, периодический бекап с усечением журнала и периодический мониторинг свободного пространства на используемых сервером дисках.

1.8 Удвоение символов в телнете

При использовании телнет-клиента Windows по умолчанию (telnet.exe) при работе с MiniM может проявляться удвоение вводимых в телнете символов. Это настройка клиентской части. Чтобы отключить ее, следует соединившись с MiniM набрать в окне телнет-клиента Windows:

```
Ctrl+]
```

Это переведет телнет-клиент в режим управления локальными настройками. Далее следует в нем набрать команду

```
unset LOCAL_ECHO
```

После чего дважды нажать клавишу Enter. Телнет-клиент вернется в состояние диалога с MiniM.

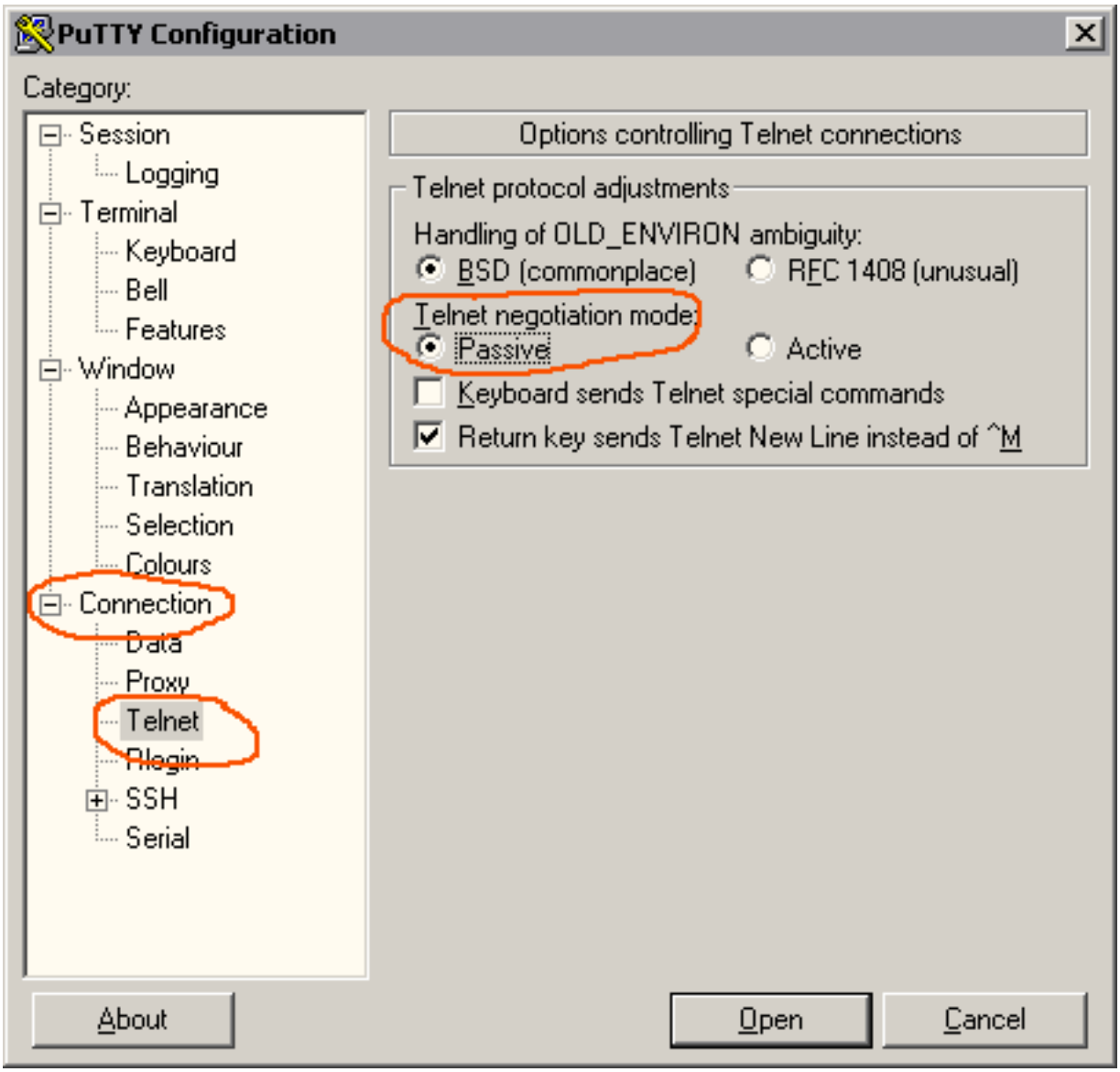
Полный список команд телнет-клиента Windows доступен по команде

```
?
```

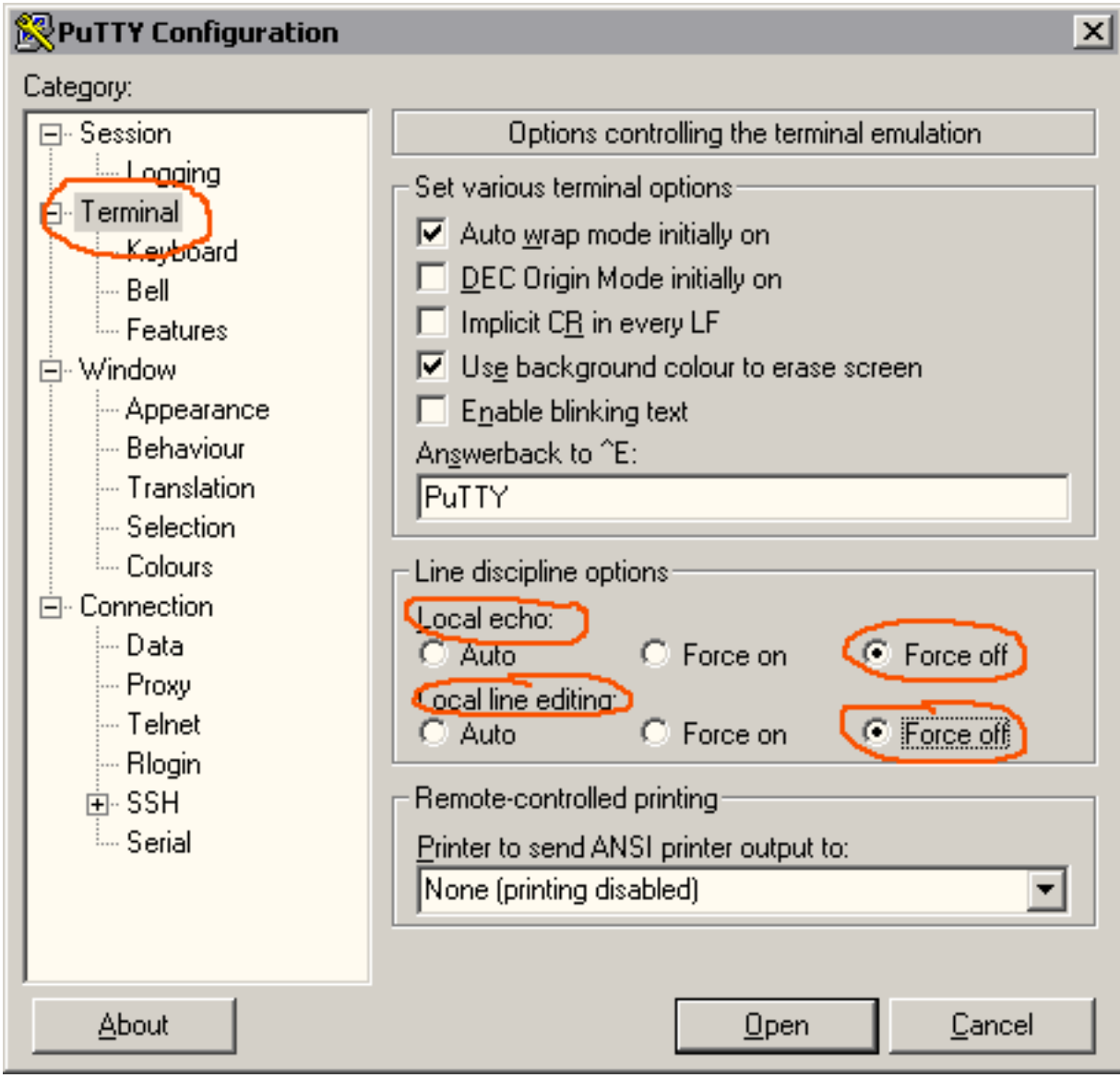
Включение и выключение опций производится командами set и unset. Список опций этих команд выводится по командам:

```
set ?
unset ?
```

При использовании телнет-клиента Putty также может наблюдаться удвоение символов при вводе. Для отключения удвоения символов следует в окне настройки конфигурации Putty в дереве настроек Category выбрать Connection, в нем выбрать Telnet, и в опции Telnet negotiation mode выбрать Passive. На скриншоте показано расположение элементов:



Также нужно установить еще одну настройку работы с эхом. В дереве настроек Category выбрать Terminal, в обоих опциях Local echo и Local line editing выбрать значение Force off. На скриншоте показано расположение элементов:



В случае какой-либо специфики работы прикладной программы с телнет-протоколом следует проконсультироваться с разработчиками прикладной программы о настройке телнет-клиента под специфику программы.

1.9 Редактор определения кодировки

В комплекте сервера MiniM находится редактор определения кодировки символов MiniM Collation Editor (/bin/minimne.exe). Эта утилита предназначена как для создания нового определения кодировки, так и для

редактирования имеющегося.

Определение кодировки содержится в файле с расширением .N и он должен находиться в подкаталоге /nat. В файле размещается три определения в виде таблиц:

1. Поднятие регистра

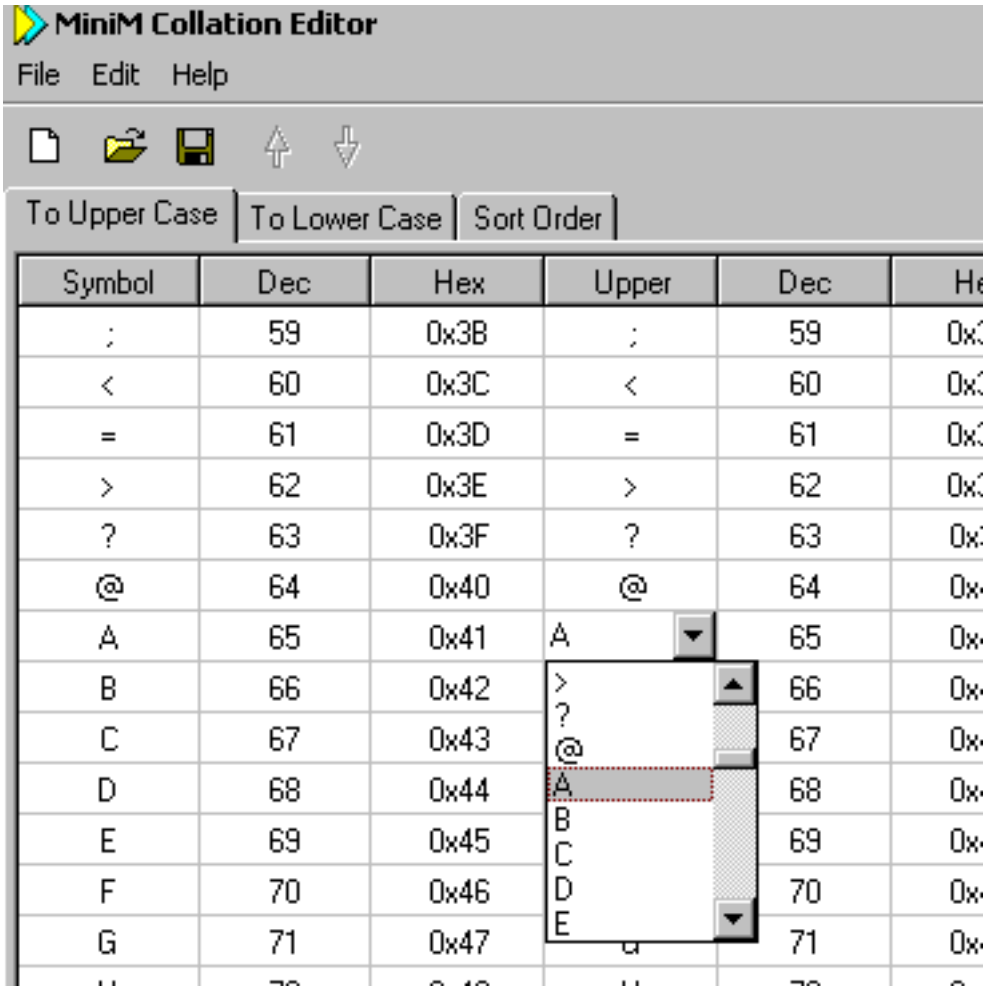
2. Опускание регистра

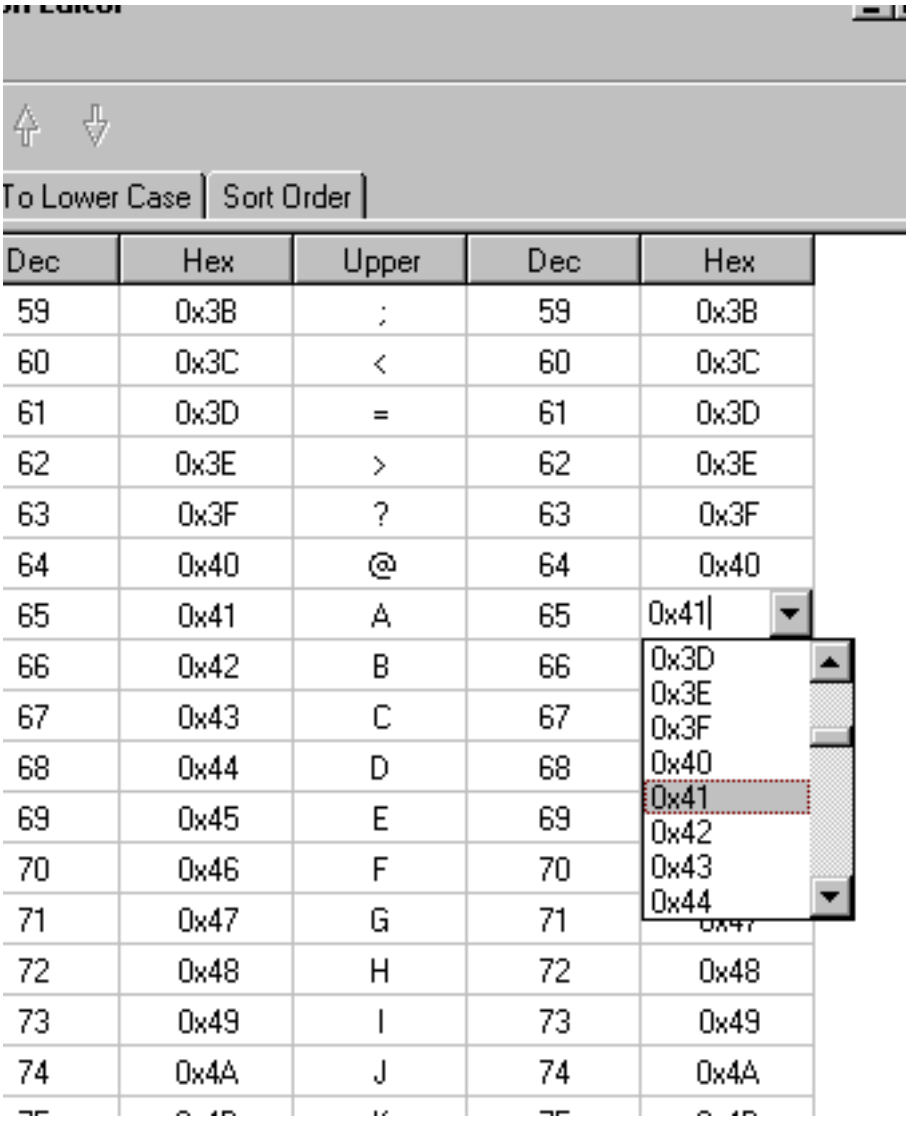
3. Порядок сортировки

Для каждой из этих таблиц редактор показывает правило отображения в виде трех панелей. При сохранении в файл эти три таблицы сохраняются одновременно.

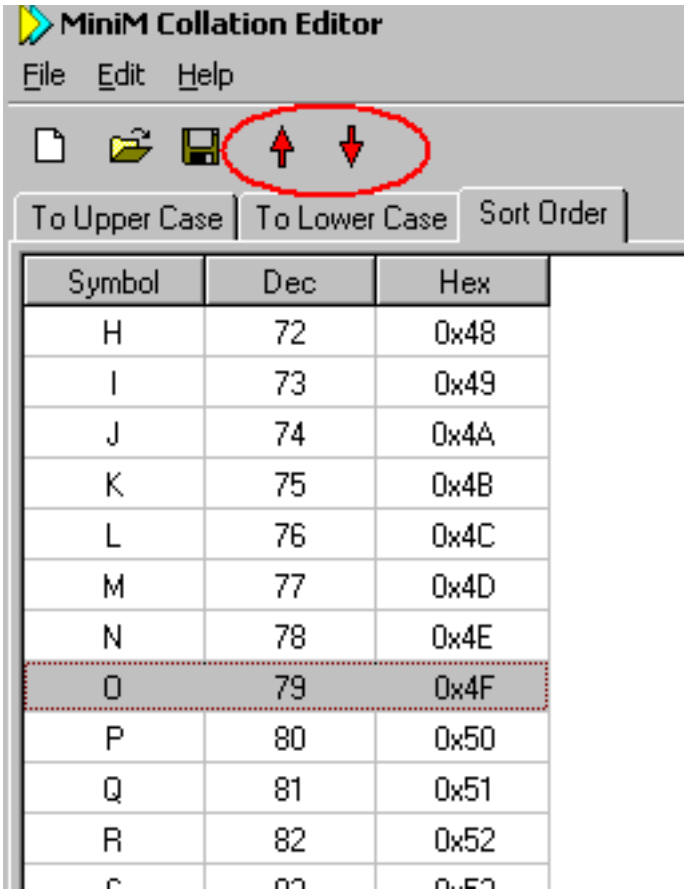
Редактор дает отдельно редактировать правило поднятия и опускания регистра и не следит за их взаимным соответствием.

На панели редактирования редактор показывает три колонки - символ в выбранном текущем шрифте, его десятичный и шестнадцатеричный код. Для выбора кода символа который должен получиться при опускании или поднятии регистра надо в правой колонке (символ, десятичный или шестнадцатеричный код символа по выбору) выбрать поле, активировать мышкой редактор и выбрать во что должен быть преобразован символ.


MiniM Collation Editor
File Edit Help
To Upper Case
To Lower Case
Sort Order
Symbol Dec Hex Upper Dec
; 59 0x3B ; 59
< 60 0x3C < 60
= 61 0x3D = 61
> 62 0x3E > 62
? 63 0x3F ? 63
@ 64 0x40 @ 64
A 65 0x41 A 65
B 66 0x42 66
C 67 0x43 67
D 68 0x44 68
E 69 0x45 69
F 70 0x46 70
G 71 0x47 71
> ? @ A B C D E



Для выбора порядка сортировки надо переключиться на закладку "Sort Order", выбрать символ для перемещения относительно других в порядке сортировки и нажимая на тулбаре кнопки перемещения поместить его в нужной очередности среди других символов. Также предоставляются акселераторы Ctrl+Up и Ctrl+Down.



При старте редактор автоматически заполняет все таблицы значениями по умолчанию. По умолчанию определено поднятие и опускание регистра для латиницы. Для символов с кодом больше 127 поднятие и опускание регистра не определено. Порядок сортировки по умолчанию соответствует очередности сортировки кодов символов.

Определение кодировки указывается в файле minim.ini, секция [Server], ключ Locale. В значении ключа должно быть указано имя файла без расширения, и этот файл должен находиться в подкаталоге /nat. Определение кодировки принимается сервером при старте.

Таблица поднятия регистра используется в функциях $zupper и $zcvt(str,"U"). Таблица опускания регистра используется в функциях $zlower и $zcvt(str,"L").

Кроме того, таблицы поднятия и опускания регистра используются для определения правила что символ есть буква - если результат поднятия и опускания регистра для символа различаются, то сервер считает этот символ буквой и использует это правило в шаблонах при сопоставлении по коду "A".

Таблица сортировки символов используется в операторе индексной сортировки при определении следования символов и при работе с индексами локальных и глобальных переменных.

Для смены таблицы определения кодировки для сервера следует:

1. Определить, для каких имеющихся в базах данных глобалов индексная сортировка критична.
2. Экспортировать эти глобалы в форматах с переменной длиной.
3. Удалить эти глобалы из баз данных.
4. Остановить сервер.
5. Указать серверу в файле minim.ini новое определение кодировки
6. Стартовать сервер
7. Импортировать данные глобалов.

Смена правила сортировки для имеющихся глобалов может привести к появлению фантомов в базе данных, когда для одних операций данные видны, а для других нет. Нужно понимать, что правило сортировки индексов становится частью алгоритмики поиска, удаления и вставки данных. Для локальных переменных смена праивла сортировки безопасна, поскольку правило сортировки применяется только при рестарте сервера, при отсутствии локальных переменных. При восстановлении баз данных из бекапа также следует помнить о соответствии сортировки баз использованных при бекапе и при восстановлении. Они должны совпадать.

1.10 Использование ключа лицензии

Для сервера MiniM используется ввод лицензии в виде ключа лицензии, который представляет собой файл minim.lic, располагающийся в подкаталоге сервера /bin.

Файл представляет собой текстовый файл формата INI, например:

```
[License]
Customer=MiniM Test Group
Count=100
Date=2010.03.31
Key=68DB33AA69A1ADE5078B2F48363
```

В файле описывается кому выдан ключ, на сколько процессов, дата окончания действия и ключ подписи корректности информации. Возможно наличие других полей.

Если дата окончания действия ключа не указана, то ключ считается бессрочным.

Если ключ отсутствует, или он некорректный, или дата действия истекла, то сервер считает что количество лицензированных процессов 0.

Общее ограничение на количество процессов сервера складывается из количества лицензированных плюс 3 инженерных. Таким образом, при истечении срока ключа или при его отсутствии сервер позволяет запускать 3 процесса. Инженерные процессы предназначены для тестирования, ознакомления или выполнения срочных нештатных работ на сервере. В любом случае эти процессы пользователь может использовать по своему усмотрению.

Общее ограничение числа процессов вычисляется как вычисленное по лицензии и указанное в файле настроек minim.ini, секция [Server], ключ ProcessLimit. Используется минимальное из них значение. При вводе ключа администратору нужно проверить оба значения. Ключ ProcessLimit позволяет ограничить использование процессов операционной системы в случае если в последующих версиях сервера будет поддерживаться неограниченное число процессов, для предотвращения бесконтрольного расхода ресурсов сервера. При старте сервера внутренние структуры данных рассчитываются на ограниченный размер и в работе не изменяются динамически, что позволяет повысить стабильность работы сервера.

Для смены лицензионного ключа нужно заменить файл minim.lic в подкаталоге /bin на новый и перезапустить сервер.

MiniMono difference

Для MiniM Embedded Edition не используется никакого ключа лицензирования, это бесплатное програмное обеспечение.

Глава 2

Работа с устройствами

2.1 Работа с устройством |TCP|

Для работы с устройством |TCP| следует первым делом выбрать характер использования устройства сокета. Это может быть клиент, сервер или конкурентный сервер.

Как использовать устройство типа |TCP| в качестве сервера.

Для использования устройства |TCP| в качестве сервера следует выполнить шаги:

1) Создать устройство.

2) Сделать его текущим и выбрать режим работы - текстовый или бинарный, и для бинарного выбрать терминатор.

3) Выполнить операцию /ACCEPT. После этого, устройство готово принимать внешние подключения и в случае такового, после команды

```
use dev:/ACCEPT
```

следующая выполняемая команда выполняется после того, как внешняя программа подключилась к устройству. После этого можно выполнять операции чтения - записи.

4) Отключить устройство командой close.

Пример работы с сокетом в качестве сервера

```
s dev="|TCP|:123"  ; 1
o dev              ; 2
u dev:/ACCEPT      ; 3
r *ch              ; 4
w $c(ch+1)         ; 5
```

```
u 0                     ; 6
c dev                   ; 7
```

Разберём этот пример по строкам.

1) Указывается, что тип устройства |TCP|, и оно принимает коннекты от всех на порт 123.

2) Устройство открывается.

3) Устройство делается текущим с одновременным применением к нему операции /ACCEPT.

4) Эта команда срабатывает после того, как внешняя программа подключена к устройству. Производится чтение кода одного символа в переменную ch.

5) В устройство, обратно клиентской программе, отсылается символ с кодом на единицу больше принятого.

6) Текущим делается устройство по умолчанию.

7) Устройство сокета закрывается, дальнейшие операции с ним невозможны.

Для этого примера можно использовать штатную программу telnet, запустив её и набрав параметры коннекта:

```
open localhost 123
```

Для повторного использования устройства его необязательно закрывать. Повторный переход в операцию /ACCEPT закрывает открытое ранее соединение с клиентской программой и снова ожидает подключения извне.

Вариант вышеприведённого сервера, обслуживающего подключения подряд:

```
s dev="|TCP|:123"
o dev
f  u dev:/ACCEPT r *ch w $c(ch+1)
```

Здесь в бесконечном цикле выполняется операция /ACCEPT, применительно к серверному сокету, после чего читается код символа, и передаётся символ с кодом на единицу больше. Первое выполнение операции /ACCEPT ожидает первого подключения, последующие - закрывают текущее подключение и ожидают очередное.

Более сложный вариант обслуживания входящих соединений - это обслуживание HTTP запросов. В примере открывается устройство, ожидается подключение к нему, читаются входящие заголовки HTTP ответа и выдается константный HTTP ответ в виде простой HTML страницы.

```
socktest
 ; open socket device
 n io="|TCP|:2233"
 ; close device if was opened
 i $d(^$d(io)) c io
 ; device was specified as server-side TCP socket on 2233 port
 ; open it in read-write and text mode
 OPEN io  USE io:(/MODE="rwt")
continue
 ; wait incoming connection
 ; this example does not use timeout for accept
 USE io:/ACCEPT
 ; accepted socket is inside of current device and will be used
 ; in read and write operation until next accept or closing
 d trace("accepted")
 ; read headers
 n line,headers f  r line:0.01 q:line=""  d
 . s headers($i(headers))=line
 d trace("headers done")
 ; dump headers to console
 d dump(.headers)
 ; write answer
 d answer
 d trace("answer done")
 ; continue
 g continue
answer
 w "HTTP/1.0 200 OK",$c(10)
 w "Content-Type: text/html",$c(10)
 n text="<html><head></head><body>"
 s text=text_"Constant HTTP answer.</body></html>"_$c(10)
 w "Content-Length: ",$l(text),$c(10)
 w $c(10)
 w text
 q
trace(str)
 ; write to principal, not to socket
 n saveio=$io
 u $p
 w "Trace: ",str,!
 u saveio
```

```
 q
dump(var)
 q:'$d(var)
 ; write to principal, not to socket
 n saveio=$io
 u $p
 n i f i=1:1:var w var(i),!
 u saveio
 q
```

После запуска этого примера в консоли или в телнете он ожидает подключения WEB браузера по адресу

```
http://localhost:2233
```

и выводит на экран диагностику обслуживания.

Как использовать устройство типа |TCP| в качестве конкурентного сервера.

Для использования устройства типа |TCP| в качестве конкурентного сервера необходимо спланировать сервер на две части - родительскую и дочернюю.

Родительская часть должна создать устройство типа |TCP|, выполнить операцию /ACCEPT, после чего готова к запуску дочерней части с передачей ей сокета в конкурентном режиме. В качестве дочерней части с помощью команды job могут запускаться как одинаковые метки, так и разные. После запуска дочерней части сокет родительской части вторично готов к операции /ACCEPT. Итого, следует выполнить примерно такие шаги для родительского процесса:

1) Создать устройство типа |TCP|, указав ему, какой порт слушать, и не указывать имя или адрес хоста. Это создаст устройство серверного типа.

2) Выбрать режим работы сокета в родительском процессе.

3) Сделать его текущим.

4) Выполнить операцию /ACCEPT.

5) После возврата управления из операции /ACCEPT родительский процесс готов переключить подключившийся извне клиентский сокет на дочерний процесс.

6) Запустить командой job дочерний процесс, указав необходимую метку и указав в параметрах процесса строку идентификации устройства |TCP|, над которым выполнена операция /ACCEPT и, значит, получено внешнее подключение.

7) После команды job текущий сокет родительского процесса не закрывается и может быть использован повторно для приёма подключений операцией /ACCEPT.

8) При необходимости, закрыть устройство |TCP|. При этом запущенные ранее дочерние процессы продолжают работать со своей копией сокета и сохраняют соединение с клиентским подключением.

Для дочернего процесса ничего специального предпринимать не нужно, за исключением учёта тех обстоятельств, что дочерний процесс получает устройство типа |TCP| в качестве устройства по умолчанию, что устройство автоматически открывается в бинарном режиме, и что ему не установлен никакой терминатор чтения.

Пример работы с TCP устройством в качестве конкурентного сервера.

```
 ; parent server part
srv
 n dev="|TCP|:2525"     ; 1
 o dev                  ; 2
 f  q:$d(^STOP)  d      ; 3
 . u dev:/ACCEPT        ; 4
 . j child:(:$io)       ; 5
 c dev                  ; 6
 q

 ; child server part
child
 u $p:(/MODE="rwt")
 w "Child job, #"_$j,!
 r "enter your name: ",name,!
 w "Name is: ",name,!
 r "Press any key to quit.",name,!
 q
```

Разберём этот пример по строкам.

Родительский процесс.

1) Объявляется строка идентификации устройства типа |TCP| серверного типа с приёмом коннекта на порт 2525.

2) Открывается устройство.

3) В цикле пока не найден признак окончания цикла выполнять.

4) Открытое ранее устройство типа |TCP| сделать текущим, и применить операцию /ACCEPT.

5) Запустить дочерний процесс с метки child с передачей ему сокета в конкурентном режиме (указано в параметрах процесса как (:$io)).

6) При окончании цикла открытое устройство закрывается, и текущим становится устройство по умолчанию.

Дочерний процесс.

Приведённый в примере дочерний процесс ориентирован на взаимодействие с телнет-клиентом и имеет простейший интерфейс. Первым делом дочерний процесс меняет текущему устройству режим с бинарного на текстовый, после чего к устройству в операциях чтения-записи автоматически применяются соглашения о терминаторе чтения и переводе строки. Далее идут простые операции с вводом-выводом.

Если телнет-клиент не отображает символов, вводимых командой read, то нужно ему включить режим отображения эха. Например, в Windows Telnet Client это выполняется командами

```
set LOCAL_ECHO
```

и

```
unset LOCAL_ECHO
```

Как использовать устройство типа |TCP| в качестве клиента.

Для использования устройства |TCP| в качестве клиента следует выполнить шаги:

1) Создать устройство, указав, к какому серверу и по какому порту следует подключаться.

2) Выбрать режим работы устройства, текстовый или бинарный, какой терминатор и прочее.

3) Сделать его текущим.

После этого устройство готово к коммуникации со сторонней программой, выступающей в качестве TCP сервера.

4) Отключить устройство командой close.

Пример работы с TCP устройством в качестве клиента.

```
s dev="|TCP|localhost:80"  ; 1
o dev:(/MODE="rwt")        ; 2
u dev                      ; 3
w "GET /",!                ; 4
```

```
f i=1:1:10 r ans(i)          ; 5
u 0                          ; 6
c dev                        ; 7
w                            ; 8
```

Разберём этот пример по строкам.

1) Объявляется, что в устройстве типа TCP, указывается сервер localhost (этот же компьютер) и порт 80. В этом примере демонстрируется запрос к веб-серверу и получение от него в качестве ответа веб-страницы.

2) Устройство открывается в режиме текст, разрешение на чтение и на запись.

3) Устройство делается текущим. После этого команды чтения и записи направляются к нему, как к текущему.

4) В устройство пишется строка "GET /" и перевод строки. В соглашениях протокола http это строка - требование выдать корневую страницу ("/") и указывает, что направляется запрос типа GET ("GET"). Вид протокола не указывается. Перевод строки в протоколе передачи веб-запроса служит индикатором, что команда завершена.

5) В цикле от 1 до 10 читаются строки из устройства в индексированную переменную ans. Устройство открывалось в текстовом режиме, поэтому окончание чтения строки определяется по приходу символов перевода строки, содержащихся в файле html, которым отвечает веб-сервер.

6) Текущим устройством делается устройство по умолчанию.

7) Клиентское TCP устройство закрывается. Закрывается коннект к веб-серверу, и забываются все данные, которые использовались этим устройством.

8) На текущее устройство выводятся все локальные переменные для визуального контроля.

Например, если обратиться к веб-серверу Apache со страницей по умолчанию, то в переменную ans прописывается следующее содержание:

```
ans(1)="<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">"
ans(2)="<HTML>"
ans(3)=" <HEAD>"
ans(4)="  <TITLE>Test Page for Apache Installation</TITLE>"
ans(5)=" </HEAD>"
ans(6)="<!-- Background white, links blue (unvisited),
  navy (visited), red (active) -->"
ans(7)=" <BODY"
```

```
ans(8)="  BGCOLOR="#FFFFFF""
ans(9)="  TEXT="#000000""
ans(10)="  LINK="#0000FF""
```

В случае если программист использует не текстовый, а бинарный режим, то он должен самостоятельно определить либо терминатор чтения, либо ограничить чем-то длину чтения или время. Сервер может отправить, например, 100 байт, и программа, использующая клиентский сокет, должна корректно определить, где закончены данные. В распоряжении программиста генерация командами чтения - записи ошибок <READ>, <WRITE>, <END OF FILE>, <DEVICE>, указание времени ожидания и количества считываемых байт.

2.2 Работа с устройством |CON|

Описание процессора эскейп последовательностей устройства консоль.

При выводе в устройство консоль символов процесс MiniM проверяет последовательность, отыскивая в нем управляющие эскейп - последовательности. Эскейп - последовательность начинается символом $c(27) и завершается символом - терминатором, до которого пропускаются символы из набора: [, ;, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, =, ?. Символы разделители отделяют численные значения параметров эскейп-последовательности. Действие, сопоставляемое эскейп - последовательности, определяется символом - терминатором, предшествующие ему числовые значения рассматриваются как параметры. В зависимости от эскейп - последовательности параметров может не быть, может быть один, два или несколько.

Сокращения в описаниях последовательностей

Сокращ.	Описание
Esc	Символ AP2 (десятичный код 27)
Pn	Число, представленное в символьном виде (12, 1,...)
Pl	Номер строки дисплея (line)
Pc	Номер позиции в строке (column)
Pa	Атрибут, определяющий характеристики отображения

Если для эскейп - последовательности значение параметра не указано (пропущено), то оно принимается за значение по умолчанию. Недопу-

стимые значения параметров игнорируются. Неподдерживаемые эскейп - последовательности игнорируются.

Управляющие символы

Код	Описание
7	Звонок
8	Шаг влево на одну позицию
9	Горизонтальная табуляция, шаг 8 символов
10	Перевод строки (позиция не меняется)
11	Вертикальная табуляция (то же, что 10)
12	Перевод формата (очистка экрана, курсор в начало экрана
13	Возврат каретки (курсор в начало строки)

Команды перемещения курсора

Начало экрана (строка 1,позиция 1)	EscH или Esc[H или Esc[f
Курсор вверх (в той же позиции)	Esc[PnA или Esc[A или EscA
Курсор вниз (в той же позиции)	Esc[PnB или Esc[B или EscB
Курсор вперед (вправо)	Esc[PnC или Esc[C или EscC
Курсор назад (влево)	Esc[PnD или Esc[D
Позиционирование(прямое)	Esc[Pl;PcH или Esc[Pl;Pcf
Сохранить координаты курсора и атрибуты	Esc[s
Восстановить координаты курсора и атрибуты	Esc[u

Команды очистки экрана

От курсора до конца строки	Esc[0K или Esc[K или EscK
От начала строки до курсора	Esc[1K
Всю строку	Esc[2K
От курсора до конца экрана	Esc[0J или Esc[J или EscJ
От начала экрана до курсора	Esc[1J
Весь экран	Esc[2J

Команды вставки/удаления

Вставка пустых строк	Esc[PnL

Удаление строк	Esc[PnM
Стирание символов (замена на пробелы)	Esc[PnX

Команды видео-атрибутов

Установить атрибут символа	Esc[Pa;Pa;...;Pam

Коды видео-атрибутов

0	все атрибуты отключены
1	повышенная яркость включена
2	повышенная яркость отключена
30	черный символ
31	красный символ
32	зеленый символ
33	желтый символ
34	синий символ
35	сиреневый символ
36	символ цвета циан
37	белый символ
40	черный фон
41	красный фон
42	зеленый фон
43	желтый фон
44	синий фон
45	сиреневый фон
46	фон цвета циан
47	белый фон

Скрытие и показ курсора

Скрыть курсор	Esc[?25l
Показать курсор	Esc[?25h

2.3 Мнемоника ATR

Мнемоника ATR поддерживается для устройств типа |CON| и |TNT| для совместимости с множеством написанных ранее программ и является

сокращенной альтернативой мнемонике SGR. Мнемоника ATR не входит в общеподдерживаемые стандарты.

Во всех случаях работы с мнемоникой /ATR(n) цветовой атрибут задается в виде числа больше 0 и меньше 256 по следующей схеме:

Номер бита	Значение
0, 1, 2	Цвет текста
3	Яркость текста
4, 5, 6	Цвет фона
7	Яркость фона

Таким образом, цветовой атрибут удобно задавать в виде шестнадцатиричного числа, в котором первая цифра задает цвет фона, а вторая цвет текста. Если требуется задать повышенную яркость текста следует прибавить ко второй цифре восемь, а если повышенную яркость фона или мигание (зависит от терминала), то восемь следует прибавить к первой цифре.

Коды цветов:

0	черный
1	красный
2	зеленый
3	желтый
4	синий
5	фиолетовый
6	голубой
7	белый

Например:

#17	красный фон белые буквы
#07	черный фон белые буквы
#4B	синий фон ярко-желтые буквы
#AF	ярко-зеленый фон ярко-белые буквы

Пример использования:

```
W /ATR(#05),"фиолетовый текст"
W /ATR(#42),"зеленый текст на синем фоне"
W /ATR(#07),"обычный белый цвет"
```

Глава 3

Технические статьи

3.1 Ключи командной строки minim.exe

Исполняемый файл minim.exe поддерживает несколько ключей командной строки, часть из которых предназначена для внутреннего использования и не документируется, и часть предназначена для управления запускаемым процессом при запуске процесса путем запуска minim.exe.

Ключи командной строки могут быть указаны в произвольной последовательности. Если у ключа есть значение, то значение должно следовать по порядку после указания ключа. Например, ключ -h можут быть указан в произвольном месте, но после ключа -x ожидается значение с заданием строк команд непосредственно следом за ключом.

Ключ -std.

Ключ указывает что запущенный процесс должен использовать для ввода-вывода устройство по умолчанию |STD|. В этом режиме поддерживается перенаправление ввода и вывода на стандартные устройства stdin и stdout. Ввод - вывод могут быть перенаправлены в файл или из файла, прочитаны и записаны иными процессами через stdin и stdout. В случае если процесс запускается с перенаправлением ввода (< или >), то процесс автоматически определяет, что он должен использовать устройство |STD|.

В случае если процесс запускается в режиме консоли, и может получать ввод с клавиатуры как консоль, то он автоматически использует устройство по умолчанию |CON|. В случае если при этом требуется использовать устройство |STD|, это можно указать явно ключом запуска -std. В этом случае процессирование ввода-вывода консоли не используется и работает простой ввод-вывод на стандартное устройство.

Ключ -x

Ключ -x задает указание использовать последующий параметр командной строки как строку команд для исполнения. Если последующий за ним параметр начинается на символ @, то остаток параметра после символа @ трактуется как имя файла, строки которого содержат строки команд для их последовательного исполнения, как если бы они вводились одна за другой с клавиатуры. Строки в файле могут быть разделены пустыми строками. Например:

```
minim.exe -x "w $zv,!"
```

здесь запускается процесс MiniM и выполняет команду

```
w $zv,!
```

```
minim.exe -x @script.m
```

здесь запускается процесс MiniM и выполняет последовательно строки команд из файла script.m.

Ключ -h

Ключ -h указывает, что после окончания выполнения строки команд или строк команд из файла следует завершить процесс, выполнив команду halt. При исполнении команд процесс может также самостоятельно выполнить команду halt, если она указана, в этом случае ключ просто игнорируется.

Ключ -ignore

Ключ -ignore указывает что процесс должен игнорировать этот и последующие за ним параметры командной строки. Встретив этот ключ, процесс прекращает дальнейший анализ параметров.

Ключ -ignore используется в случае необходимости запуска процесса из сторонних средств, которые автоматически формируют дополнительные параметры командной строки, например cgi шлюз вебсервера Apache.

Ключ -nspace

Ключ -nspace указывает что процесс должен при старте переключиться в указанную в параметре ключа базу данных. Пример:

```
minim.exe -x "d label^routine" -nspace USER
```

Если база данных процесса явно не указана, то используется база данных по умолчанию, указанная в настройках в файле minim.ini, секция [Process], ключ Namespace.

Иные ключи

Иные ключи, если не являются внутренними и недокументируемыми, выводятся в файл minim.log и игнорируются. Запуску процесса MiniM они не препятствуют, но считаются ошибкой формирования командной строки и запись в файл minim.log является предупреждением программистам и администраторам. Это может быть либо опечатка либо ошибка.

Перенаправление ввода-вывода

Для перенаправления ввода-вывода процесс MiniM поддерживает перенаправление через символы <, >, |. Если процесс определяет, что было перенаправление ввода, то он автоматически запускается с устройством по умолчанию |STD|. По окончании перенаправленного ввода процесс заканчивает работу. При работе с текущим устройством |STD|, вызванного либо перенаправлением ввода-вывода либо явным заданием ключа -std, процесс не выводит в текущее устройство промпт в виде имени области и символа >, а также не выполняет перевод строки после выполнения каждой строки. Процесс считает, что программист самостоятельно задает, как должен формироваться вывод. Процесс MiniM не формирует вывод в устройство stderr.

Примеры использования перенаправления:

```
1) minim.exe < cmds.m
2) minim.exe < cmds.m > out.txt
3) minim.exe < cmds.m >> out.txt
4) echo w $zv,! | minim.exe
5) echo w $zv,! | minim.exe > out.txt
```

Здесь первый пример запускает процесс MiniM с вводом строк команд из файла cmds.m. Все строки разбираются последовательно. После того, как строки кончатся, процесс прекращает работу.

Второй пример есть то же самое, что и первый, но вывод производится не на экран, а в файл out.txt. По окончании работы весь вывод который выполняют команды из файла cmds.m сохраняется в файле out.txt

Третий пример есть то же самое, что и второй, но вывод, который производят команды из cmds.m, не замещает содержание файла out.txt, а дописывается к нему с конца (append).

Четвертый пример запускает команду echo, которая выводит символы w $zv,! на вход перенаправления процесса MiniM. Символы аргумента команды echo до символа | идут на вход процесса MiniM.

Пятый пример есть то же самое, что и четвертый, но вывод производится не в текущее устройство отображения операционной системы, а в указанный файл out.txt.

Использование перенаправления ввода-вывода для процесса MiniM позволяет использовать его различными способами в разнородных средах.

3.2 Разработка модуля ZDLL

Поддержка модулей ZDLL системой MiniM позволяет сторонним программистам разрабатывать различные расширения на компилируемых языках, в виде динамически подключаемых библиотек (dll). Механизм передачи данных позволяет как вызывать внешние dll, соответствующие программному интерфейсу ZDLL, из среды исполнения MiniM, так и вызывать операции исполняющей среды MiniM из таких библиотек. Библиотеки динамического подключения, поддерживающие интерфейс ZDLL, далее называются модулями ZDLL.

Программный интерфейс с внешними ZDLL модулями на стороне исполняющей среды MiniM обеспечивается системной z-функцией $zdll. Её описание содержится в документации по языку minimlang.pdf.

Программный интерфейс ZDLL в настоящее время предоставляется в виде, ориентированном на применение средств разработки на языках C или C++. Интерфейс допускает использование его в средах разработки, отличающихся от C / C++, например Delphi. В этом случае программист должен самостоятельно описать интерфейс, в соответствии с используемым им средством разработки.

Общий принцип работы модулей ZDLL состоит в следующем. Системная функция $zdll производит поиск файла dll, его загрузку и вызывает специальную функцию, которую библиотека должна экспортировать. Эта функция должна вернуть адрес массива структур с описанием поддерживаемых модулем функций. Массив структур должен оканчиваться структурой с нулевыми указателями, это считается признаком окончания описания. Модуль может описать несколько поддерживаемых им функций, и описание всех их должно присутствовать в возвращаемом массиве описаний. Функция получения описаний вызывается однократно, и указатель должен указывать на область памяти, доступную в

течение всей жизни модуля ZDLL. Эта память может быть как статической, так и динамически выделенной. Во втором случае модуль должен принять меры по освобождению памяти при выгрузке dll.

Описание одной поддерживаемой функции состоит из двух полей - указатель на функцию, по которой она будет вызываться с предопределённым прототипом, и указатель на строку с именем функции, как она будет доступна в среде исполнения MiniM. Строка с именем функции может содержать любые символы. Среда исполнения MiniM ищет функции по их именам нечувствительно к регистру.

Одна функция, указатель которой указан в описании функций, должна принимать, четыре параметра. Это указатель на структуру из функций обратного вызова и указатель на буфер для результата выполнения. Функция $zdll перед вызовом формирует этот результат как пустую строку. В любом случае функция $zdll что-то возвращает. Третий параметр принимает число аргументов передаваемых функцией $zdll и четвертый аргумент это массив значений передаваемых аргументов. Таким образом, одна и та же функция может принимать переменное число аргументов.

Кроме двух обязательных параметров функция модуля должна принимать определённое программистом количество указателей на структуры, через которые система MiniM передаёт все остальные значения, полученные функцией $zdll.

При работе функции модуля ZDLL она может обращаться к среде исполнения MiniM для 1) вычисления выражения на языке M, 2) выполнения последовательности операторов языка M, как если бы они были выполнены командой xecute и 3) обращаться к функциям и подпрограммам с прямой передачей аргументов. Для этого требуется сформировать строку с выражением и передать её функциям, предоставленным указателями первого аргумента.

Все передаваемые данные между системой исполнения MiniM и модулем ZDLL передаются структурой, описанной в zdll.h как MINIM_STR. Эта структура, первый элемент которой указывает (len или type), что следует далее в последовательности байт data. Если значение len больше нуля, то считается, что далее идёт строка на указанную длину. Иначе, если в поле len указано одно из специальных значений (MT_INT32, MT_INT64 или MT_DOUBLE), то далее в байтовой последовательности data находятся бинарные представления чисел, соответственно либо int32, либо int64, либо double. Порядок следования байт как у процессора Интел.

Для конвертации, чтения и записи нестроковых значений система MiniM предоставляет набор функций GetDouble, GetInt32, GetInt64, GetStr, SetDouble, SetInt32, SetInt64. Для формирования строкового значения функция ZDLL должна разместить байты строки в поле data и указать, какое их количество использовано в поле len. Все остальные байты поля data игнорируются. Функции конвертации формата представления возвращают результат вне зависимости от исходного формата. Например, GetInt32 возвращает int32 даже если была указана строка, и GetStr возвращает строковое представление даже если там была одна из форм чисел.

Предоставляемая функция ErrStr возвращает значение системной переменной $zerror.

В дистрибутиве MiniM распространяется три примера на написание модулей ZDLL. Первый показывает, как принять несколько аргументов из исполняющей среды MiniM и составить возвращаемое значение, вычисленное по каким-то правилам. Второй пример показывает, как обратиться при выполнении функции модуля ZDLL к исполняющей среде MiniM для вычисления значения выражения на языке M. Третий пример показывает, как при выполнении функции выполнить набор команд и изменить состояние исполняющей среды MiniM. Далее проведём разбор примеров по шагам. Предполагается, что программист модуля ZDLL владеет базовыми вещами, как-то функция, указатель на структуру, указатель на функцию, массив, экспорт, сборка dll.

```
int WINAPI DllEntryPoint(HINSTANCE hinst,
  unsigned long reason, void* lpReserved)
```

Эта функция вызывается операционной системой Windows как при загрузке, так и при выгрузке dll, и в ней программист выполняет необходимые ему операции инициализации и деинициализации. При работе этой функции возможности исполняющей системы MiniM недоступны.

```
#ifdef __cplusplus
extern "C"
#endif
__declspec( dllexport ) ZDLLFUNC* __stdcall ZDLL ( void)
{
  return functions;
};
```

Здесь экспортируется функция, имеющая предопределённый прототип и имя. Её будет искать исполняющая система MiniM. Библиотека dll может экспортировать также иные функции на своё усмотрение, но их система MiniM искать не будет. Если библиотека не экспортирует функцию ZDLL или экспортирует некорректно, то работать этот ZDLL модуль не сможет. В примере функция ZDLL возвращает указатель на массив структур описаний, поддерживаемых модулем функций (functions). В примере используется размещение массива в сегменте данных. Кроме того, массив может быть размещён в динамической памяти, которую модуль должен, впоследствии, самостоятельно освободить. Разумеется, массив структур должен быть доступен после возврата управления из функции ZDLL, и нельзя возвращать указатель на стек.

```
ZDLLFUNC functions[] =
{
  { concat, "concat"},
  { NULL, NULL }
};
```

Здесь описывается массив структур, первый элемент даёт указатель на функцию исполнения и имя функции, как она будет видна из исполняющей среды MiniM. Второй элемент содержит нулевые значения и является признаком конца описания. Такой массив описаний функций может описывать несколько функций, но последний элемент массива обязательно должен быть нулевым.

```
int __stdcall concat( ZDLLCB* cbfunc, MINIM_STR* result,
  int argc, MINIM_STR** argv)
```

Указатель на эту функцию используется в описании поддерживаемых модулем ZDLL функций, и она может быть вызвана из исполняющей среды MiniM по имени "concat", как было указано во втором поле структуры её описания. Совпадение строки с именем функции и с именем функции, заданным программистом, необязательно. Функция ожидает приём четырех аргументов. Исполняющая система MiniM при вызове функции передает количество параметров указанных в вызове $zdll и массив указателей на значения параметров переданных после имени функции.

```
  MINIM_STR a;
  MINIM_STR b;
```

Заводятся промежуточные временные структуры для конвертации данных в строки. Поскольку система MiniM никогда не гарантирует, что будут предоставлены данные именно в строковом или в одном из числовых видов. Если принимающая сторона должна оперировать определённым представлением данных, то она должна самостоятельно гарантировать, что данные будут конвертированы.

```
cbfunc->GetStr( argv[ 0], &a);
cbfunc->GetStr( argv[ 1], &b);
```

Здесь обращаемся к исполняющей среде MiniM, чтобы она провела конвертацию из возможно числовых значений в строковые. Если значения были строковыми, то они так строковыми и останутся.

```
if( a.len + b.len > MINIM_STR_MAX)
{
  return 1;
}
```

Производится проверка, что получающийся результат не выходит за допустимые пределы. В этом случае возвращается значение, не равное нулю. Получив такое значение, функция $zdll генерирует ошибку <FUNCTION>. Функция ZDLL модуля не должна обращаться к памяти за пределами предоставленных извне структур, так как это может привести к сбою процесса MiniM и к аварийному окончанию его работы. Данные процесса и его блокировки не останутся в неизвестном состоянии, и к ним будет применён механизм очистки, даже если dll терминирует текущий процесс Windows, но дальнейшая работа процесса может оказаться невозможной.

```
memcpy( result->data, a.data, a.len);
memcpy( result->data + a.len, b.data, b.len);
```

Производится формирование последовательности байт результата. Сначала копируются байты первого аргумента, потом - второго.

```
result->len = a.len + b.len;
```

Производится установка, сколько байт используется в значении результата.

```
  return 0;
```

Возвращается нулевое значение как индикатор, что выполнение функции завершилось успешно.

Функция первого примера может быть вызвана из исполняющей среды MiniM, как указано в файле exam1.rsa:

```
$zdll("call","exam1.dll","concat",1234,"abcd")
```

Здесь специально демонстрируется, что передаваемые значения аргументов могут быть сформированы как нестроковые значения, для чего и использовалась принудительная конвертация. С другой стороны, необходимо ли будет использовать конвертацию данных или нет - решение полностью предоставляется программисту ZDLL модуля.

Далее разберём второй пример, но опустим элементы, разобранные ранее, такие как экспорт функции ZDLL и формирование описаний функций.

```
int __stdcall getdate( ZDLLCB* cbfunc, MINIM_STR* result,
  int argc, MINIM_STR** argv)
```

Описывается функция, в действительности не использующая никаких дополнительных аргументов.

```
  MINIM_STR expr;
```

Заводится временная структура для формирования обращения к исполняющей среде MiniM.

```
  expr.len = sprintf( expr.data, "$zd($h,13)");
```

В поле строки выражения прописывается выражение на языке M, в поле длины строки прописывается используемое количество байт.

```
  cbfunc->Eval( &expr, result);
```

Производится обращение к исполняющей среде MiniM, передаётся подготовленное для вычисления выражение, и в качестве ответа указывается переменная result, чтобы функция Eval сразу прописала результат в возвращаемое функцией getdate значение.

```
  return 0;
```

Возвращается нулевое значение как признак успешности выполнения. В реальных случаях применения ZDLL модулей следует проверять, что функция ErrStr возвращает пустую строку, и что при обращении к среде MiniM не произошло ошибок. Проверку лучше производить после каждого обращения к исполняющей среде MiniM. Либо использовать значение возвращаемое функциями Eval и Execute. Если оно равно ZDLL_CALLBACK_DONE, то выполнение закончилось полностью успешно. Иначе возвращается значение ZDLL_CALLBACK_SYNTAX, ZDLL_CALLBACK
или ZDLL_CALLBACK_UNDEFINED. Если переданы некорректные параметры, то возвращается значение ZDLL_CALLBACK_PARAMETERS.

В рутине exam3.rsa содержится пример обращения к ZDLL модулю:

```
$zdll("call","exam2.dll","getdate")
```

Здесь не передаётся никаких дополнительных параметров.

Далее разберём третий пример, но опустим элементы, разобранные ранее, такие как экспорт функции ZDLL и формирование описаний функций.

```
int __stdcall set( ZDLLCB* cbfunc, MINIM_STR* result,
  int argc, MINIM_STR** argv)
```

Эта функция, предназначенная для вызова из MiniM, не принимает никаких дополнительных параметров.

```
  MINIM_STR str;
```

Временные данные для формирования запроса.

```
  str.len = sprintf( str.data, "s a(1)=1");
```

Формируется строка, которую требуется выполнить как последовательность команд с аргументами. В байтовую последовательность data прописывается последовательность байт строки, в поле len прописывается количество используемых байт.

```
  cbfunc->Execute( &str);
```

Сформированная для выполнения строка отправляется на выполнение системой исполнения MiniM.

Нужно отметить для программистов модулей ZDLL, что система MiniM, вызывая функции в dll, предоставляет под переменную result место на стеке, поэтому при рекурсивных вызовах MiniM -> ZDLL -> MiniM -> ZDLL можно формировать результаты вызовов функций zdll независимо друг от друга.

Выполняя функцию обратного вызова Execute, система MiniM создаёт каждый раз новый уровень стека с контекстом xecute. Поэтому, выполненные функцией Execute() команды new, после возврата из функции Execute(), отменяются, и уровень стека восстанавливается.

Для функций обратного вызова UserFunc, UserDo, ReadLocal, WriteLocal, ReadGlobal, WriteGlobal, KillLocal, KillGlobal, которые могут быть названы группой функций обратного вызова нижнего уровня, новый уровень стека не создается. Эти функции возвращают коды ошибок перечисленные макросами

```
#define ZDLL_CALLBACK_DONE        ( 0 )
#define ZDLL_CALLBACK_SYNTAX      ( 1 )
#define ZDLL_CALLBACK_PARAMETERS  ( 2 )
#define ZDLL_CALLBACK_ARGC        ( 3 )
#define ZDLL_CALLBACK_UNDEFINED   ( 4 )
#define ZDLL_CALLBACK_ERROR       ( 5 )
#define ZDLL_CALLBACK_HALT        ( 6 )
```

Значения функций:

UserFunc	Вызывает пользовательскую функцию на M, ожидая возврат.
UserDo	Вызывает пользовательскую подпрограмму на M, не ожидая возврат.
ReadLocal	Читает значение локальной переменной.
WriteLocal	Записывает значение локальной переменной.
ReadGlobal	Читает значение глобальной переменной.
WriteGlobal	Записывает значение глобальной переменной.
KillLocal	Удаляет локальную переменную вместе с ее подъиндексами.
KillGlobal	Удаляет глобальную переменную вместе с ее подъиндексами.

SetTEST	Возвращает текушее значение системной переменной $test и устанавливает новое значение
OrderLocal	Применяет функцию $order к локальной переменной и возвращает следующее значение ключа
OrderGlobal	Применяет функцию $order к глобальной переменной и возвращает следующее значение ключа
IncLocal	Увеличивает на единицу значение локальной переменной
IncGlobal	Увеличивает на единицу значение глобальной переменной
DataLocal	Применяет функцию $data к локальной переменной и возвращает индикатор существования
DataGlobal	Применяет функцию $data к глобальной переменной и возвращает индикатор существования

Значения возвратов:

ZDLL_CALLBACK_DONE - Действие выполнено успешно.

ZDLL_CALLBACK_SYNTAX - Один из аргументов функции приводит к синтаксической ошибке.

ZDLL_CALLBACK_PARAMETERS - Переданные параметры функции имеют недопустимые значения.

ZDLL_CALLBACK_ARGC - Указано недопустимое значение количества индексов или аргументов.

ZDLL_CALLBACK_UNDEFINED - При чтении обнаружилось что указанная переменная имеет неопределенное значение.

ZDLL_CALLBACK_ERROR - При выполнении функции произошла ошибка в процессе или в базе данных.

ZDLL_CALLBACK_HALT - При выполнении кода в контексте MiniMono была вызвана команда halt, приложение должно завершить работу самостоятельно и выгрузить библиотеку MiniMono.dll

Для вызова функции UserFunc нужно в качестве строки с именем пользовательской функции на M передать строку начинающуюся на символы "$$". Например, вызов пользовательской функции $$func^cbfunc с параметром 5555 будет выглядеть так:

```
char* name = "$$func^cbfunc";
MINIM_STR result;
MINIM_STR arg;
```

```
cbfunc->SetInt32( 5555, &arg);
MINIM_STR* argv[ 1];
argv[ 0] = &arg;
cbfunc->UserFunc( name, 1, argv, &result);
```

Для вызова функции UserDo нужно передавать только ее имя. Для функций UserFunc и UserDo указание имени рутины и базы данных рутины необязательно, при их отсутствии используется текущая рутина в текущей базе данных. Например:

```
char* name = "func";
MINIM_STR arg;
cbfunc->SetInt32( 5555, &arg);
MINIM_STR* argv[ 1];
argv[ 0] = &arg;
cbfunc->UserDo( name, 1, argv);
```

Для функций нижнего уровня удвоение кавычек для получения правильного синтаксиса M, в отличие от функций Eval и Execute, в значениях *argv* не требуется, эти значения используются как есть. Значения автоматически нормируются по соглашениям языка M - если указана строка с значением, являющимся строковым каноническим представлением числа, то используется числовой индекс.

Для функций ReadGlobal, WriteGlobal и KillGlobal указывать базу данных необязательно. Если указано значение NULL или пустая строка, то используется текущая база данных. В имени глобала указывать символ глобала циркумфлекс (^) не нужно.

Если процесс обнаруживает, что при чтении ReadLocal или ReadGlobal указанная переменная имеет неопределенное значение, то возвращается значение ZDLL_CALLBACK_UNDEFINED и переменная *result* не содержит значимой информации, иначе в нее записывается значение этой переменной и возвращается значение ZDLL_CALLBACK_DONE. Значение записывается в том виде, в котором оно хранится в базе данных или в локальной переменной. Для приведения ее типа к нужному формату необходимо использование функций из группы GetDouble, GetInt32, GetInt64, GetStr.

Пример чтения и записи глобальной переменной ^GlobalName("abc"):

```
// пишем переменную
```

```
MINIM_STR value;
value.len = sprintf( value.data, "%s", "abcdef");
MINIM_STR index;
index.len = sprintf( index.data, "%s", "abc");
MINIM_STR* argv[ 1];
argv[ 0] = &index;
char* name = "GlobalName";
int ret = cbfunc->WriteGlobal( name,
  NULL, 1, argv, &value);
// читаем переменную
ret = cbfunc->ReadGlobal( name,
  NULL, 1, argv, &value);
```

При установке значения системной переменной $test значение параметра svn_test_value сравнивается с нулем. Если равно нулю, то значение $test устанавливается в 0, иначе в 1.

3.3 Разработка модуля ZDEVICE

MiniM Database Server поддерживает механизм расширения встроенных типов устройств путем подключения устройств описанных в специальных библиотеках динамической компоновки. Эти библиотеки носят общее название z-устройства.

Для создания z-устройства необходимо средство создания Win32 динамически подключаемых библиотек. В примерах в подкаталоге zdevice использован язык C++.

Библиотека dll должна экспортировать одну функцию с определенным прототипом и именем GetZDevice. Эта функция должна вернуть вызывающему процессу структуру с указателями на функции-обработчики операций с устройством. Прототипы этих функций описаны в заголовочном файле zdevice.h.

После того, как устройство описано, к нему можно обращаться как к любому другому устройству MiniM. Для этого необходимо чтобы библиотека dll размещалась в подкаталоге bin инсталляции сервера. Имя типа устройства должно совпадать с именем файла dll без расширения и начинаться на символ Z. Например, при указании типа устройства

```
s devname="|ZDEV1|name"
```

процесс MiniM будет использовать файл zdev1.dll.

Библиотека должна поддержать по крайней мере одну функцию, обработчик открытия устройства. Остальные поддерживаются на усмотрение разработчика z-устройства и являются необязательными. Например, устройство, ориентированное только на чтение, может не поддерживать интерфейсы записи и определения $X и $Y.

После того как файл dll расположен в подкаталоге bin, любой процесс MiniM может открыть и использовать это устройство. Библиотека динамической компоновки загружается в адресное пространство процесса при открытии z-устройства и выгружается при закрытии z-устройства. Если процесс одновременно открывает несколько z-устройств одного типа, то библиотека выгружается при закрытии последнего из них по времени.

Библиотека z-устройства выполняется по схеме, очень похожей на ZDLL модуль и может одновременно реализовать оба интерфейса. При необходимости частого открытия-закрытия z-устройства можно предварительно загрузить библиотеку вызовом $zdll("load"), и по окончании использования освободить через вызов $zdll("unload"), если эта библиотека реализует интерфейс ZDLL модуля. Между этими вызовами библиотека остается загруженной и открытие устройства производится быстрее. Если библиотека реализует интерфейс ZDLL модуля, который ей не требуется, она может экспортировать пустую таблицу функций.

Функциям-обработчикам z-устройства среди параметров передается параметр обратного вызова и данные в формате совпадающем с соглашениями ZDLL модуля. При работе обработчики могут обращаться к контексту процесса и выполнять преобразования данных.

Описание интерфейса

Интерфейс z-устройства определяется структурой данных, состоящей из указателей на функции-обработчики действий с устройством. Функции обработчики имеют как минимум два параметра, это контекст процесса в виде указателя на таблицу функций обратного вызова

```
ZDLLCB* cbfunc
```

и контекст текущего устройства в виде указателя, значение которого определяет программист при создании устройства

```
ZDEVICEHANDLE hDevice
```

Контекст текущего устройства создается при вызове функции открытия устройства. Значение hDevice должно быть ненулевым. В случае если оно нулевое, процесс MiniM считает что устройство не открыто. Процесс MiniM никак не использует значение hDevice кроме того что передает его в функции обработчики. Это значение предназначено для идентификации текущего экземпляра устройства, поскольку библиотека z-устройств может поддерживать несколько открытых устройств одного типа.

Функции обработчики z-устройств должны возвращать код завершения как число. Если этот код равен ZDEV_NO_ERROR, то процесс считает что выполнение произошло успешно. Если этот код соответствует одному из перечисленных, то процесс генерирует ошибку:

Код возврата	**Ошибка**
ZDEV_ERR_READ	<READ>
ZDEV_ERR_WRITE	<WRITE>
ZDEV_ERR_ENDOFFILE	<ENDOFFILE>
ZDEV_ERR_DEVICE	<DEVICE>

Если возвращается иное значение, то процесс использует это значение для конструирования имени ошибки, например если возвращено значение 456, то генерируется ошибка <ZDEVICE456>.

Все обработчики кроме обработчика открытия устройства являются необязательными. В случае если разработчик устройства считает что устройство не поддерживает какие-либо операции, то в структуре описания интерфейса должен стоять нулевой указатель. При обращении к этому нереализованному действию процесс MiniM выполняет действия по умолчанию. Например, если не реализована функция WriteStr, то при выполнении команды записи строки данные утрачиваются и вывод никуда не выполняется, а если не реализован возврат значения $X, то процесс возвращает значение 0.

Обработчики команд Open, Close, Use принимают параметры опций, если они переданы программой на MUMPS. Параметры передаются в виде массива указателей на пары имя + значение и указанием количества таких пар. В массиве имена и значения идут строго парами, последовательно, в порядке их указания в MUMPS, сначала имя, затем значение, затем снова имя и значение и так далее. Если в указанной позиции опция пропущена, то оба указателя, и имя и значение, представлены нулевыми указателями. Если опущено имя опции, то в месте имени подставляется пустой вказатель, если опущено значение опции, то в значении под-

ставляется пустой указатель. Нужно обратить внимание, что количество опций передается как количество пар, и соответствует двойному числу передаваемых указателей. Имена и значения передаются во внутреннем формате MiniM, и для получения строки или числа нужно обратиться к преобразованию в требуемый вид используя контекст процесса cbfunc. Разработчик устройства самостоятельно определяет поведение разрабатываемого устройства при передаче опций.

При использовании на языке MUMPS мнемоник для команд open и use мнемоники обрабатываются процессом MiniM самостоятельно. Для z-устройств по умолчанию нет рутины обработки мнемоник.

Обработчик Open вызывается при открытии еще одного экземпляра устройства. В параметрах передается контекст процесса, опции и имя устройства. Имя устройства это строка следующая после указания типа устройства, например если открывается устройство "|ZDEV1|abcd", то в параметре имени передается строка "abcd" завершающаяся нулем. Дальнейшая идентификация устройства при использовании команд use и close выполняется по полному имени устройства.

Разработчик может использовать имя устройства по своему усмотрению. В каждом процессе MiniM создаваемое устройство существует индивидуально и принадлежит процессу, а не серверу в целом, и разработчик z-устройства должен учитывать, что одновременно два или более процессов могут открыть устройства с одинаковыми именами.

Обработчик Close вызывается при закрытии устройства командой close. После его выполнения устройство закрывается процессом MiniM и процесс освобождает свои внутренние структуры данных.

Обработчик Use вызывается при выполнении команды use и принимает передаваемые опции. На этот момент устройство может как уже быть текущим, так и стать текущим после выполнения обработчика.

Обработчик ReadStr вызывается при выполнении команды read в форме чтения строки. Обработчик принимает значения ограничения длины чтения и таймаут. Если ограничение длины чтения или таймаут не указаны, то передаются значения -1. В качестве значения таймаута процесс MiniM передает значение в миллисекундах, хотя в языке MUMPS указывается в секундах. Декодирование в миллисекунды процесс выполняет автоматически в соответствии с общими правилами работы с таймаутами в MiniM. Обработчик должен вернуть прочитанную строку в параметр

```
MINIM_STR* str
```

Формат возврата (строка или число) определяется разработчиком устройства самостоятельно.

Обработчик ReadChar вызывается при выполнении команды read в форме чтения кода символа. Обработчик принимает таймаут чтения в миллисекундах, или значение -1 если он не был указан. Обработчик должен вернуть код символа в параметр

```
int* result
```

По соглашениям языка MUMPS команда чтения кода символа при неуспехе чтения или при невозможности чтения символа должна вернуть значение -1. В случае успеха должно возвращаться значение в диапазоне от 0 до 255 включительно.

В случае если разработчик z устройства полагает, что команда чтения кода символа должна в соответствии со спецификой устройства вернуть значение из другого диапазона, он обязательно должен описать это отклонение от стандарта в документации на свое новое устройство.

Обработчики ReadStr и ReadChar при обнаружении что выполняется чтение за пределами допустимых данных должны вернуть значение ZDEV_ERR_ENDOFFILE. В этом случае процесс MiniM анализирует текущее состояние процесса и либо игнорирует возврат и программа на MUMPS должна проверить значение $zeof, либо генерирует ошибку <ENDOFFILE>.

Разработчик z-устройства должен понимать, что команда чтения read может приводить к записи в устройство. К таким формам команды чтения относятся формы

```
read "any string"
read ?expr
read !
read #
```

и для поддержки таких форм требуется реализация обработчиков записи, иначе вывод утрачивается. Форма команды чтения с передачей ей числа является синтаксически недопустимой в стандарте языка MUMPS. Например:

```
USER>r 123

<SYNTAX> :READ: *r 123
```

Обработчик WriteStr вызывается при необходимости выполнить вывод в устройство строки. Процесс MiniM передает данные в параметре

```
MINIM_STR* str
```

в своем внутреннем формате. Обработчик должен приводить данные к необходимому ему виду.

Обработчик WriteChar вызывается при необходимости вывести код символа. В качестве кода может быть указано и отрицательное значение. Как трактовать выводимый код разработчик устройства должен решить самостоятельно.

Обработчик WriteNL вызывается при необходимости выполнить вывод в устройство перевода строки (New Line). Как трактовать это действие разработчик устройства должен решить самостоятельно.

Обработчик WriteFF вызывается при необходимости выполнить вывод в устройство перевода страницы (Form Feed). Как трактовать это действие разработчик устройства должен решить самостоятельно.

Обработчик WriteTab вызывается при необходимости выполнить вывод табулирования. В параметре указывается число передаваемое в команде read ?expr или write ?expr. Значение вычисляется как целое число. Как трактовать это действие разработчик устройства должен решить самостоятельно.

Обработчики GetX и GetY должны быть реализованы если устройство поддерживает трактовку позиций переменных $X и $Y. Характер понимания этих значений оставляется на усмотрение разработчика устройства.

Обработчики SetX и SetY должны быть реализованы если устройство поддерживает присваивание системным переменным $X и $Y. Характер понимания этих значений оставляется на усмотрение разработчика устройства.

Обработчики GetKEY и SetKEY должны быть реализованы если устройство поддерживает работу с системной переменной $KEY и ее присваивание. Характер понимания этого значения оставляется на усмотрение разработчика устройства.

Обработчик GetZEOF должен быть реализован если устройство поддерживает системную переменную $ZEOF. Работа этого обработчика должна быть согласована с возвратом значения ZDEV_ERR_ENDOFFILE

из обработчиков ReadStr и ReadChar для согласованной работы устройства с точки зрения программиста MUMPS.

Обработчики GetZA и GetZB должны быть реализованы если устройство поддерживает работу с системными переменными $za и $zb соответственно. Значение этих переменных определяется разработчиком устройства.

Обработчики системных переменных GetKEY, GetZA и GetZB вызываются при каждом чтении системных переменных $key, $za и $zb из программы на языке MUMPS. Процесс MiniM эти значения не кеширует.

Пример ZDEV1

В качестве примера z-устройства в подкаталоге zdevice/example1 приводится z-устройство ZDEV1. Оно выполняет генерацию случайной последовательности символов при открытии, после чего поддерживает чтение их этой последовательности с использованием команд чтения строки и кода символа. Устройство выполнено как устройство только на чтение.

В качестве примера использования опций команды use устройство поддерживает опцию /POS, например

```
use dev:/POS=123
```

для позиционирования в последовательности на указанную позицию.

Для иллюстрации и тестирования z-устройства приводится рутина ZDEV1, в которой выполняются действия по чтению одной строки (readstr), чтение строки с указанием ограничения на длину (readlimit), чтение полностью с генерацией ошибки <ENDOFFILE> (readtofail), чтение полностью с определением конца чтения через системную переменную $ZEOF (readtoeof) и повторное чтение сначала после использования позиционирования (doubleread).

```
run ; k  d run^ZDEV1 w
 d readstr
 d readlimit
 d readtofail
 d readtoeof
 d doubleread
 q
readstr
 ; read one string
```

```
 n dev="|ZDEV1|123",str
 o dev u dev r str u $p c dev
 w "str is ",$l(str)," length",!
 q
readlimit
 ; read one limited string
 n dev="|ZDEV1|123",str,limit=1024
 o dev u dev r str#limit u $p c dev
 w "str is ",$l(str)," length",!
 q
readtofail
 ; read all to <ENDOFFILE> error
 n $et="g err"
 n dev="|ZDEV1|123",str
 o dev u dev f  r str
 u $p c dev
 q
err
 u $p
 w "reached ",$ze," error",!
 s $ec=""
 c dev
 q
readtoeof
 ; read all to $zeof indicator
 n dev="|ZDEV1|123",str,zeof=$v("proc",5,0)
 o dev u dev f  r str q:$zeof
 u $p c dev
 i $v("proc",5,zeof)
 w "reached $zeof",!
 q
doubleread
 ; read one string, rewind and read again
 n dev="|ZDEV1|123",str1,str2
 o dev u dev r str1 u dev:/POS=0 r str2
 u $p c dev
 w "strings compared: ",(str1=str2),!
 q
```

Пример получения вывода при работе этого примера:

```
USER>k  d run^ZDEV1 w
str is 32767 length
str is 1024 length
reached <ENDOFFILE> :readtofail+4^ZDEV1: error
reached $zeof
strings compared: 1
```

Пример ZDEV2

В качестве примера z-устройства в подкаталоге zdevice/example2 приводится z-устройство ZDEV2. Устройство выполняет вывод данных в отладочный порт Windows используя функцию OutputDebugString. При запуске монитора отладочного порта разработчик может получить информацию о том, какой обработчик выполнился и как переданы параметры. Устройство выполнено как устройство только на запись.

Используя это устройство, разработчик может проверить, что при выполнении команд чтения в форме вывода выполняется вызов соответствующих обработчиков записи.

```
run ; k  d run^ZDEV2 w
 n dev="|ZDEV2|123"
 o dev u dev
 w 67
 w *67
 w !
 w #
 w ?67
 u $p
 c dev
 q
```

Пример получения вывода этого примера в отладочный порт с использованием утилиты Microsoft dbmon.exe:

```
600: ZDEV2:WriteStr: 67
600: ZDEV2:WriteChar: C (67)
600: ZDEV2:WriteNL
600: ZDEV2:WriteFF
600: ZDEV2:WriteTab 67
```

3.4 Пользовательские z-функции

Сервер MiniM поддерживает механизм пользовательских z-функций и пользовательских z-переменных. Оба эти понятия далее описаны как z-функции, поскольку вызов пользовательской переменной сервер MiniM приводит к вызову пользовательской функции с нулевым количеством аргументов.

Пользовательской z-функцией называется функция, имя которой начинается на символ '$' за которым следует символ z, за которым следует последовательность латинских букв или цифры. Если после имени следует открывающая скобка, то ожидаются параметры и это пользовательская z-функция, иначе это пользовательская z-переменная.

Реализация z-функции может быть помещена программистом в рутину, имя которой начинается на символы "%ZFUNC". После последовательности символов "%ZFUNC" может не стоять других или стоять произвольная последовательность с допустимым именем рутины.

Вызов пользовательской z-функции производится без указания рутины, например

```
w $zmyfunc(123,456)
s a=$zmyext(.b)
```

Имена пользовательских z-функций не должны повторять зарезервированные внутренние z-функции MiniM. В этом случае MiniM использует их, а не пользовательские.

В рутинах %ZFUNCXXX для используемой z-функции должна быть метка с соответствующим количеством формальных параметров. Имя метки должно быть в верхнем регистре. Процесс MiniM при вызове функции ищет рутину начинающуюся на символы %ZFUNC в порядке их индексной сортировки. Используется та рутина, которая содержит нужную метку в верхнем регистре и которая была найдена первой. В случае, если есть две такие рутины с одинаковыми метками, будет использована первая из них.

Пример. Пусть есть рутина %ZFUNC001 такого содержания:

```
ZZVAR
 q 123
ZZFUNC(param)
 q param
```

тогда к этим z-переменной и z-функции можно обратиться:

```
USER>w $zzVar
123
USER>w $zzFunc(789)
789
```

При исполнении пользовательской z-функции используется только ее скомпилированный вариант, байткод. При отсутствии подходящего байткода, даже если есть исходный код, генерируется ошибка <NOZROUTINE>. Если в рутинах %ZFUNCXXX нет соответствующей метки в верхнем регистре, то генерируется ошибка <NOZROUTINE>.

Передача параметров z-функции производится также как и другим пользовательским функциям. Возможно использовать прием аргументов со значением по умолчанию и переменное количество аргументов.

3.5 Пользовательские z-команды

Сервер MiniM поддерживает механизм пользовательских z-команд. Пользовательской z-командой называется команда имя которой начинается на символ z или Z и реализация которой описана в рутине %ZCMDXXX.

Для пользовательской z-команды может быть использована как аргументная форма, так и безаргументная. Для аргументной формы поддерживается только одноаргументная. В случае вызова пользовательской z-команды с перечислением аргументов через запятую MiniM применяет пользовательское определение к каждому указанному аргументу последовательно слева направо.

Для пользовательской z-команды может быть указано постусловие. В этом случае постусловие вычисляется перед выполнением команд. В случае если значение постусловия после вычисления приводится к нулю, то все аргументы этой z-команды пропускаются и выполнение переходит на следующую в этой строке команду.

Реализация пользовательской z-команды должна быть описана в рутине с именем %ZCMDXXX, где в качестве XXX используется любая последовательность символов, образующая корректное имя рутины или отсутствовать.

Каждой пользовательской z-команде в одной из рутин %ZCMDXXX должна соответствовать метка в верхнем регистре. При выполнении

команды процесс MiniM отыскивает такую рутину в порядке индексного следования имен. В случае если не найдено ни одной рутины %ZCMDXXX в которой есть метка с именем команды в верхнем регистре генерируется ошибка <NOZROUTINE>.

Имена пользовательских z-команд не должны повторять встроенные в MiniM z-команды. В случае если они встречаются, они выполняются как встроенные и их нельзя переопределить.

Имя метки должно точно соответствовать имени z-команды и быть в верхнем регистре. Если команда zmycmd, то метка должна быть ZMYCMD. Если команда zzshow, то метка должна быть ZZSHOW. Использовать пользовательские z-команды можно нечувствительно к регистру.

Пример. Пусть есть рутина %ZCMD001 такого содержания:

```
ZCMD0
 w 123,!
ZCMD1(param)
 w param,!
```

тогда можно использовать команду zcmd0 без аргументов и команду zcmd1 с одним аргументом. При вызове zcmd1 можно перечислить несколько аргументов через запятую, в этом случае производится последовательность вызовов метки ZCMD1^%ZCMD001 для каждого указанного аргумента.

Передача параметров пользовательской z-команде производится всегда только одного параметра и только по значению.

В качестве аргументов пользовательских z-команд можно указывать произвольные вычисляемые выражения.

3.6 Аккаунты процессов

Для корректной настройки и, как следствие, работы сервера MiniM требуется точно понимать политику сервера в отношении аккаунтов процессов сервера.

В состав процессов сервера MiniM входят процессы сервиса (mnmsvc.exe), демон журнала (minimjd.exe), демон записи (minimwd.exe), демон расширения баз (minimed.exe), контроллер сервера (minimti.exe) и процессы

исполнения job (minim.exe). Контроллер сервера не является необходимым для работы сервера и служит для утилитарных целей - запуск, останов сервера и запуск вспомогательных программ.

Основным аккаунтом, с которым требуется работать администратору сервера, является аккаунт сервиса. Сервис запускает дочерние процессы демонов, процесс автостарта и процессы обслуживания телнет клиентов. Все запущенные им процессы принадлежат его аккаунту. Если процесс minim.exe запускает дочерний job, то аккаунт запускаемого процесса тот же что и аккаунт родительского. Кроме того, могут быть запущены процессы minim.exe как консольные или с устройством по умолчанию |STD|. В этом случае процесс запускается под текущим аккаунтом запускающего.

Если из процесса автостарта производится запуск дочернего процесса, то он, соответственно, также запускается под аккаунтом сервиса.

В большинстве случаев понимание политики аккаунтов некритично к работе сервера, но есть ситуации в которых ошибка в понимании может привести к проблеме работы сервера. В первую очередь - это 1) использование сетевых ресурсов, создаваемых для отдельного пользовательского аккаунта и 2) продолжение работы при logoff пользователя.

При окончании работы пользовательской сессии аккаунт пользователя прекращает работу, поэтому прекращают работу все процессы, запущенные от его имени. Если был запущен сервис от этого аккаунта, то сервер будет остановлен. Если от этого аккаунта был запущен консольный вариант minim.exe, то процесс будет остановлен. Если из такого процесса был запущен фоновый процесс, то он будет остановлен. Для того, чтобы после logoff пользователя необходимые для работы сервера процессы продолжали работу, нужно запускать фоновые процессы через телнет-клиент, и настроить аккаунт запуска сервера от системного пользователя.

В версиях Windows2000 и WindowsNT автоматический останов сервера не предусмотрен и корректная работа сервера при logoff пользователя в настоящей версии не гарантируется. В этих версиях требуется перезапуск сервера MiniM или настройка сервиса для работы под системной учетной записью. Для корректной работы телнет-доступа требуется настройка разрешения взаимодействия сервиса с рабочим столом.

Другим вариантом является применение специальных средств запуска процессов. Например, можно запустить процесс используя Windows-проводник. Выбрать файл minim.exe, нажать клавишу Shift и не отпус-

кая, нажать правую кнопку мыши. В всплывающем меню выбрать пункт "Запуск от имени...", и ввести имя и пароль нужного пользователя.

Для корректной работы сервера с сетевыми ресурсами рекомендуется создать отдельного пользователя для сервера MiniM и для него настроить права доступа. После чего следует настроить запуск сервиса MiniM от этого пользователя.

Для корректной настройки доступа к иным ресурсам также следует помнить о различии аккаунтов. Например, при доступе к ODBC DSN нужно понимать, что есть два вида DSN - User и System. При настройке User DSN эти ODBC DSN будут видны процессам текущего пользовательского аккаунта, при настройке System DSN эти ODBC DSN будут видны всем процессам. Также требуется проверка авторизации при доступе через иные аккаунты. Например, некоторые СУБД имеют режим автоматической авторизации, если логин происходит от имени определенного пользователя. При этом программа может использовать System DSN работая в совершенно ином аккаунте. Все эти детали требуется обязательно проверять и корректно настраивать для всех случаев, которые будут работать в реальной инсталляции.

3.7 Клавишные комбинации редактора рутин

Up	Перемещение на строку вверх.
Shift+Up	Перемещение на строку вверх с выделением.
Ctrl+Up	Прокрутка вверх.
Down	Перемещение на строку вниз.
Shift+Down	Перемещение на строку вниз с выделением.
Ctrl+Down	Прокрутка вниз.
Left	Перемещение на символ влево.
Shift+Left	Перемещение на символ влево с выделением.
Ctrl+Left	Перемещение на слово влево.
Shift+Ctrl+Left	Перемещение на слово влево с выделением.
Right	Перемещение на символ вправо.
Shift+Right	Перемещение на символ вправо с выделением.
Ctrl+Right	Перемещение на слово вправо.
Shift+Ctrl+Right	Перемещение на слово вправо с выделением.
PgDn	Перемещение на страницу вниз.
Shift+PgDn	Перемещение на страницу вниз с выделением.
Ctrl+PgDn	Перемещение в конец текущей страницы.

Shift+Ctrl+PgDn	Перемещение в конец текущей страницы с выделением.
PgUp	Перемещение на страницу вверх.
Shift+PgUp	Перемещение на страницу вверх с выделением.
Ctrl+PgUp	Перемещение в начало текущей страницы.
Shift+Ctrl+PgUp	Перемещение в начало текущей страницы с выделением.
Home	Переход в начало страницы.
Shift+Home	Переход в начало страницы с выделением.
Ctrl+Home	Переход в начало текста.
Shift+Ctrl+Home	Переход в начало текста с выделением.
End	Переход в конец строки.
Shift+End	Переход в конец строки с выделением.
Ctrl+End	Переход в конец текста.
Shift+Ctrl+End	Переход в конец текста с выделением.
Ins	Переключение режима вода текста - вставка или перезапись.
Ctrl+Ins, Ctrl+C	Копировать выделенный текст в клипборд.
Shift+Del, Ctrl+X	Вырезать в клипборд.
Shift+Ins, Ctrl+V	Вставить из клипборда.
Del	Удалить символ справа от каретки.
BkSp, Shift+BkSp	Удалить символ слева от каретки.
Ctrl+BkSp	Удалить слово слева от каретки.
Alt+BkSp, Ctrl+Z	Отмена последнего редактирования. (Undo)
Shift+Alt+BkSp, Shift+Ctrl+Z	Повтор последующего редактирования. (Redo)
Enter, Shift+Enter, Ctrl+M	Вставить строку.
Tab	Вставить символ табуляции.
Shift+Tab	Удалить влево до предыдущей табуляции.
Ctrl+A	Выделить весь текст.
Shift+Ctrl+I	Втяжка блока вправо.
Shift+Ctrl+U	Втяжка блока влево.
Ctrl+N	Открыть новый редактор с пустым текстом.
Ctrl+O	Открыть новый редактор с выбором рутины.

Ctrl+T	Удалить текущее слово.
Ctrl+Y	Удалить текущую строку.
Shift+Ctrl+Y	Удалить до конца строки.
Ctrl+0, ... Ctrl+9	Перейти на маркер с этим номером.
Shift+Ctrl+0, ... Shift+Ctrl+9	Установить маркер с этим номером на текущую позицию.
Ctrl+U	Выделенный текст поднять в верхний регистр.
Ctrl+L	Выделенный текст опустить в нижний регистр.
Shift+Ctrl+B	Перейти к парной круглой скобке.
Shift+Ctrl+N	Переключить режим выделения в нормальный.
Shift+Ctrl+C	Переключить режим выделения в колоночный.
Shift+Ctrl+L	Переключить режим выделения в строчный.
Ctrl+F9	Компилировать текущую ретину.
F9	Продолжить выполнение отлаживаемого процесса.
Ctrl+F2	Завершить выполнение отлаживаемого процесса.
F5	Установить или снять точку останова для отладчика.
Ctrl+Alt+B	Показать список текущих установленных точек останова.
Ctrl+Alt+S	Показать окно со стеком отлаживаемого процесса.
Ctrl+Alt+V	Показать текущее состояние системных переменных отлаживаемого процесса.
Ctrl+Alt+W	Показать список вычисляемых выражений в отлаживаемом процессе.
Ctrl+F5	Добавить вычисляемое выражение.
F8	Продолжить выполнение процесса на одну строку на том же уровне стека.
F7	Продолжить выполнение процесса на одну строку включая вложенные уровни стека.
F4	Продолжить выполнение процесса до достижения текущей строки или другой точки останова.
F6	Переключить в парный текст (INT -> MAC или MAC -> INT)

3.8 MiniM Server Connect

Для обращения к СУБД MiniM из сторонних программ предоставляется dll модуль minimsc.dll. Модуль размещается в каталоге разрабатываемой клиентской программы или в каталоге находящемся в путях PATH на клиентском компьютере.

Основным интерфейсом MiniM Server Connect является интерфейс на языке C. Примеры обращения из других сред исполнения и языков программирования даются в качестве демонстрационных и вспомогательных. В реальном применении minimsc.dll программист должен самостоятельно выбрать способ применения модуля в соответствии с применяемой в его программе дисциплиной передачи данных и обработки ошибок. При применении юникодовой версии Delphi программисту следует внимательно отнестись к применяемым в демонстрационных примерах способам кодирования, поскольку примеры даются в ANSI версии Delphi.

При применении модуля MiniM Server Connect программисту также следует учитывать применяемую на сервере кодировку символов и обеспечить согласованность кодировок сервера и клиента в случае использования национальных алфавитов.

Модуль MiniM Server Connect может вызываться клиентской программой как статически так и динамически. В комплекте MiniM даются примеры доступа из программ на языках C++ и ObjectPascal (среда разработки Delphi) с сборкой в статическом варианте. Выбор варианта вызова (статический или динамический) оставляется на усмотрение разработчика клиентской программы.

Библиотека minimsc.dll предназначена для создания программного объекта соединения с СУБД MiniM и соединение обслуживается по протоколу поддерживаемому на сервере рутиной ^%srv. Для работы соединения сервер обслуживания должен быть запущен командой

```
d ^%srv
```

При инсталляции сервера по умолчанию эта команда запускается автоматически при старте сервера из файла autostart.m. Протокол передачи данных не шифруется. В протокол не входит соглашение о логине пользователя. Полагается, что в прикладной программе средства идентификации пользователя обеспечиваются средствами и по соглашениям этой прикладной программы.

Программный интерфейс к модулю minimsc.dll описан в заголовочном файле minimsc.h для языка C++, в интерфейсном файле minimscint.pas для языка ObjectPascal, в модуле minimsc.vb для языка VB.NET, в модуле minimsc.cs для языка C#, для MiniM и для Cache на языке MUMPS и в модуле minimscj.java + minimscj.dll для языка Java в формате JNI/Win32. Для языка Perl в реализации ActivePerl интерфейс дается непосредственно в файле примера.

Программный интерфейс выполнен в виде нескольких функций в стандартных соглашениях о передаче параметров, применяемых в Windows.

Для идентификации соединения библиотека создает объект соединения, идентифицируемый хендлом HMNMConnect. Объект создается функцией MNMCreateConnect и удаляется функцией MNMDestroyConnect. Объект создается в несоединенном состоянии. Для соединения используется функция MNMConnectOpen и для отсоединения используется функция MNMConnectClose.

Все функции работы с объектом соединения возвращают число типа целое со знаком. В случае успешного выполнения функции возвращается значение 1, при неуспехе возвращается значение 0. При возврате значения 0 можно получить описание произошедшей ошибки с помощью функции MNMGetLastError.

Значения в объект соединения и из него передаются в виде структуры, содержащей как количество значимых байт так и сами значимые байты. В значимых байтах могут присутствовать произвольные байты. Рассматриваться как содержательные могут только байты, количество который указано в поле len. Для приема выходных значений инициализация структуры не требуется. Программа, обращающаяся к соединению, может использовать функции, оперирующие строками завершающимися нулем, но завершающий нуль функции соединения на рассматривают и используют только значимое количество байт и при формировании выходного значения завершающий нулевой байт также не устанавливается.

К основным операциям с соединением относятся операции чтения значения вычисляемого выражения, выполнение одной или последотвательности команд, запись значения в переменную и удаление переменной.

Функция

```
int MNMSCPROC MNMRead( HMNMConnect pConnect,
  MINIMSTR* Expression,
  MINIMSTR* Result);
```

В значимых байтах Expression должно быть указано выражение, значение которого требуется вычислить, в структуру Result возвращается значение выражения после вычисления.

Функция

```
int MNMSCPROC MNMWrite( HMNMConnect pConnect,
  MINIMSTR* VarName,
  MINIMSTR* VarValue);
```

В значимых байтах VarName указывается имя переменной, которой требуется присвоить значение значимых байт из структуры VarValue.

Функция

```
int MNMSCPROC MNMExecute( HMNMConnect pConnect,
  MINIMSTR* Commands);
```

Функция выполняет последовательность команд, указанную в значимых байтах структуры Commands.

Функция

```
int MNMSCPROC MNMKill( HMNMConnect pConnect,
  MINIMSTR* VarName);
```

Выполняет удаление переменной, имя которой указано в значимых байтах структуры VarName.

Функция

```
int MNMSCPROC MNMExecuteOutput( HMNMConnect pConnect,
  MINIMSTR* Commands);
```

Выполняет последовательность команд, указанную в значимых байтах структуры Commands, но при этом в случае, если выполнение команд привело к выводу данных в текущее устройство, вызывается функция обратного вызова установленная вызовом функции MNMSetOutput. Порция передачи не определена по длине или по значению терминатора, передается то количество байт, которое было передано от сервера на сторону соединения. Если программа на сервере вызывает вывод в текущее устройство нескольких строк, то функция обратного вызова может быть вызвана на каждую выводимую строку, на все выводимые строки или даже на часть каждой из выводимых строк. Команды вывода в текущее устройство в этом случае не должны вывести в текущее устройство

нулевой байт, это служит индикатором окончания выполнения. Традиционное применение выполнения команд с получением вывода в текущее устройство это, например, выполнение компиляции и получение отчета о компиляции.

Функции

```
int MNMSCPROC MNMListGet( MINIMSTR* List,
  int pos, MINIMSTR* Element);

int MNMSCPROC MNMListSet( MINIMSTR* List,
  int pos, MINIMSTR* Element);

int MNMSCPROC MNMListLength( MINIMSTR* List);
```

являются сервисными, для упаковки и распаковки списковой структуры. Для работы с ними соединение или его активация необязательны. Функция MNMListGet возвращает в переменную Element значение элемента списка List из позиции pos. Функция MNMListSet записывает в списке List в позиции pos значение Element. Функция MNMListLength возвращает количество элементов списка List.

Функция

```
int MNMSCPROC MNMText( MINIMSTR* Source,
  MINIMSTR* Target);
```

заменяет последовательность байт заданную в Source на строку, синтаксически корректную для языка MiniM. Результат возвращается в значении Target. При успехе кодирования возвращается ненулевое значение. Если результат не может поместиться в размер массива байт в Target, то возвращается значение 0.

Примеры декорирования последовательностей байт, заданных строками C:

```
"abcd"      ->   "abcd"
"ab\"cd"    ->   "ab""cd"
"ab\r\ncd"  ->   "ab"_$C(13,10)_"cd"
```

Демонстрационные примеры по использованию MiniM Server Connect находятся в подкаталогах подкаталога minimsc с именем, соответствующим применяемому языку.

Для установки модуля minimsc.dll на клиентский компьютер вместе с прикладной программой дополнительные модули, не входящие в Windows, не требуются. Для работы minimsc.dll не требует регистрации никаких COM объектов и никаких записей в реестре.

Режим чтения значения выражения

Клиент получает данные функцией *MNMRead*. В качестве вычисляемого значения может использоваться произвольное выражение языка MiniM. При выполнении чтения серверу отправляется команда чтения и ожидается ответ. Полученный ответ возвращается.

Режим получения вывода в текущее устройство

Клиент включает режим передачи выполняя *MNMExecuteOutput*. На сервер отправляется команды выполнить указанную последовательность команд. Если при выполнении команд производится вывод в текущее устройство, то выводимые байты отсылаются клиенту и вызывается установленный обработчик события *Output*.

Вызов обработчика производится по мере поступления данных от сервера и в общем случае принимаемые обработчиком данные могут не совпадать по порционности с выполняемыми командами *write*. В один обработчик может попасть как часть выводимой строки, так и несколько выводимых строк. Клиентская часть относится к потоку данных как к текстовому, в нем не должен встречаться символ $c(0). Отсылка данных серверной частью выполняется без ожидания подтверждения приема данных клиентом.

Пример применения этого режима - получение вывода компиляции или работы иных утилит, выводящих результат работы в текущее устройство.

Режим группового чтения

Режим группового чтения образуется сочетанием на клиенте обработчика события группового чтения, установленного функцией *MNMSetGroupRead* и выполнением на сервере функции

```
d wo^%srv(data)
```

Данные в значении data отсылаются клиентской стороне и на каждый такой вызов на клиенте вызывается обработчик группового чтения. Обработчик вызывается целостно на всю порцию данных *data* и в последовательности байт могут быть произвольные байты. Серверная часть

при вызове отсылки данных не ожидает подтверждения их приема клиентской стороной и продолжает работу далее.

Пример применения этого режима - пакетная отсылка строк данных для заполнения, например, таблиц, возврат клиентской части массива строк при выполнении функции.

Режим обратного вызова

Режим обратного вызова образуется сочетанием на клиенте обработчика обратного вызова установленного функцией *MNMSetCallback* и выполнением на серверной стороне чтения результата функции

```
$$cb^%srv(command)
```

Данные передаваемые в значении *command* согут содержать произвольные байты и передаются обработчику обратного вызова целостно. После вызова обработчика *Callback* серверная сторона ожидает возврата значения от клиента на посланную команду. Если при этом клиентская сторона выполняет запрос иной кроме возврата из обработчика *Callback*, то он выполняется.

Обе стороны, и клиентская и серверная, поддерживают условное соответствие своих уровней стеков обработки и обработчик обратного вызова допускает рекурсию в зависимости от того, что указал выполнять программист. После того, как обработчик *Callback* выполнится и возвратит строку результата, она возвращается из соответствующего ей вызова

```
$$cb^%srv(command)
```

Пример применения этого режима - интерактивный запрос пользователя при выполнении программы или отправка данных клиентской части, при получении которых клиентская часть может вызвать действия на сервере в зависимости от каких-либо условий.

В режимах группового чтения и обратного вызова отправляемые клиентской части данные отправляются целостно, полностью всей указанной последовательностью, и могут включать в себя произвольные байты. Способ кодирования данных в случае отсылки в одной порции нескольких значений программист выбирает самостоятельно, например это может быть формат с разделителями или результат функции *$listbuild()*.

3.9 MiniM Server Connect, ActiveX

Компонент ActiveX библиотека minimscx.dll представляет собой ActiveX интерфейс к библиотеке MiniM Server Connect и преобразует вызовы в соглашениях OLE Automation к соглашениям MiniM Server Connect. Для работы на клиентском компьютере должны быть одновременно размещены в доступном каталоге оба файла, minimsc.dll и minimscx.dll и файл minimscx.dll должен быть зарегистрирован после своей инсталляции

```
regsvr32 minimscx.dll
```

В зависимости от назначения инсталлятора могут быть использованы различные ключи программы regsvr32.

Библиотека MiniMCSX реализует два объекта ActiveX:

```
MiniM.ServerConnect
MiniM.ServerString
```

Тип MiniM.ServerConnect предназначен для соединения с сервером MiniM и выполнения операций взаимодействия, а тип MiniM.ServerString представляет строку символов передаваемую или принимаемую от сервера. Часть функций объекта MiniM.ServerConnect имеет дублирование с суффиксом Str. Если основная функция оперирует данными в виде объекта MiniM.ServerString, то упрощенные функции принимают и передают просто строки. Тип MiniM.ServerString может хранить в себе символы, недопустимые в простой строке, в частности, значения списков в формате функции $lb().

Функции объекта MiniM.ServerConnect

```
CreateConnect( server, port, database)
```

server	Строка, задает имя компьютера где работает MiniM Database Server
port	Число, задает номер порта который обслуживает рутина %srv
database	Строка, задает базу данных, которая должна стать текущей

Функция возвращает 1 при успешном выполнении или генерирует ошибку создания объекта подключения. После создания объекта подключения он остается неподключенным. Для подключения нужно вызвать функцию Open. После создания объекта подключения он должен быть освобожден вызовом функции DestroyConnect.

`Open`

Функция не имеет параметров и возвращает 1 при успешности соединения или 0 при неуспехе. Если перед функцией Open объект соединения не был создан, то генерируется ошибка что объект соединения не существует.

`Close`

Функция не имеет параметров и возвращает 1 при успешном отсоединении от базы или 0 при ошибке отсоединения. Если перед функцией Close объект соединения не был создан, то генерируется ошибка что объект соединения не существует.

`DestroyConnect`

Функция не имеет параметров и возвращаемых значений. Если перед ее вызовом был создан объект соединения то он отсоединяется если был соединен и уничтожается.

`GetLastError`

Функция не имеет параметров и возвращает строку с последней ошибкой произошедшей при работе соединения.

`Read( Expression, Value )`

Функция читает с сервера значение указанного в объекте Expression вычисляемого выражения и возвращает значение в объект Value. Возвращаемое значение 1 при успешном выполнении или 0 при ошибке.

Expression	Объект типа MiniM.ServerString, задает вычисляемое выражение на языке MUMPS
Value	Объект типа MiniM.ServerString, в него возвращается вычисленное значение

Возвращаемое значение может содержать произвольные символы и бинарную последовательность, в частности, значение списков в формате функции $lb().

```
ReadStr( Expression)
```

Функция, парная к функции Read, принимает один строковый аргумент и возвращает строку. Функция вычисляет значение выражения указанного в Expression.

Expression	Строка, задает вычисляемое выражение на языке MUMPS

В возвращаемом значении, в силу того что это строка, не должны встретиться нулевые символы.

```
Write( VarName, VarValue)
```

Функция присваивает переменной, имя которой указано в VarName, значение указанное в VarValue. Функция возвращает значение 1 при успешном выполнении или 0 при ошибке.

VarName	Объект MiniM.ServerString, задает имя переменной на языке MUMPS
VarValue	Объект MiniM.ServerString, задает значение которое следует присвоить

```
WriteStr( VarName, VarValue )
```

Функция является парной к функции Write и принимает параметры в виде простых строк. Возвращает значение 1 при успешном выполнении или 0 при ошибке.

```
Execute( Commands )
```

Функция выполняет указанную в Commands последовательность команд. Параметр Commands должен быть объектом типа MiniM.ServerString. Функция возвращает значение 1 при успешном выполнении или 0 при ошибке.

`ExecuteStr( Commands )`

Функция является парной к функции Execute и принимает параметр в виде простой строки. Функция возвращает значение 1 при успешном выполнении или 0 при ошибке.

`ExecuteOutput( Commands )`

Функция выполняет указанную в Commands последовательность команд. Параметр Commands должен быть объектом типа MiniM.ServerString. Функция возвращает значение 1 при успешном выполнении или 0 при ошибке. При выполнении команд все что процесс будет выводить в текущее устройство ввода-вывода перенаправляется в событие OnTerminalOutput с произвольным разбиением на фрагменты. При выполнении команд ExecuteOutput отличие от Execute в том, что соединение выполняет команды в состоянии перенаправления вывода. При возвращении управления из ExecuteOutput соединение переключается в обычный режим.

`ExecuteOutputStr( Commands )`

Функция, парная к функции ExecuteOutput и принимает параметр в виде просто строки.

`Kill( VarName )`

Функция принимает объект типа MiniM.ServerString и считает что в нем задано имя переменной. Индексы переменной могут содержать произвольные вычисляемые выражения языка MUMPS. Функция удаляет указанную переменную. При успешном выполнении функция возвращает значение 1, при ошибке 0.

`KillStr( VarName )`

Функция, парная к функции Kill и принимает в качестве параметра просто строку с именем переменной. Возвращает значение 1 при успешном выполнении или 0 при ошибке.

`ListLength( List )`

List	Объект MiniM.ServerString, задает значение в формате функции $lb()

Функция возвращает число, количество элементов списка в значении List.

```
ListGet( List, Pos, Elem)
```

Функция возвращает в объект Elem значение элемента списка из объекта List в позиции Pos.

List	Объект MiniM.ServerString, рассматривается как список в формате функции $lb()
Pos	Целое число, задает номер позиции в списке
Elem	Объект MiniM.ServerString, в который необходимо записать значение элемента списка

При успешном выполнении функция возвращает значение 1, иначе значение 0.

```
ListSet( List, Pos, Elem)
```

Функция записывает в объект List рассматривая его как список в позиции Pos значение из объекта Elem.

List	Объект MiniM.ServerString, рассматривается как список в формате функции $lb()
Pos	Целое число, задает номер позиции в списке
Elem	Объект MiniM.ServerString, значение которого нужно записать в список в указанную позицию

При успешном выполнении функция возвращает значение 1, иначе значение 0.

```
Text( Value, Result )
```

Функция производит декорирование строки указанной в Value чтобы результат соответствовал строковому представлению этого значения по правилам языка MUMPS.

Value	Объект MiniM.ServerString, значение которого требуется декорировать
Result	Объект MiniM.ServerString, в который требуется записать результат

При успешном выполнении функция возвращает значение 1, иначе значение 0.

```
TextStr( Value)
```

Функция является парной к функции Text и имеет упрощенный интерфейс. Принимаемый аргумент должен быть строкой, функция возвращает строку.

События объекта MiniM.ServerConnect

```
OnTerminalOutput( Value )
```

Value	Строка, которая была передана в текущее устройство на сервере при выполнении одной из функций ExecuteOutput или ExecuteOutputStr

Событие вызывается по мере прихода клиенту символов записанных в текущее устройство ввода-вывода. Объект соединения не выполняет разбор содержания или объединения данных в цельные строки, символы могут прийти разделенными на произвольное число байт. В строке не может прийти значение нулевого байта, поскольку он является терминатором режима ExecuteOutput. При выполнении иных функций кроме ExecuteOutput или ExecuteOutputStr объект соединения работает в обычном режиме и это событие не вызывается.

```
OnGroupRead( Value )
```

Value	Объект типа MiniM.ServerString, передаваемый с сервера.

Событие OnGroupRead вызывается на клиенте при выполнении на сервере функции

```
d wo^%srv(value)
```

При этом передаваемое значение value собирается полностью и вызывается событие OnGroupRead с передачей этого значения в параметре Value. Событие выполняется асинхронно, сервер не ожидает ответа от клиента и может выполнять массовую потоковую передачу данных без подтверждения от клиентской программы. При срабатывании этого события нельзя обращаться к серверу для выполнения кода на языке MUMPS.

```
OnCallback( Command, Answer )
```

Command	Объект типа MiniM.ServerString, передаваемый с сервера клиенту в качестве команды.
Answer	Объект типа MiniM.ServerString, возвращаемый с клиента серверу в качестве ответа.

Событие OnCallback вызывается на клиенте при вызове на сервере функции

```
$$cb^%srv(command)
```

При этом значение command полностью собирается на клиенте и передается в качестве параметра Command. При срабатывании этого события серверная часть ожидает ответа клиента. При вызове обработчика этого события объект соединение ждет возврата управления и значение записанное в параметр Answer передается серверу. Сервер собирает значение Answer полностью, вне зависимости от работы сети.

При срабатывании события OnCallback можно повторно или рекурсивно обращаться к серверу для выполнения команд или чтения значений выражений. При этом сохраняется соответствие уровней стеков выполнения серверной и клиентской программ.

Объект MiniM.ServerConnect никак не структурирует значения Command и Answer и не налагает на их структуру никаких требований. Значение и характер передаваемых данных полностью определяется программистом.

Функции объекта MiniM.ServerString

```
Value
```

Value является свойством объекта MiniM.ServerString, имеет тип строка. Свойство доступно на чтение и запись.

`Length`

Length является свойством типа число и определяет длину данных в объекте MiniM.ServerString. Свойство доступно на чтение и запись. При присваивании значения меньше 0 значение устанавливается в 0, при присваивании больше допустимого (32767) длина устанавливается в максимально допустимое значение.

`GetAt( Pos)`

Функция возвращает целое число, код символа в позиции Pos, где позиция Pos задается целым числом. Если Pos указывает за пределы данных, то возвращается значение -1. Коды символов возвращаются как числа от 0 до 255.

`SetAt( Pos, Code)`

Функция записывает символ с кодом Code в позиции Pos. Значения Code и Pos задаются целыми числами. Если значение Pos указывает за пределы данных, то никаких изменений не выполняется.

`Add( Val )`

Функция конкатенирует к текущему значению объекта значение объекта Val типа MiniM.ServerString. Значение объекта Val не изменяется. При успехе функция возвращает значение 1, иначе 0.

`AddStr( Val)`

Функция конкатенирует к текущему значению объекта строку Val. Значение строки Val не изменяется. При успехе функция возвращает значение 1, иначе 0.

Примеры использования объекта MiniM.ServerConnect и MiniM.ServerString приведены в подкаталоге VBS и выполнены на языке Visual Basic Script. Для выполнения скриптов примеров нужно выполнить команду

```
cscript.exe //Nologo example1.vbs
```

Все примеры являются консольными.

3.10 Интерфейс экспорта-импорта

Сервер MiniM предоставляет средства для программного экспорта и импорта глобалов, рутин и байткода рутин.

3.10.1 Импорт глобалов

Функция

```
$$import^%GI(fname,fmt,.list,show)
```

Параметры

fname	Имя файла, из которого следует импортировать данные глобалов
fmt	Номер формата, в котором записаны данные
list	Ссылка на локальную переменную, в ключи которой записываются имена импортированных глобалов (необязательный параметр)
show	Выводить (1) или нет (0) отчет об импорте в текущее устройство (необязательный параметр, по умолчанию 0)

Форматы

Параметр fmt может принимать одно из значений:

1	Потоковый формат (Cache или MSM), символы перевода строки зарезервированы форматом.
2	Формат записей с переменной длиной (Cache или MSM), данные и индексы могут содержать любые значения.

Возвращаемое значение

Возвращается количество импортированных глобалов с различными именами. При неуспехе возвращается 0.

Пример

```
s fname="e:\Data.gsa"
```

```
s fmt=2
s show=1
s count=$$import^%GI(fname,fmt,.list,show)
```

3.10.2 Блочный импорт глобалов

Функция

```
$$import^%GBI(filename)
```

Параметры

filename	Имя файла, из которого следует провести блочный импорт.

Возвращаемое значение

При успехе выполнения функция возвращает значение 0. При неудаче функция возвращает внутренний код ошибки, после которого после запятой следует текст описания ошибки. Например:

```
1,Failed to read or write file specified
2,Invalid MiniM block export file
```

При импорте в текущее устройство ввода-вывода ничего не выводится.

3.10.3 Импорт рутин

Функция

```
$$import^%RI(fname,.rlist,compile,show)
```

Параметры

fname	Имя файла, из которого следует импортировать рутины
rlist	Ссылка на локальную переменную, в ключи которой записываются имена импортированных рутин (необязательный параметр)

compile	Индикатор 0 или 1 следует ли компилировать рутины после импорта (необязательный параметр, по умолчанию 1)
show	Индикатор следует ли отображать ход импорта в текущее устройство (необязательный параметр, по умолчанию 0)

Возвращаемое значение

Возвращается количество импортированных рутин.

Пример

```
s fname="e:\matr.rou"
s compile=1
s show=0
s count=$$import^%RI(fname,.rlist,compile,show)
```

3.10.4 Импорт байткода

Функция

```
$$import^%RIMF(fname,.list,show)
```

Параметры

fname	Имя файла, из которого следует импортировать байткод рутин
list	Ссылка на локальную переменную, в ключи которой записываются имена рутин, чей байткод импортирован (необязательный параметр)
show	Индикатор отображать (1) или нет (0) ход импорта в текущее устройство (необязательный параметр, по умолчанию 0)

Возвращаемое значение

Возвращается количество рутин, чей байткод был импортирован.

Пример

```
s fname="e:\query.mmo"
```

```
s show=1
s count=$$import^%RIMF(fname,.list,show)
```

3.10.5 Экспорт глобалов

Функция

```
$$export^%GO(source,fname,descr,fmt,show)
```

Параметры

source	Имя переменной (локальной или глобальной), в ключах которой перечислены имена глобало для экспорта
fname	Имя файла куда следует экспортировать
descr	Комментарий экспорта (записывается в файл)
fmt	Номер формата экспорта
show	Показывать (1) или нет (0) ход экспорта в текущее устройство (необязательный параметр, по умолчанию 0)

Форматы экспорта

1	Формат Cache, потоковый, символы перевода строки используются как служебные символы
2	Формат Cache, записи переменной длины, значения и индексы могут содержать любые символы
3	Формат MSM, потоковый, символы перевода строки используются как служебные символы
4	Формат MSM, записи переменной длины, значения и индексы могут содержать любые символы

Если в качестве комментария указан один символ ^ (циркумфлекс) и формат либо 1 либо 3, то вместо заголовка в файл записывается заголовок автоматического импорта. Впоследствии этот файл может быть импортирован из командной строки операционной системы:

```
minim.exe < globalexport.gsa
```

Возвращаемое значение

Возвращается количество экспортированных глобалов.

Пример

```
s names("^Data")=""
s names("^Global2")=""
s fname="e:\gdata.gsa"
s descr="Daily export"
s fmt=4
s show=0
s count=$$export^%GO($na(names),fname,descr,fmt,show)
```

3.10.6 Блочный экспорт глобалов

Функция

```
$$export^%GBO(name,fname,descr)
```

Параметры

name	Имя локальной или глобальной переменной, в последнем индексе которой перечислены имена глобалов для экспорта.
fname	Имя файла, в который надо экспортировать эти глобалы.
descr	Комментарий, строка записывается в файл, не влияет на значения глобалов.

Возвращаемое значение

При успешном выполнении возвращается значение 0. При неуспехе выполнения экспорта вызвращается ненулевое значение внутреннего номера ошибки и после него после запятой текст с описанием ошибки, например:

```
1,Failed to read or write file specified
2,Invalid MiniM block export file
```

Пример программного экспорта глобалов в блочном формате:

```
s gnames("Data")=""
```

```
s gnames("Ind")=""
s filename="e:\data.g"
s comment="Daily export"
s err=$$export^%GBO($na(gnames),filename,comment)
```

При экспорте в текущее устройство ввода-вывода ничего не выводится.

3.10.7 Экспорт рутин

Функция

```
$$export^%RO(source,fname,descr,show)
```

Параметры

source	Имя переменной (локальной или глобальной), в ключах которой перечислены имена рутин для экспорта.
fname	Имя файла для экспорта.
descr	Комментарий, записывается в файл.
show	Индикатор, выводить (1) или нет (0) ход экспорта в текущее устройство, параметр необязательный, по умолчанию 0.

Если в качестве комментария указан один символ ^ (циркумфлекс), то вместо заголовка в файл записывается заголовок автоматического импорта. Впоследствии этот файл может быть импортирован из командной строки операционной системы:

```
minim.exe < routineexport.rsa
```

Возвращаемое значение

Возвращается количество экспортированных рутин.

Пример

```
s rounames("query")=""
s rounames("order")=""
s show=0
s descr="Daily export"
```

```
s fname="e:\routines.rsa"
s count=$$export^%RO($na(rounames),fname,descr,show)
```

3.10.8 Экспорт байткода

Функция

```
$$export^%ROMF(fname,.rounames,descr,show)
```

Параметры

fname	Имя файла для экспорта
rounames	Ссылка на локальную переменную, в ключах которой перечислены имена рутин, байткод который следует экспортировать
descr	Комментарий, записывается в файл
show	Индикатор, показывать (1) или нет (0) ход экспорта в текущее устройство, параметр необязательный, по умолчанию 0

Возвращаемое значение

Возвращается количество рутин, чей байткод был экспортирован.

Пример

```
s rounames("query")=""
s rounames("order")=""
s descr="Daily export"
s show=0
s fname="e:\routines.mmo"
s count=$$export^%ROMF(fname,.rounames,descr,show)
```

3.11 Интерфейс изменения рутин

MiniM Database Server предоставляет программный интерфейс изменения рутин и получения служебной информации о рутинах. Подпрограммы находятся в служебной рутине ^%R. Подпрограммы используют имена рутин для которых могут быть указаны типы в виде расширения. Если расширение не указано, то подпрограммы используют рутины типа

INT. Расширение нечувствительно к регистру. При неуспехе выполнения подпрограммы возвращают значение 0.

Примеры приводятся на рутине first.inc следующего содержания:

```
 ; header line
 w "Hello from include",!
 w 1234,!
```

Загрузка строк рутины из базы в указанную переменную

Прототип:

```
s err=$$LOAD^%R(rouname,$na(text))
```

Подпрограмма записывает строки рутины в подиндексы указанной в *text* глобальной или локальной переменной. Подпрограмма не записывает служебную информацию. Перед записью переменная *text* очищается.

Пример:

```
USER>s err=$$LOAD^%R("first.inc",$na(rou("rou")))
USER>w
err=1
rou("rou",1)=" ; header line"
rou("rou",2)=" w "Hello from include",!"
rou("rou",3)=" w 1234,!"
```

Запись строк рутины из указанной переменной

Прототип:

```
s err=$$SAVE^%R(rouname,$na(text))
```

Подпрограмма создает или перезаписывает рутину строками из подъиндексов указанной в *text* переменной. Подпрограмма использует строки в порядке их следования в переменной *text*, нумерация строк числами необязательна.

Пример:

```
USER>s rou("a")=" ; comment"
USER>s rou("b")=" w 1234,!"
USER>s rou("c")=" w 7896,!"
USER>s err=$$SAVE^%R("first.inc",$na(rou))
USER>w
err=1
rou("a")=" ; comment"
rou("b")=" w 1234,!"
rou("c")=" w 7896,!"
```

После выполнения этого кода в базе данных создается рутина first.inc следующего содержания:

```
 ; comment
 w 1234,!
 w 7896,!
```

Замена строки в рутине

Прототип:

```
s err=$$SETLINE^%R(rouname,lnum,line)
```

Подпрограмма перезаписывает в рутине *rouname* строку с номером *lnum* на значение из переменной *line*. Нумерация строк для счетчика *lnum* производится от единицы.

Пример:

```
USER>s err=$$SETLINE^%R("first.inc",1," ; changed comment")
```

После выполнения этого кода рутина first.inc получает следующие строки:

```
 ; changed comment
 w 1234,!
 w 7896,!
```

Получение числа строк в рутине

Прототип:

```
s err=$$GETCOUNT^%R(rouname)
```

Подпрограмма возвращает число строк в указанной рутине.

Получение строки рутины

Прототип:

```
s err=$$GETLINE^%R(rouname,lnum,.line)
```

Подпрограмма возвращает в переменную *line* значение строки рутины *rouname* с номером *lnum*. Нумерация строк для счетчика *lnum* производится от единицы.

Пример:

```
USER>s err=$$GETLINE^%R("first.inc",2,.line)
USER>w
err=1
line=" w 1234,!"
```

Вставка строки

Прототип:

```
s err=$$INSLINE^%R(rouname,lnum,line)
```

Подпрограмма вставляет строку *line* в рутину *rouname* начиная со строки *lnum*. Имеющиеся в рутине строки начиная со строки *lnum* сдвигаются далее. Нумерация строк для счетчика *lnum* производится от единицы.

Пример:

```
USER>s err=$$INSLINE^%R("first.inc",2," #; second comment")
```

После выполнения этого кода рутина first.inc получает значение:

```
 ; changed comment
 #; second comment
 w 1234,!
 w 7896,!
```

Удаление строки

Прототип:

```
s err=$$DELLINE^%R(rouname,lnum)
```

Подпрограмма удаляет в указанной рутине *rouname* строку с номером *lnum*. Строки старше чем *lnum* сдвигаются в младшие номера. Нумерация строк для счетчика *lnum* производится от единицы.

Пример:

```
USER>s err=$$DELLINE^%R("first.inc",3)
```

После выполнения этого кода рутина first.inc получает значение:

```
 ; changed comment
 #; second comment
 w 7896,!
```

Получение времени модификации рутины

Прототип:

```
s hor=$$DATE^%R(rouname)
```

Подпрограмма возвращает дату и время модификации рутины *rouname* в формате системной переменной *$horolog*. В случае если указанная рутина не существует возвращается пустая строка.

Пример:

```
USER>w $$DATE^%R("first.inc")
62136,59140
```

Получение времени модификации байткода рутины

Прототип:

```
s hor=$$BCDATE^%R(rouname)
```

Подпрограмма возвращает дату и время модификации байткода рутины *rouname* в формате системной переменной *$horolog*. В случае если байткод для указанной рутины не существует возвращается пустая строка.

Пример:

```
USER>w $$BCDATE^%R("first.inc")
USER>w $$BCDATE^%R("%MPP")
62131,58917
```

Изменение текста рутин в MiniM Database Server не сказывается на выполнении компилированного байткода, поскольку подпрограммы рутины ^%R его не изменяют. Для того чтобы скомпилировать рутину предоставляется подпрограмма

```
d compile^%RCOMPIL(rouname)
```

Подпрограмма не компилирует рутину если указан тип INC. Если указан тип MAC, то предварительно вызывается препроцессинг рутины и получение INT рутины. Далее рутина компилируется в исполняемый байткод. При работе подпрограмма выводит сообщения об обнаруженных ошибках в текущее устройство.

Глава 4

CHUI утилиты

CHUI - аббревиатура от CHaracter User Interface. Это программы, вызываемые в интерактивной алфавитно-цифровой среде, например консоль или телнет. На экран выводится текст руководства к действию и предлагается ввести с клавиатуры данные.

CHUI утилиты могут быть использованы как при локальном доступе с консоли, так и с произвольного компьютера, подключенного к сети, через телнет-клиент. Утилиты CHUI традиционно выполняются в спартанском стиле "введите число" для того, чтобы они могли быть выполнены оператором или администратором в произвольной критической ситуации, используя произвольный телнет-клиент и минимизируют использование специфики отображающего устройства.

Утилиты есть интерфейс к исполняемой на сервере программе и следует понимать, что если запрашивается имя файла или каталога, то эти файлы и каталоги должны располагаться в видимости процессов сервера и быть доступны в рамках прав аккаунта выполняющегося процесса.

4.1 %BACKUP

Утилита ^%BACKUP предназначена для выполнения сохранения одной или нескольких или всех баз данных сервера в файл бекапа.

Использование

```
do ^%BACKUP
```

Утилита предлагает выбор типа сохранения - полный или дифференциальный бекап. Утилита выполняет горячий бекап и может быть использована при активных процессах, работающих в транзакции.

```
MiniM Backup utility.
Select backup type:
 1) Full backup
 2) Differential backup
Select option:
```

Ожидает ввода числа - 1 или 2, при выборе неподходящей альтернативы (например, просто Enter) прекращает работу.

Далее предлагается выбор баз для бекапа

```
Select databases to backup.
 1) All databases
 2) Selected databases
Select option:
```

При выборе альтернативы 1 (All databases) в бекап попадут все базы, которые сконфигурированы, и у которых отсутствует индикатор автосоздания на старте (Autocreate = 0).

При выборе альтернативы 2 (Selected databases) утилита предлагает список имеющихся баз данных и предлагает выбрать номера баз для бекапа.

```
List of available databases:
 1) USER   2) %SYS   3) TEMP
Select db number to add to backup:
```

После выбора номера базы эта база помещается в список выборанных и предлагается выбор других баз данных.

```
List of available databases:
 1) USER   2) %SYS   3) TEMP
Select db number to add to backup: 1

List of selected databases:
USER
List of available databases:
 2) %SYS   3) TEMP
Select db number to add to backup:
```

При выборе несуществующей альтернативы (например, просто Enter) утилита заканчивает выбор баз. Если не было выбрано ни одной, то утилита прекращает работу, если были выбраны по крайней мере одна база, то далее предлагается выбор имени файла для бекапа.

```
List of selected databases:
USER
List of available databases:
 2) %SYS   3) TEMP
Select db number to add to backup:

Enter file name to backup to:
```

При вводе пустого имени утилита считает что был отказ от сохранения и прекращает работу. Имя файла должно быть доступно процессу на сервере. Утилита не налагает ограничений на имя и расширение файла, и могут быть указаны промежуточные каталоги, в том числе несуществующие еще. Если их нет то утилита бекапа их сделает.

Далее предлагается опция операции с журналом:

```
Journal truncate option:
 1) Truncate journal
 2) Keep journal as is
Select option:
```

При выборе альтернативы 1 (Truncate journal) из журнала будут удалены все записи ранее самой ранней необходимой для отката текущих выполняемых транзакций. При выборе альтернативы 2 (Keep journal as is) никаких операций с журналом не выполняется, он оставляется как есть.

Далее запрашивается опция отчета о выполняемых действиях.

```
Report option:
 1) Make report
 2) No report
Select option:
```

При выборе опции 1 (Make report) утилита будет выводить перечень выполняемых ей действий.

Если была выбрана опция (Make report), далее запрашивается имя файла для вывода отчета.

```
Report option:
 1) Make report
 2) No report
Select option: 1
Enter file name to print backup report to:
```

Если введена пустая строка то отчет выводится но не в файл, а на экран. Если указан файл, то отчет о выполнении бекапа выводится в него.

Содержание отчета от выполняемых действиях зависит от выбранных опций и длительность каждой фазы бекапа зависит от объема выполняемой работы. Например, при дифференциальном бекапе одной базы USER отчет может выглядеть так:

```
Start backup modified databases.
Start backup modified database USER
Start finish backup phase.
Finish backup USER
Backup done successfully.
```

4.2 %DBCLEAN

Утилита ^%DBCLEAN предназначена для физического удаления из блоков базы данных неиспользуемых данных и очистки занимаемого ими места.

Использование

```
do ^%DBCLEAN
```

Утилита предлагает список имеющихся для проверки баз данных и запрашивает номер базы в этом списке для очистки.

```
MiniM database cleanup utility
Utility cleans unused database space.
All used data will not be touch.

Available database list:
 1) USER 2) %SYS 3) TEMP
Select database number to clean:
```

При выборе отсутствующего варианта (или просто при нажатии Escape или Enter) утилита прекращает работу. Иначе производит перебор всех имеющихся в базе данных блоков и выполняет очистку.

Утилита выполняет чтение всех блоков, но модифицирует только некоторые типы, для которых можно определить неиспользуемое пространство.

Время работы утилиты определяется количеством блоков базы данных. При работе утилита вычитывает их все в кеш глобалов, это может приводить к обесцениванию кеша для других процессов и кратковременному замедлению работы некоторых процессов. Схема кеширования LRU, применяемая в кеше глобалов, снижает эффект вымывания кеша однократным чтением, но для некоторых процессов может потребоваться повторное чтение вымытого из кеша блока.

Примерный вид работы утилиты:

```
TEMP>d ^%DBCLEAN

MiniM database cleanup utility
Utility cleans unused database space.
All used data will not be touch.

Available database list:
 1) USER 2) %SYS 3) TEMP
Select database number to clean: 3
Cleanup database TEMP
Database TEMP contains 128 blocks.
Cleanup utility ends.
```

Утилита может быть запущена из любой базы данных и для любой базы данных, для которой разрешена запись.

4.3 %DBCRC

Утилита ^%DBCRC предназначена для проверки всех блоков базы данных на соответствие контрольной сумме.

Использование

```
do ^%DBCRC
```

Утилита предлагает список имеющихся для проверки баз данных и запрашивает номер базы в этом списке для проверки.

```
MiniM database CRC check utility

Available database list:
 1) USER 2) %SYS 3) TEMP
Select database number to check:
```

При выборе отсутствующего варианта (или просто при нажатии Escape или Enter) утилита прекращает работу. Иначе производит проверку всех блоков принадлежащих базе данных на соответствие их контрольных сумм.

Контрольная сумма присутствует в заголовке каждого из блоков и формируется при записи. Проверка на соответствие контрольных сумм позволяет определить наличие проблем с хранением файлов базы данных в файловой системе, присутствуют ли в файловой системе искажения информации.

Время работы утилиты определяется количеством блоков базы данных. При работе утилита вычитывает их все в кеш глобалов, это может приводить к обесцениванию кеша для других процессов и кратковременному замедлению работы некоторых процессов. Схема кеширования LRU, применяемая в кеше глобалов, снижает эффект вымывания кеша однократным чтением, но для некоторых процессов может потребоваться повторное чтение вымытого из кеша блока.

При окончании работы утилита указывает, обнаружена ли проблема или нет. Проблема указывается для всей базы данных и не детализируется, в каком именно файле. Для исправления проблемы нужно проанализировать, почему файловая система искажает файл, исправить проблему, и восстановить базу данных из бекапа.

4.4 %DBSIZE

Утилита ^%DBSIZE показывает текущий размер баз данных в мегабайтах и пределы роста баз данных, а также расширяет текущий объем баз данных.

Использование

```
do ^%DBSIZE
```

Утилита предлагает на выбор две опции - показать текущие размеры баз данных, которые они занимают на дисках и добавление пространства к базе данных.

```
USER>d ^%DBSIZE
MiniM Show database size utility

Select utility option:
 1) Show current database size and limit
 2) Extend current database size
Select option:
```

При выборе опции 1 утилита выводит список текущих используемых инсталляцией баз данных, занимаемый ими размер и предел роста. Значения указываются в мегабайтах.

Перед выводом списка утилита запрашивает необязательное имя файла для вывода отчета. Если имя файла указано, то отчет будет выводиться в него.

По окончании вывода размеров утилита переходит к началу работы, запрашивает действие, которое нужно выполнить. Примерный экран работы утилиты:

```
MiniM Show database size utility

Select utility option:
 1) Show current database size and limit
 2) Extend current database size
Select option: 1

Enter file name to print to (optional):
Database                        Size (MB)  Limit (MB)
--------                       ----------  ----------
USER                                   11   unlimited
%SYS                                    1        1024
TEMP                                    6        1024

Select utility option:
 1) Show current database size and limit
 2) Extend current database size
Select option:
```

При выборе опции 2 утилита выводит список имеющихся баз данных и запрашивает номер базы в этом списке, для которой требуется расширение занимаемого пространства.

После этого утилита запрашивает количество мегабайт, которое требуется добавить к занимаемому пространству. После чего производится расширение базы данных вне зависимости от установленных для этой базы данных настроек по авторасширению. Примерный экран работы утилиты:

```
Select utility option:
 1) Show current database size and limit
 2) Extend current database size
Select option: 2

Available databases:
1) USER 2) %SYS 3) TEMP

Select database number to extend: 3

Enter megabytes count to add to database TEMP : 2
Database extension done.
```

Утилита ^%DBSIZE предназначена для контроля расходуемого базами данных дискового пространства и для ручного добавления администратором пространства для баз данных, для которых отключено авторасширение по каким-либо причинам. Добавление пространства к базам данных выполняется как для баз с авторасширением, так и без. Расширение баз этой утилитой не выполняется свыше установленного в конфигурации предела роста баз. При запросе добавления пространства свыше предела роста расширение выполняется только до предела роста баз.

4.5 %GBI

Утилита ^%GBI выполняет импорт данных из файла блочного экспорта глобалов и показывает служебную информацию из таких файлов.

Использование

```
do ^%GBI
```

Утилита предлагает ввести имя файла и запрашивает номер действия:

```
USER>d ^%GBI
MiniM block global import
Enter file name to import globals from: e:\blocks.g
Select operation:
 1. Show export comment
 2. Show global's names
 3. Import globals
Option:
```

При выборе опции 1 утилита выводит комментарий, котоырй был введен при экспорте глобалов в этот файл.

При выборе опции 2 утилита выводит перечень имен глобалов, которые были экспортированы в этот файл.

При выборе опции 3 утилита производит импорт глобалов из этого файла.

Для импорта могут быть использованы только файлы, созданные блочным экспортом MiniM, и никакой другой реализацией MUMPS.

При выполнении импорта импортируются не блоки, а логические данные, которые они содержат. Импорт выполняется в текущую базу данных и в текущем окружении, с текущими настройками журналирования и изменением индикаторов бекапа. Поэтому блочный импорт может быть произведен в базу данных, которая была изменена после блочного экспорта.

Импорт выполняется с той же моделью поведения, что и команда merge - при совпадении импортируемого имени и имени в базе данных значение перезаписывается, если такое имя отсутствует, то оно создается, если в базе есть ключи отсутствующие в файле экспорта, то они не затрагиваются.

Блочный импорт может выполняться заметно медленнее блочного экспорта, поскольку требуется время на преобразование ключей и данных, записанных в файле экспорта, для выполнения логических модификаций в базе данных по каждому экспортированному ключу и для выполнения журнальных операций. Соотношение скоростей блочного экспорта и импорта также зависит от текущего состояния кеширования глобалов.

Блочный экспорт выполняется не цельно, а в режиме команды merge - при его выполнении изменение глобала иными процессами не блокируется.

При импорте вставка ключей использует текущее определение таблицы символов для индексного сравнения.

4.6 %GBO

Утилита ^%GBO выполняет экспорт выбранных глобалов в файл на сервере в блочном формате.

Использование

```
do ^%GBO
```

Утилита предлагает ввести имя файла, в который следует экспортировать глобалы. В случае если такой файл существует, то он будет перезаписан. После чего запрашивает комментарий экспорта. В комментарии можно отметить информацию описывающую этот экспорт. Ввод комментария необязателен.

```
USER>d ^%GBO
MiniM block global export
Enter file name to export globals: e:\datablk.g
Enter description:
```

После этого утилита запрашивает ввод масок имен глобалов для экспорта. Для окончания ввода имен нужно ввести пустую строку (нажать Enter).

```
Global name mask: GloData
Global name mask:
```

После ввода имен глобалов утилита выполняет блочный экспорт данных глобала.

Блочный экспорт отличается от других форматов тем, что экспортируются значимые части блоков, содержащие ключи и данные необходимые для последующего восстановления этих глобалов. Используется внутренний формат, аналогичный формату блоков баз данных и он не документируется. Внутренние функции блочного экспорта и импорта группы $v("db") также не документируются и могут быть изменены в последующих версиях. Экспорт выполняется из текущей области.

Блочный экспорт выполняется цельно, глобал экспортируется в том виде в каком он был на момент начала экспорта. В течении блочного экспорта модификация глобала невозможна. Другие процессы при попытке изменения глобала ставятся в ожидание окончания экспорта. Цельный экспорт выполняется только для одного глобала. Если на экспорт назначены два или более глобалов, то последующий глобал экпортируется в том виде, в каком он был на момент окончания экспорта предыдущего глобала.

Блочный формат экспорта глобалов предназначен для импорта только сервером MiniM, и никакими другими реализациями MUMPS. Блочные форматы иных реализаций также не принимаются для импорта в MiniM. Поэтому блочный формат можно рассматривать как средство переноса данных только между серверами MiniM.

Блочный экспорт выполняется существенно быстрее чем экспорт данных в другие форматы, поскольку практически не требует прохода по глобалам как по логическим структурам и записывает фактически содержательные части блоков в том виде в котором они находятся в базе данных.

4.7 %GDIR

Утилита ^%GDIR выводит список имен глобалов текущей базы данных, имеющих данные.

Использование

```
do ^%GDIR
```

Утилита предлагает ввести маску имен глобалов. В маске допустимо использовать символ ? как заменитель одного символа и символ * как заменитель произвольной последовательности.

Утилита запрашивает последовательно несколько масок. Если введена пустая строка (нажата клавиша Enter или Escape) то утилита считает это окончанием ввода масок имен.

Далее утилита запрашивает необязательное имя файла, в который следует вывести список имен глобалов. Если введена пустая строка (нажата клавиша Enter или Escape) то утилита выводит список имен на экран, иначе в указанный файл.

Пример:

```
Global name mask: *
Global name mask:
Enter file name to print to (optional):
List of selected globals's in database USER

^A
^D
^ROUTINE
^UTILITY
^a
^rOBJ

USER>d ^%GDIR
MiniM Globals listing utility

Global name mask: *O*
Global name mask:
Enter file name to print to (optional):
List of selected globals's in database USER

^ROUTINE
^rOBJ
```

4.8 %GI

Утилита ^%GI выполняет импорт в текущую базу данных глобалов из файла экспорта глобалов.

Использование

```
do ^%GI
```

Утилита предлагает ввести имя файла из которого нужно провести импорт данных. Если вводить пустую строку (нажата клавиша Enter или Escape), то утилита считает это отказом от импорта и заканчивает работу.

После этого утилита предлагает ввести код формата, в котором был произведен экспорт.

```
Select file format:
  1. Stream format (Cache or MSM or ANSI)
  2. Variable-length format (Cache or MSM)
Format option:
```

При выборе кода формата утилита использует его для распаковки данных в указанном файле. Если указан код, отсутствующий в списке альтернатив, то утилита считает это отказом от импорта и завершает работу.

Утилита использует уже применяемые в эксплуатации форматы файлов, определенные и поддерживаемые стандартом ANSI и компаниями производителями InterSystems и Micronetics.

Форматы потокового типа (Stream format) могут быть использованы, если экспортируются данные глобалов которые не содержат в значениях и в индексах непечатных символов. Если или в индексах или в значениях глобалов используются все возможные байты, то нужно применять форматы переменной длины (Variable-length format). В первом случае индикатором конца порции данных является символ перевода строки, во втором случае индикатор длины содержится в самом файле.

При импорте данных из файла утилита не удаляет данные в тех же глобалах, которые присутствуют в файле, и производит перезапись поверх существующих данных.

Время работы импорта глобалов определяется объемом файла импорта.

4.9 %GL

Утилита ^%GL выполняет импорт в текущую базу данных глобалов из файла экспорта глобалов.

Использование

```
do ^%GL
```

Утилита ^%GL является переходником на утилиту ^%GI для совместимости по именам рутин с другими M системами.

4.10 %GO

Утилита ^%GO выполняет экспорт глобалов в файл.

Использование

```
do ^%GO
```

Утилита предлагает ввести имя файла в который нужно произвести экспорт. При вводе пустой строки (или нажатие Escape или Enter) утилита считает что был отказ от экспорта и прекращает работу.

Далее утилита запрашивает формат экспорта.

```
Select one of the following formats:
 1. Cache stream
 2. Cache variable length
 3. MSM stream
 4. MSM variable length
 Q. Quit export
Choose format:
```

Утилита использует для экспорта уже используемые в эксплуатации форматы данных, определенные производителями других M систем - InterSystems и Micronetics. Потоковые форматы (Cache stream и MSM stream) могут быть использованы только для глобалов, которые не содержат в значениях и в индексах непечатных символов. Если глобалы содержат такие символы, то следует использовать форматы переменной длины (Cache variable length или MSM variable length). В потоковых форматах признаком конца порции данных является символ перевода строки, в форматах переменной длины индикатор длины содержится в файле.

Далее утилита запрашивает краткое описание для этого файла или индикатор использования заголовка автозагрузки. Впоследствии по краткому описанию администратор может установить что содержится в этом файле. Если выбран вариант автозагрузки, то вместо заголовка будет сгенерирован код автозагрузки и впоследствии файл можно будет импортировать автоматически, вызвав

```
minim.exe < filename.gsa
```

Кроме того, при использовании заголовка автозагрузки формат сохраняется и можно импортировать файл с данными обычными средствами, как и без автозагрузки.

При использовании заголовка автозагрузки формируется две первые строки исполняемого кода на M, поэтому использовать автозагрузку в сочетании с форматами переменной длины (Cache variable length или MSM variable length) бесполезно.

После этого утилита запрашивает имена глобалов для экспорта

```
Global name mask: A
Global name mask:
```

При вводе имени глобала символ циркумфлекса "^" необязателен.

При вводе пустой строки утилита считает что выбор имен глобалов закончен и производит экспорт если в базе данных есть глобалы подходящие под выбранные имена.

В качестве имен глобалов может быть использована маска имен, где символ ? заменяет один символ, а символ * заменяет произвольную последовательность символов.

Время работы утилиты экспорта зависит от объема данных, попадающих в экспорт.

4.11 %GS

Утилита ^%GS выполняет экспорт глобалов в файл.

Использование

```
do ^%GS
```

Утилита ^%GS является переходником на утилиту ^%GO для совместимости по именам рутин с другими M системами.

4.12 %JOBTAB

Утилита ^%JOBTAB показывает текущий перечень процессов на сервере и часть их характеристик, позволяет удалить процесс.

Использование

```
do ^%JOBTAB
```

Утилита показывает таблицу процессов с колонками в которых указаны некоторые характеристики процессов - 1) номер процесса, 2) его текущая база данных, 3) текущая исполняемая рутина, 4) устройство ввода-вывода по умолчанию и 5) последняя глобальная ссылка.

Примерный вид экрана утилиты:

```
                Job table at 02.06.2009 13:03:39

Entry Job   Database  Routine    Principal    Last global referenc
   1) 812   %SYS      %srv       |TCP|        ^ROUTINE("%JOBTAB",0
   2) 908   USER      %srv       |NULL|       ^%SRV("stop")
   3) 940   USER      %JOBTAB    |CON|        ^UTILITY(940,2)
Page #1 of 1
[R]eload | [Q]uit | kill [J]ob | kill [E]ntry | Up/Down
Command:
```

Утилита показывает таблицу процессов с перелистыванием по страницам, если количество текущих процессов не дает показать их на одном экране и выводит номер текущей страницы списка и общее число страниц.

Утилита ожидает ввода символа для команды в верхнем или нижнем регистре.

R	Перечитать список и показать его с первой страницы
Q	Выход из утилиты
J	Удалить процесс указав номер процесса, например 940 на приведенном экране
E	Удалить процесс, указав номер строки процесса, например 3 на приведенном экране

При нажатии стрелок вверх и вниз утилита переходит к предыдущей и последующей странице списка соответственно.

Список процессов формируется не мгновенным снимком, а последовательным опросом процессов, поэтому характеристики процессов попадают в список с небольшой разницей во времени. Полное время формирования списка процессов пропорционально количеству процессов на

сервере.

4.13 %JOURNAL

Утилита ^%JOURNAL показывает текущее состояние журнала и управляет переключением и усечением журнала.

Использование

```
do ^%JOURNAL
```

Утилита при запуске выводит текущее состояние журнала - каталог журнала и текущее последнее имя файла журнала.

После этого утилита предлагает выбор операции - переключить текущий файл журнала на новый или усечь журнал.

Примерный вид экрана:

```
Minim Journal state utility

Journal directory:
e:\workfiles\minim\journal\
Current journal file:
e:\workfiles\minim\journal\2009-06-02.002

Select journal operation:
 1) Switch current journal file
 2) Truncate journal
Select option:
```

При выборе альтернативы 1 утилита переключает сервер на новый журнальный файл, увязывая его в общею цепочку, при выборе альтернативы 2 утилита проводит усечение журнала. В каталоге журнала оставляются только файлы, необходимые для отката текущих активных транзакций. Остальные записи в файлах журнала утрачиваются, при необходимости удаляются ненужные более файлы.

При выборе иных альтернатив утилита считает отказом от продолжения работы и прекращает работу.

4.14 %LOCKTAB

Утилита ^%LOCKTAB показывает список имеющихся на сервере блокировок, выполненных командами lock в виде таблицы и выполняет удаление указанной блокировки.

Использование

```
do ^%LOCKTAB
```

Утилита при запуске получает список блокировок и выводит их в виде таблицы постранично. По каждой блокировке выводится информация о процессе, который взял блокировку, о счетчике блокирований, и имя блокировки. Имена блокировок совпадают с именами локальных или глобальных переменных. Примерный вид экрана такой:

```
           Lock table at 02.06.2009 18:51:52
           8388608 bytes total, 160 bytes used

Entry Owner   Count  Item
   1) 940         1  a
   2) 940         1  ^|"USER"|b
Page #1 of 1
[R]eload | [Q]uit | [D]elete one | delete [J]ob's  >>
         >> | delete [A]ll | Up/Down
Command:
```

Утилита выводит текущий расход памяти блокировок, номер страницы таблицы и общее число страниц. После чего ожидает ввода команды, символ в верхнем или нижнем регистре. Команды утилиты:

R	Перечитать таблицу снова и вывести с первой страницы
Q	Окончить работу
D	Удалить одну блокировку, указав номер блокировки в списке
J	Удалить все блокировки процесса, указав номер процесса
A	Удалить все блокировки на сервере

При нажатии стрелок вверх и вниз утилита переходит к предыдущей и последующей странице списка соответственно.

Утилита получает список блокировок не единым снимком, а последовательно. Поэтому в списке при активной работе процессов могут оказаться логически взаимоисключающие данные.

Утилита может удалить блокировку установленную любым процессом, а не только процессом который вызвал утилиту.

4.15 %PERFMON

Утилита ^%PERFMON показывает текущие значения статистик сервера.

Использование

```
do ^%PERFMON
```

При запуске утилита выводит экран мониторинга текущих статистик сервера. Экран обновляется 1 раз в секунду. В первой колонке выводится название счетчика, во второй его текущее значение и в третьей среднее значение за последние 12 секунд.

Утилита ждет произвольного ввода. Если его не происходит, то 1 раз в секунду экран продолжает обновляться. Если нажать любую клавишу, то утилита прекращает работу.

Пример экрана работы утилиты

```
         MiniM Performance Monitor
         -------------------------
Performance Counter                Current       Middle
-------------------------------------------------------
Blocks read/sec                          0            0
Blocks write/sec                         0            0
Blocks allocated/sec                     0            0
Blocks freed/sec                         0            0
Datafile expand MB/sec                   0            0
Globals read/sec                         1            1
Globals write/sec                        0            0
Globals kill/sec                         0            0
Locals read/sec                       1206         1209
```

```
Locals write/sec                    659          662
Locals kill/sec                       0            0
Block split/sec                       0            0
Journal write Bytes/sec               0            0
Routine read/sec                      0            0

Press any key to exit
```

4.16 %RCHANGE

Утилита ^%RCHANGE производит поиск и замену текста в выбранных рутинах и если указано то перекомпиляцию их.

Использование

```
do ^%RCHANGE
```

Утилита запрашивает перечень пар "что заменить" и "на что заменить" с опцией поиска чувствительно или нечувствительно к регистру. Примерный вид экрана:

```
MiniM Replace text in routines utility
  1. Search for: abc
     Replace with: def
     Case sensitive search? [Y/N]: n
  2. Search for:
```

Утилите можно указать несколько пар поиска и замены текста. При вводе пустой строки поиска утилита прекращает запрос пар поиска и замены и запрашивает перечень масок имен рутин.

```
Routine name mask: *a*
Routine name mask:
```

При вводе пустой строки (нажатие Escape или Enter) утилита заканчивает ввод масок имен рутин. В маске можно использовать символы подстановки - ? для одного любого символа и * для произвольной последовательности символов. Утилите можно указать несколько масок имен рутин.

Далее утилита запрашивает следует ли перекомпилировать те рутины которые будут изменены в результате замены текста.

```
Recompile changed routines? [Y/N]:
```

Далее утилита запрашивает имя файла для вывода отчета о выполняемой работе.

```
Enter file name to print to (optional):
```

Если имя файла не указывать, то отчет выводится на экран, иначе в указанный файл.

При работе утилита выводит имена рутин, которые попали под выбранные маски, строки после замены и результат компиляции, если было указано перекомпилировать рутины. Если в рутине не было сделано замены текста, то рутина не перекомпилируется.

Если имена рутин указаны без расширения, то используются только стандартные рутины (INT), иначе используются рутины с указанным расширением. В расширении не различаются маски, и могут быть использованы расширения INT, MAC, INC. Пример:

```
Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: MP*.MAC
Routine name mask: f*.INC
Routine name mask: #L
List of selected routine names:
MPPSRC.MAC
first.INC
--------
```

4.17 %RCOMPIL

Утилита ^%RCOMPIL перекомпилирует указанные рутины.

Использование

```
do ^%RCOMPIL
```

Утилита запрашивает перечень масок имен рутин, которые следует перекомпилировать:

```
MiniM Compile routines utility

Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: ?a*
Routine name mask:
```

Утилите можно указать одну или более масок имен. В маске имени рутины можно использовать символ подстановки - ? для одного любого символа и * для произвольной последовательности.

Далее утилита запрашивает имя файла (необязательного) в который следует выводить отчет о перекомпиляции. Если указана пустая строка (нажата клавиша Escape или Enter) то отчет о работе выводится на экран.

При перекомпиляции выводятся имя текущей базы данных, имена рутин и результат их компиляции с диагностикой синтаксических ошибок. Пример экрана работы утилиты:

```
Compile selected routines in USER

Compile label
Compile matrix
```

После компиляции рутин утилита заканчивает работу.

Если имена рутин указаны без расширения, то используются только стандартные рутины (INT), иначе используются рутины с указанным расширением. В расширении не различаются маски, и могут быть использованы расширения INT, MAC, INC. Пример:

```
Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: MP*.MAC
Routine name mask: f*.INC
Routine name mask: #L
List of selected routine names:
MPPSRC.MAC
first.INC
--------
```

Рутины типа INC не компилируются, рутины типа MAC предварительно транслируются макро препроцессором.

4.18 %RCOPY

Утилита ^%RCOPY копирует выбранные рутины в другую базу данных.

Использование

```
do ^%RCOPY
```

Утилита запрашивает перечень масок имен рутин, которые следует перенести в другую базу данных и при необходимости перекомпилировать:

```
USER>d ^%RCOPY
MiniM Routine Copy Utility

Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: CLI*
Routine name mask:
```

Утилите можно указать одну или более масок имен. В маске имени рутины можно использовать символ подстановки - ? для одного любого символа и * для произвольной последовательности.

Далее утилита запрашивает надо ли проводить перекомпиляцию перенесенных рутин.

```
Recompile routines after copy? [Y/N]: y
```

Далее утилита выводит список имеющихся баз данных и текущую базу данных и запрашивает имя базы данных в которую надо перенести рутины.

```
List of available databases:
   %SYS
   TEMP
   USER
   APPDB
Current database: USER
Enter target database name: appdb
```

Имя базы данных может вводиться без учета регистра. В случае если вместо имени базы данных введена пустая строка, утилита прекращает работу. Если введено имя текущей базы данных, то утилита повторно спрашивает имя базы для переноса.

После переноса всех выбранных рутин утилита при необходимости выполняет их перекомпиляцию.

Если имена рутин указаны без расширения, то используются только стандартные рутины (INT), иначе используются рутины с указанным расширением. В расширении не различаются маски, и могут быть использованы расширения INT, MAC, INC. Пример:

```
Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: MP*.MAC
Routine name mask: f*.INC
Routine name mask: #L
List of selected routine names:
MPPSRC.MAC
first.INC
--------
```

Рутины типа INC не компилируются, рутины типа MAC предварительно транслируются макро препроцессором.

4.19 %RDELETE

Утилита ^%RDELETE удаляет рутины и, опционально, их байткод.

Использование

```
do ^%RDELETE
```

Утилита запрашивает набор масок имен рутин. В маске имени можно использовать символы подстановки - ? для произвольного символа и * для произвольной последовательности. Примерный экран:

```
MiniM Delete routines utility

Routine name mask: q*
Routine name mask:
```

При вводе пустой маски имени утилита запрашивает нужно ли удалять также байткод для рутин с этими именами.

Далее утилита запрашивает необязательное имя файла для вывода отчета об удалении рутин. Если указана пустая строка, то отчет выводится на экран. Утилита выводит имя текущей базы данных и перечень удаленных рутин с указанием что это - исходный текст рутины или компилированный байткод. Примерный экран отчета:

```
Deletion selected routines in database USER

q111                                (source)
q111                                (bytecode)
```

При окончании удаления утилита заканчивает работу.

Если имена рутин указаны без расширения, то используются только стандартные рутины (INT), иначе используются рутины с указанным расширением. В расширении не различаются маски, и могут быть использованы расширения INT, MAC, INC. Пример:

```
Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: MP*.MAC
Routine name mask: f*.INC
Routine name mask: #L
List of selected routine names:
MPPSRC.MAC
first.INC
--------
```

4.20 %RDIR

Утилита ^%RDIR выводит список рутин в текущей базе данных по выбранным маскам.

Использование

```
do ^%RDIR
```

При старте утилита запрашивает перечень масок имен рутин. В маске можно использовать символы подстановки - ? для произвольного символа и * для произвольной последовательности символов. При вводе Пустой строки (нажатие Escape или Enter) утилита заканчивает выбор имен и запрашивает необязательное имя файла. Примерный экран:

```
MiniM Routines listing utility

Routine name mask: *
Routine name mask:
Enter file name to print to (optional):
```

При вводе пустой строки перечень имен рутин выводится на экран, при указании имени файла - в него. Примерный результат вывода списка рутин:

```
List of selected routine's in database USER

PITON
def
job
label
matrix
q111
```

После вывода списка рутин утилита заканчивает работу.

Если имена рутин указаны без расширения, то используются только стандартные рутины (INT), иначе используются рутины с указанным расширением. В расширении не различаются маски, и могут быть использованы расширения INT, MAC, INC. Пример:

```
Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: MP*.MAC
Routine name mask: f*.INC
Routine name mask: #L
List of selected routine names:
MPPSRC.MAC
first.INC
--------
```

4.21 %RESTART

Утилита ^%RESTART завершает работу сервера и запускает его снова.

Использование

```
do ^%RESTART
```

После вызова утилита запрашивает подтверждение на перезапуск сервера.

```
USER>d ^%RESTART
MiniM server restart utility.

Are you sure to restart server? [Y/N]:
```

При вводе ответа утилита завершает работу. Если был ввод Y то отправляется внутренний сигнал останова и сервер завершает работу своих процессов, в том числе того кто вызвал останов, и демонов. Дальнейшая работа в консоли или телнете прерывается. Время завершения зависит от объема работы, необходимой процессам и демонам MiniM для завершения.

После останова сервер снова запускается. Различие с новой запущенной копией может быть в различных настройках как сервера, так и конфигурации баз данных.

Внутренняя точка программного вызова перезапуска сервера

```
d restart^%RESTART
```

4.22 %RESTORE

Утилита ^%RESTORE восстанавливает базу данных их указанного бекап-файла и выводит информацию о бекап файле.

Использование

```
do ^%RESTORE
```

При запуске утилита запрашивает выполняемую операцию - показ информации о бекап файле или восстановление из бекап-файла.

MiniM Database restore utility

```
Select restore option:
 1) View backup info
 2) View database list in backup file
 3) Restore database from backup file
Select option:
```

При выборе отсутствующей альтернативы утилита прекращает работу. При выборе альтернативы 1) View backup info утилита запрашивает имя бекап файла и выводит информацию о нем.

```
Enter backup file name: full.bak

Backup info:
Full backup
File has 2 backed up databases
Backup date 03.06.2009
```

После чего снова запрашивает выполняемое действие.

При выборе альтернативы 2) View database list in backup file утилита запрашивает имя бекап файла и выводит перечень сохраненных в него баз данных, после чего снова запрашивает выполняемое действие.

```
Enter backup file name: full.bak

Backed up database list:
  1) USER
  2) %SYS
```

При выборе альтернативы 3) Restore database from backup file утилита запрашивает имя бекап файла и просит подтвердить тип восстановления

```
Enter backup file name: full.bak

Select restore type:
 1) Full
 2) Differential
Select option:
```

После этого запрашивает операцию, выполняемую с журналом.

```
Select journal operation:
 1) Apply journal records after backup point
 2) Rollback uncomplete transactions from backup point
 3) Skip journal operations
Select option:
```

При выборе первой опции утилита после восстановления баз из бекапа отыскивает в журнале точку бекапа и применяет к базам все журнальные операции до конца журнала. При выборе второй опции утилита после восстановления баз из бекапа отыскивает в журнале точку бекапа и откатывает все транзакции которые в точке быкапа были незакоммичены. При выборе третьей опции утилита ничего не делает с журналом.

Далее утилита запрашивает нужно ли выводить отчет о проведении восстановления.

```
Report option:
 1) Make report
 2) No report
Select option:
```

Если была выбрана опция выводить отчет то утилита запрашивает имя файла для вывода отчета. Если имя файла не указано то отчет выводится на экран.

Далее утилита восстанавливает базы данных из бекапа согласно выбранным опциям. Примерный вид отчета при восстановлении из файла полного бекапа:

```
Checking backup file full.bak
Disabling job starting and freezing active jobs...
Checking job transaction states.
Database to restore from backup: USER
Database to restore from backup: %SYS
Start restoring from backup file full.bak
Start journal phase.
Journal phase skipped.
Enabling job starting and unfreezing active jobs.
Database restore completed successfully.
```

В случае, если при восстановлении из бекапа происходит ошибка, то утилита выводит сообщение как в отчет так и на экран независимо от того, была ли отправка отчета в файл.

При работе утилита приостанавливает работу остальных процессов и отказывается восстанавливать базы данных если на сервере присутствует по крайней мере один процесс в незаконченной транзакции, чтобы предотвратить потерю данных.

Время работы утилиты зависит от количества восстанавливаемых баз, от состояния текущих файлов данных и от количества блоков подлежащих восстановлению. При работе утилита может как увеличивать размер базы данных, так и уменьшать.

Базы данных восстанавливаются в текущей конфигурации экстентов, независимо от конфигурации экстентов, которая была при сохранении базы в бекап.

4.23 %RFIND

Утилита ^%RFIND производит поиск текста в рутинах и выводит имена рутин и найденные в них строки.

Использование

```
do ^%RFIND
```

При старте утилита запрашивает строку, которую нужно найти и опцию поиска - чувствительно к регистру или нет.

```
USER>d ^%RFIND
MiniM Search in routines utility
Search for: *27
Case sensitive search? [Y/N]:
```

После этого рутина запрашивает набор масок имен рутин. В маске имени можно использовать символы подстановки - ? для произвольного исмвола и * для произвольной последовательности символов. При вводе пустой строки (или нажатие Escape или Enter) утилита заканчивает ввод масок имен рутин.

Далее утилита запрашивает необязательное имя файла для вывода отчета. Если указано пустая строка, то отчет о поиске выводится на экран.

Для каждой рутины, имя которой попало в одну из введенных масок имен, выводится ее имя и если в рутине была найдена строка, то выводится имя метки, смещение и строка рутины.

После окончания поиска утилита прекращает работу.

Если имена рутин указаны без расширения, то используются только стандартные рутины (INT), иначе используются рутины с указанным

расширением. В расширении не различаются маски, и могут быть использованы расширения INT, MAC, INC. Пример:

```
Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: MP*.MAC
Routine name mask: f*.INC
Routine name mask: #L
List of selected routine names:
MPPSRC.MAC
first.INC
--------
```

4.24 %RFIRST

Утилита ^%RFIRST выводит первые строки выбранных рутин.

Использование

```
do ^%RFIRST
```

При запуске утилита запрашивает набор масок имен рутин, первые строки которых нужно показать.

```
USER>d ^%RFIRST
MiniM Show routine's first lines utility

Routine name mask: *
Routine name mask:
```

При вводе пустой строки утилита заканчивает ввод масок имен. В маске можно использовать символы подстановки - ? для произвольного символа и * для произвольной последовательности символов.

Далее утилита запрашивает необязательное имя файла, в который нужно вывести результат, список первых строк. Если введена пустая строка, то вывод производится на экран.

Примерный вид списка первых строк:

```
        First lines of selected routines in USER

job                             job ; k  d ^job w
label                           label ; k  d ^label w
matrix                          matrix ; k  d ^matrix w
```

После вывода первых строк утилита заканчивает работу.

Если имена рутин указаны без расширения, то используются только стандартные рутины (INT), иначе используются рутины с указанным расширением. В расширении не различаются маски, и могут быть использованы расширения INT, MAC, INC. Пример:

```
Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: MP*.MAC
Routine name mask: f*.INC
Routine name mask: #L
List of selected routine names:
MPPSRC.MAC
first.INC
--------
```

4.25 %RI

Утилита ^%RI производит импорт исходного текста рутин.

Использование

```
do ^%RI
```

При старте утилита запрашивает имя файла импорта и проверяет открывается ли он на чтение. При вводе пустой строки (или нажатие Escape или Enter) утилита считает что нужно закончить работу.

```
USER>d ^%RI
MiniM routine import
Enter file name to import routines from:
```

После ввода имени файла утилита показывает первые две его строки, считая что в них содержится учетная информация об экспорте. Пример экрана:

```
Enter file name to import routines from:
w:\minim\routines\misc\matrix.rtn
File w:\minim\routines\misc\matrix.rtn has been >>
   >> written with description:
9:10   10-???-2009~Format=ANSI.S~
File timestamp:
```

В зависимости от используемого формата значение строк, первой и второй, может отличаться.

Далее утилита запрашивает какую операцию следует выполнить:

```
Routine input option ([L]ist,[Q]uit,[C]ompile):
```

При выборе команды Q утилита прекращает работу, при выборе команды L утилита показывает список рутин которые входят в указанный файл и при выборе команды C утилита импортирует рутины и компилирует их.

При импорте отчет выводится на экран. Примерный вид отчета об импорте:

```
Routine input option ([L]ist,[Q]uit,[C]ompile): c
Load matrix... compile... success
```

Утилита ^%RI импортирует только стандартные рутины, для импорта макрорутин типов MAC и INC нужно использовать MiniM Control Center или MiniM Routine Editor или утилиту ^%RSAIN.

В различных версиях MiniM экран работы утилиты может незначительно отличаться от приведенного.

4.26 %RIMF

Утилита ^%RIMF производит импорт компилированного байткода рутин.

Использование

```
do ^%RIMF
```

Утилита запрашивает имя файла, который содержит байткод рутин. В одном файле может содержаться байткод нескольких рутин.

```
USER>d ^%RIMF
MiniM routine's bytecode import
Enter file name to import from:
```

При вводе пустой строки утилита заканчивает работу. Иначе предлагает на выбор альтернативы - показать заголовок файла, который был записан при экспорте, провести импорт всех и показать список рутин, байткод которых присутствует в файле. Пример экрана при показе заголовка:

```
USER>d ^%RIMF
MiniM routine's bytecode import
Enter file name to import from:
w:\minim\routines\misc\matrix.mmo
Enter option ([H]eader, [I]mport, [L]ist):h
File w:\minim\routines\misc\matrix.mmo exported with header:
Matrix screen
```

Пример экрана при показе списка рутин в файле:

```
USER>d ^%RIMF
MiniM routine's bytecode import
Enter file name to import from:
w:\minim\routines\misc\matrix.mmo
Enter option ([H]eader, [I]mport, [L]ist):l
matrix
```

При выборе опции импорт утилита импортирует из файла байткод рутин, замещая имеющийся в базе. После этого выполняться будет импортированный байткод, независимо от наличия исходных текстов и их соответствия. При работе процессы MiniM вызывают код рутин не тот, который находится в исходном тексте, а тот, который скомпилирован в байткод и процессам для исполнения не требуется исходный текст рутин за исключением одного случая - для исполнения функции $text(). Если функции $text() обращаются к строкам с двойным комментарием (;;), то исходный текст так же не требуется, поскольку строка с двойным комментарием сохраняется в байткоде.

4.27 %RL

Утилита ^%RL производит импорт исходного текста рутин.

Использование

```
do ^%RL
```

Утилита ^%RL является переходником на утилиту ^%RI для совместимости по именам рутин с другими M системами.

4.28 %RO

Утилита ^%RO производит экспорт исходного текста рутин в файл.

Использование

```
do ^%RO
```

Утилита запрашивает список масок имен рутин, которые нужно экспортировать. В маске имени можно использовать символ подстановки - ? для произвольного символа и * для произвольной последовательности символов. При вводе пустой строки (или нажатии Escape или Enter) утилита заканчивает ввод масок имен. Примерный вид экрана:

```
USER>d ^%RO
MiniM routine export

Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: ma*
Routine name mask:
```

Далее утилита запрашивает имя файла для экспорта. В одном файле экспорта может находиться одна или более рутин, ограничений на длину файла нет. В файле рутины хранятся как текст, без компрессии.

Далее утилита запрашивает описание для файла экспорта или индикатор что нужно сделать файл автоимпорта.

```
Enter description or ^ to make auto-loading file:
```

Описание - это произвольная строка, по которой впоследствии можно понять что содержится в этом файле.

При выборе опции автозагрузки (ввод ^) утилита создает файл со специальным заголовком, который подходит как для импорта обычными средствами, так и для импорта в пакетном режиме, вызывая

```
minim.exe < filename.rou
```

Утилита производит экспорт исходного текста рутин в ANSI формате. При окончании экспорта утилита заканчивает работу. Время работы утилиты определяется количеством данных, которые необходимо экспортировать. Данные для экспорта берутся не мгновенным снимком, а последовательно.

Утилита ^%RO экспортирует только стандартные рутины. Для экспорта макро рутин (MAC, INC) нужно использовать MiniM Control Center или MiniM Routine Editor или утилиту ^%RSAOUT и формат экспорта Cache.

4.29 %ROMF

Утилита ^%ROMF производит экспорт в файл байткода одной или нескольких рутин.

Использование

```
do ^%ROMF
```

Утилита запрашивает маски имен рутин, байткод которых нужно экспортировать. В маске имени рутины можно использовать символы подстановки - ? для произвольного символа и * для произвольной последовательности символов. Если введена пустая строка, то утилита заканчивает ввод масок имен. Примерный вид экрана:

```
USER>d ^%ROMF
MiniM routine's bytecode export
Enter routine's name mask: ma*
Enter routine's name mask:
```

Далее утилита запрашивает имя файла для экспорта. MiniM никак не ограничивает использование произвольных расширений, но можно учитывать, что MiniM Control Center по умолчанию предлагает расширение для файлов байткода рутин mmo.

Далее утилита запрашивает комментарий для файла. Это произвольного вида строка, в которой можно отметить необходимую впоследствии учетную информацию.

После этого утилита производит экспорт выбранных байткодов в указанный файл. Примерный вид экрана при работе утилиты:

```
USER>d ^%ROMF
MiniM routine's bytecode export
Enter routine's name mask: ma*
Enter routine's name mask:
Enter file name to export to:
w:\minim\routines\misc\matrix.mmo
Enter file comment: Matrix screen
Exported 1 routine's bytecode.
```

По окончании экспорта утилита заканчивает работу. Время работы определяется количеством экспортируемых данных.

При работе процессы MiniM вызывают код рутин не тот, который находится в исходном тексте, а тот, который скомпилирован в байткод и процессам для исполнения не требуется исходный текст рутин за исключением одного случая - для исполнения функции $text(). Если функции $text() обращаются к строкам с двойным комментарием (;;), то исходный текст так же не требуется, поскольку строка с двойным комментарием сохраняется в байткоде.

4.30 %RS

Утилита ^%RS производит экспорт исходного текста рутин в файл.

Использование

```
do ^%RS
```

Утилита ^%RS является переходником на утилиту ^%RO для совместимости по именам рутин с другими M системами.

4.31 %RSAIN

Утилита ^%RSAIN производит импорт исходного текста рутин сохраненных в формате RSA.

Использование

```
do ^%RSAIN
```

Формат рутин RSA позволяет различить типы рутин (INC, MAC, INT) и сохранять дату и время их изменения.

При старте утилита запрашивает имя файла импорта и проверяет открывается ли он на чтение. При вводе пустой строки (или нажатие Escape или Enter) утилита считает что нужно закончить работу.

```
USER>d ^%RSAIN
MiniM RSA format routine import
Enter file name to import routines from:
```

После ввода имени файла утилита показывает первые две его строки, считая что в них содержится учетная информация об экспорте. Пример экрана:

```
Enter file name to import routines from:
w:\minim\st\routines\test.rsa
File w:\minim\st\routines\test.rsa has >>
   >>been written with description:
this is description
File timestamp: MiniMRE on jun 09 2011  23:47
```

Далее утилита запрашивает какую операцию следует выполнить:

```
Routine input option ([L]ist,[Q]uit,[C]ompile):
```

При выборе команды Q утилита прекращает работу, при выборе команды L утилита показывает список рутин которые входят в указанный файл и при выборе команды C утилита импортирует рутины и компилирует их.

Утилита дополнительно запрашивает имя файла для формирования отчета. Эта возможность может быть очень полезна при импорте файла с большим числом рутин и для проверки отчета о компиляции если он может быть большого объема. Если не введено имя файла для отчета, утилита выводит результат работы на текущий экран.

Примерный вид отчета об импорте:

```
Routine input option ([L]ist,[Q]uit,[C]ompile): c
Enter file name for report:

Load %BACKUP.INT...
```

```
Load %CONX364.INT...

Compile %BACKUP.INT
Compile %CONX364.INT
```

При импорте рутин утилита сначала импортирует все рутины находящиеся в файле, и лишь после полного импорта компилирует. При компиляции рутины с типом INC не компилируются. Рутины с типом MAC сначала пропускаются через препроцессор, после чего полученный INT код компилируется. Рутины с типом INT просто компилируются. В случае если утилита импорта встречает две рутины с одинаковыми именами но с типами INT и MAC, импорт выполняется для обеих, но из них компилируется только рутина с типом MAC. В случае если для работы препроцессора требуется предварительно скомпилированная одна из рутин, ее необходимо импортировать предварительно из отдельного файла, чтобы при работе препроцессора она уже была готова к использованию.

В различных версиях MiniM экран работы утилиты может незначительно отличаться от приведенного.

4.32 %RSAOUT

Утилита ^%RSAOUT производит экспорт исходного текста рутин в файл в формате RSA с сохранением информации о типе рутины и дате и времени ее модификации.

Использование

```
do ^%RSAOUT
```

Утилита запрашивает список масок имен рутин, которые нужно экспортировать. В маске имени можно использовать символ подстановки - ? для произвольного символа и * для произвольной последовательности символов. При вводе пустой строки (или нажатии Escape или Enter) утилита заканчивает ввод масок имен. Примерный вид экрана:

```
USER>do ^%RSAOUT
MiniM RSA format routine export

Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: A4*.MAC
Routine name mask:
```

Далее утилита запрашивает имя файла для экспорта. В одном файле экспорта может находиться одна или более рутин, ограничений на длину файла нет. В файле рутины хранятся как текст, без компрессии.

Далее утилита запрашивает описание для файла экспорта.

```
USER>do ^%RSAOUT
MiniM RSA format routine export

Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: A4*.MAC
Routine name mask:
Enter description for export:
```

Описание - это произвольная строка, по которой впоследствии можно понять что содержится в этом файле.

Далее утилита запрашивает имя файла в который надо выполнить экспорт.

```
USER>do ^%RSAOUT
MiniM RSA format routine export

Enter name masks or #L to list selected or #D to remove selection.
Routine name mask: A4*.MAC
Routine name mask:
Enter description for export:
Enter file name to export routines:
```

Если было введено пустое имя файла то рутина заканчивает работу.

Далее утилита производит экспорт исходного текста рутин в RSA формате. При окончании экспорта утилита заканчивает работу. Время работы утилиты определяется количеством данных, которые необходимо экспортировать. Данные для экспорта берутся не мгновенным снимком, а последовательно.

4.33 %SHUTDOWN

Утилита ^%SHUTDOWN завершает работу сервера.

Использование

```
do ^%SHUTDOWN
```

После вызова утилита запрашивает подтверждение на завершение работы сервера.

```
USER>d ^%SHUTDOWN
MiniM server shutdown utility.

Are you sure to shutdown server? [Y/N]:
```

При вводе ответа утилита завершает работу. Если был ввод Y то отправляется внутренний сигнал останова и сервер завершает работу своих процессов, в том числе того кто вызвал останов, и демонов. Дальнейшая работа в консоли или телнете прерывается. Время завершения зависит от объема работы, необходимой процессам и демонам MiniM для завершения.

Внутренняя точка программного вызова останова сервера

```
d stop^%SHUTDOWN
```

Глава 5

Макро препроцессор

5.1 Макро рутины

MiniM Database Server начиная с версии 1.6 поддерживает работу с препроцессором макросов в макро рутинах. Стандатные рутины определенные в языке MUMPS называются рутинами непосредственного кода (INTermediate routines), рутины содержащие макросы - макрорутины (MACro routines) и рутины предназначенные для включения при препроцессировании - включаемые рутины (INClude routines).

Имена рутин в MiniM Routine Editor и в MiniM Control Center различаются в спецификации по условному расширению, например:

```
ROUNAME.MAC
ROUNAME.INC
ROUNAME.INT
```

Поддерживаются правила компиляции: 1) INC рутины не компилируются и не порождают исполняемый байткод, 2) INT рутины компилируются в исполняемый байткод и 3) MAC рутины транслируются препроцессором макросов в INT рутину с тем же именем, после чего она компилируется в исполняемый байткод.

Макро рутины MAC и INC начинающиеся с символа процент (%) отображаются по тем же правилам что и процентные INT рутины, хранятся в базе данных %SYS и видны из любой другой базы данных.

Макрорутины по своему назначению являются исходным текстом для получения INT рутины и последующего исполняемого байткода. При

препроцессировании препроцессор сканирует строку за строкой последовательно и, если в строке есть директивы препроцессора, то строка соответствующим образом изменяется или не включается в выходную INT рутину. Рутина с макросами может содержать просто текст на языке MUMPS без директив препроцессора, смешивание MUMPS кода с директивами или одни только директивы на усмотрение программиста.

Рутины разных типов могут иметь одинаковые имена. Текст рутин хранится в разных глобалах. После трансляции MAC рутины полученный INT код доступен для редактирования и просмотра.

Макросы поддерживаются только в макрорутинах. В косвенных выражениях, в аргументе команды *xecute*, в командном режиме и в тегах MWA макросы не поддерживаются.

MiniM Routine Editor поддерживает выделение цветом директив препроцессора для макро рутин и правила трансляции рутин.

MiniM Routine Editor и MiniM Control Center поддерживают расширенный формат экспорта макро рутин в формате Cache RSA (Routine Save Archive) с сохранением различий рутин по типам и датам изменений.

5.2 #define

Директива препроцессора #define определяет имя макроса с возможными его аргументами. Имя макроса должно начинаться на латинскую букву и может содержать латинские буквы и цифры.

Директива имеет два аргумента - определяемый макрос и строку на которую он должен быть заменен при трансляции:

```
#define NAME[(args)] [string for replace]
```

Строка для замены необязательна, если ее нет, то макрос заменяется на пустую строку. Аргументы макроса необязательны. В случае если они есть, то должны указываться в круглых скобках непосредственно после имени макроса и каждый аргумент должен начинаться на символ процент (%) после чего латинская буква и далее содержать латинские буквы или цифры. Например:

```
#define NAME(%arg1) ^ABC("NAME",%arg1)
```

Для использования макроса указывается его имя с предварительными символами $$$. Например:

```
write $$$NAME(123)
write $d($$$NAME(456))
```

преобразуется в INT рутине в строки:

```
write ^ABC("NAME",123)
write $d(^ABC("NAME",456))
```

В случае если макрос содержит аргументы, то при его использовании также должны быть указаны аргументы.

В определении макроса использование его аргументов определяется сопоставлением по именам. Например, если определен макрос:

```
#define NAME(%n,%v) s ^ABC("ind",%n)=%v
```

То при его использовании

```
$$$NAME(123,456)
```

он преобразуется в

```
s ^ABC("ind",123)=456
```

При использовании макросы допускают вложение, допускается указывать макрос в качестве аргумента макроса, например текст:

```
#define A1 "A"
#define B1(%a) BB(%a)
 w $$$B1($$$A1)
```

преобразуется в

```
w BB("A")
```

Если макрос указан в строке или в комментарии, то он не разворачивается, например:

```
#define A1 "A"
#define B1(%a) BB(%a)
 w "$$$B1($$$A1)"
 ; $$$B1($$$A1)
```

преобразуется в

```
 w "$$$B1($$$A1)"
 ; $$$B1($$$A1)
```

Директива препроцессора #define должна занимать строку целиком. При трансляции в выходную INT рутину строка не добавляется.

После того как макрос определен, это имя для препроцессора считается определенным независимо от его аргументов и определенность имени может быть проверена директивами #ifdef и #ifndef.

После того как имя макроса определено, его повторное определение считается ошибкой независимо от указанных аргументов макроса.

Если имя макроса не было определено, то его использование считается ошибкой.

Программист может удалить определение макроса директивой #undef.

5.3 Макро комментарий

Макро комментарий начинается с символов

```
 #;
```

после которых идет комментарий. При препроцессировании вся строка не включается в выходной код INT рутины.

Макро комментарий должен размещаться с начала строки. Если перед символом # присутствуют непробельные символы, то макро комментарий остается в INT рутине как есть и приводит к ошибке синтаксиса.

Пример:

```
;; comment included into bytecode
;; and into INT routine

 ; comment included into INT routine
```

```
   ; but not into bytecode

  #; comment does not included
  #; into INT routine or bytecode
```

5.4 #else

Директива #else переключает условие для условного блока препроцессирования на противоположное. Директива применяется к парной ей предшествующей директиве #if или #ifdef или #ifndef.

Директива #else разделяет условный блок на две части. Если условие начала блока выполнилось, то включаются строки от начинающей директивы #if или #ifdef или #ifndef до #else, а если условие не выполнилось то от #else до закрывающей условный блок директивы #endif.

Пример:

```
#if $zv["MiniM"
 #; code for MiniM
 ...
#else
 #; code for others
 ...
#endif
```

Директивы #if, #ifdef и #ifndef допускают вложение условных блоков друг в друга и препроцессор следит за парностью директив #else и #endif.

5.5 #endif

Директива #endif завершает условный блок открытый директивой #if или #ifdef или #ifndef.

При обработке директивы #endif препроцессор удаляет текущее условие обработки условного блока и обработка продолжается на уровне блока предшествующего уровня или на начальном уровне, без условия.

Пример завершения блока разделенного директивой #else:

```
#if $zv["MiniM"
 #; code for MiniM
 ...
#else
 #; code for others
 ...
#endif
```

Пример завершения блока начатого директивой #ifndef:

```
  #; application definitions
#ifndef incDEFINESINCLUDED
#define incDEFINESINCLUDED

 #define APPDATA(%id)  ^APP("data",%id)
 #define APPIND(%n,%v) ^APP("ind",%n,%v)

#endif
```

Директивы #if, #ifdef и #ifndef допускают вложение условных блоков друг в друга и препроцессор следит за парностью директив #else и #endif.

5.6 #execute

Директива препроцессора #execute выполняет свой аргумент как строку команд. Команды выполняются командой *xecute*.

Макро препроцессор MiniM Database Server использует локальные переменные имеющие префикс имени %mpp. Использовать такие локальные переменные нельзя, это может нарушить работу препроцессора.

В качестве аргумента директивы #execute могут быть использованы произвольные команды, допустимые в контексте команды *xecute*, а также может быть применен вывод в текущее устройство вывода. Использовать текущее устройство для чтения не рекомендуется, поскольку в обычных условиях трансляции вызов препроцессора не предполагает интерактивного взаимодействия с пользователем.

Программист может использовать косвенный результат выполнения директив #execute и использовать их последовательность на свое усмотрение, а также сочетать побочные эффекты директив #execute и #if.

Пример:

```
#if $zv["MiniM"
 #; terminate this job
 k ^$JOB(pid)
#else
#execute w "WARNING: Unknown job termination.",!
#endif
```

В побочных эффектах директив #if и #execute, если была смена текущего устройства, нужно восстанавливать текущее устройство вывода для продолжения нормальной работы препроцессора.

5.7 #if

Директива препроцессора #if определяет начало условного блока. Директива должна иметь обязательный аргумент в виде вычисляемого выражения на языке MUMPS.

Директива #if начинает условный блок обработки. Блок образуется последовательностью строк после директивы до следующей парной директивы #endif. Блок может быть разделен на две части директивой #else.

При обработке MAC или INC рутины препроцессор вычисляет значение аргумента как число и если число не ноль то последующие строки до строки с директивой #else или #endif включаются в результат. Если аргумент вычисляется как 0, то последующие строки не включаются, а если встречена соответствующая директива $else, то включаются строки после нее до директивы #endif.

Пример:

```
#if $r(2)
 w 123,!
#endif
```

Здесь в примере вычисляется значение функции $r(2) и в зависимости от результата включается или нет строка с командой write.

Директивы #if, #ifdef и #ifndef допускают вложение условных блоков друг в друга и препроцессор следит за парностью директив #else и #endif.

Директива #if в отличие от команды *if* вычисляется в момент компиляции и выполняется однократно. Её можно использовать, например, для генерации кода в зависимости от текущего окружения M системы:

```
#if $zv["MiniM"
 #; code for MiniM
 ...
#endif
#if $zv["Cache"
 #; code for Cache
 ...
#endif
```

После компиляции такой MAC рутины в MiniM получается INT рутина с кодом специфичным для MiniM, при компиляции в Cache - с кодом для Cache, при компиляции в другой M системе, поддерживающей препроцессор, - соответствующий код, предусмотренный программистом.

При работе препроцессор использует локальные переменные, имеющие префикс имен %mpp. Использовать такие локальные переменные в вычисляемых выражениях нельзя, это может нарушить работу препроцессора. Программист может использовать в выражениях побочный результат выполнения директивы #execute.

5.8 #ifdef

Директива препроцессора #ifdef определяет начало условного блока. Директива должна иметь обязательный аргумент в виде имени макроса без аргументов.

Директива #ifdef начинает условный блок обработки. Блок образуется последовательностью строк после директивы до следующей парной директивы #endif. Блок может быть разделен на две части директивой #else.

Директива проверяет, было ли определено такое имя ранее. Определенное имя может иметь или не иметь подстановку. Если указанное имя макроса было определено ранее, то препроцессор включает строки после директивы #ifdef до последующей директивы #else или #endif. Иначе препроцессор пропускает строки до последующей директивы #endif, а если встретит #else то включает строки от #else до следующей парной #endif.

Пример:

```
#if $zv["MiniM"
 #define verMiniM
#endif
 ...
#ifdef verMiniM
 ; code for MiniM
 ...
#else
 ; code for other MUMPS
 ...
#endif
```

Директивы #if, #ifdef и #ifndef допускают вложение условных блоков друг в друга и препроцессор следит за парностью директив #else и #endif.

5.9 #ifndef

Директива препроцессора #ifndef определяет начало условного блока. Директива должна иметь обязательный аргумент в виде имени макроса без аргументов.

Директива #ifndef начинает условный блок обработки. Блок образуется последовательностью строк после директивы до следующей парной директивы #endif. Блок может быть разделен на две части директивой #else.

Директива проверяет, было ли определено такое имя ранее. Определенное имя может иметь или не иметь подстановку. Если указанное имя макроса не было определено ранее, то препроцессор включает строки после директивы #ifdef до последующей директивы #else или #endif. Иначе препроцессор пропускает строки до последующей директивы #endif, а

если встретит #else то включает строки от #else до следующей парной #endif.

Пример включаемого файла defines.INC:

```
 #; application definitions
#ifndef incDEFINESINCLUDED
#define incDEFINESINCLUDED

 #define APPDATA(%id)  ^APP("data",%id)
 #define APPIND(%n,%v) ^APP("ind",%n,%v)

#endif
```

Директивы #if, #ifdef и #ifndef допускают вложение условных блоков друг в друга и препроцессор следит за парностью директив #else и #endif.

5.10 #include

Директива препроцессора #include включает указанную в аргументе включаемую рутину. Имя INC рутины должно быть указано без расширения.

Пример:

```
#include common
#include defines
```

При обработке директивы #include препроцессор условно вставляет вместо строки с директивой указанную INC рутину как если бы ее текст был вставлен программистом самостоятельно. Содержание INC рутин может быть произвольным, на усмотрение программиста, в том числе содержать подпрограммы на языке MUMPS, директивы выполнения #execute и директивы определения макросов.

MiniM Database Server не компилирует включаемые INC рутины в непосредственный код INT рутин, такие рутины предназначены только для включения в рутины MAC или в другие рутины INC. Включаемые INC рутины могут иметь имена совпадающие с именами MAC и INT рутин.

Препроцессор MiniM допускает повторное включение включаемых INC рутин, но содержит защиту от рекурсии включения. В случае, если INC рутина допускает рекурсивное включение, рекомендуется использовать защиту от рекурсии и от переопределения уже определенных в этой рутине макросов с помощью макроса с именем, производным от имени INC рутины, например:

```
  #; application definitions
#ifndef incDEFINESINCLUDED
#define incDEFINESINCLUDED

 #define APPDATA(%id) ^APP("data",%id)
 #define APPIND(%n,%v)  ^APP("ind",%n,%v)

#endif
```

Если препроцессор обнаруживает рекурсию включения INC рутины, то выдается однократное предупреждение и рекурсия не выполняется.

5.11 #undef

Директива препроцессора #undef отменяет определение макроса. Директива должна иметь обязательный аргумент в виде имени макроса без аргументов.

Директива проверяет было ли определение указанного в аргументе имени макроса. Имя должно указываться без аргументов. Если имя было определено, то директива отменяет определение независимо от того как оно было определено. Если такого имени не было определено, то директива не имеет побочного результата.

Пример:

```
#define NAME abc
 s $$$NAME=123
#undef NAME
#define NAME def
 s $$$NAME=456
```

Здесь директива #undef отменяет определение макроса NAME и последующее его переопределение не вызывает ошибку трансляции препроцессором.

5.12 Макро функции

MiniM Database Server начиная с версии 1.12 поддерживает работу с макро функциями, расширяющими возможности препроцессора пользовательскими выражениями.

Макро функции могут вызываться в контексте вычисляемых директив препроцессора #if и #execute. В контексте обработки этих директив работает рутина препроцессора %MPP, и макро функции, выполненные в рутине %MPP, используются без указания имени рутины.

Макро функции могут изменить подстановку макроса, проверить существование определения макроса, вернуть значение, и другое. В контексте вычисляемых директив использование макро функций может существенно расширить возможности разработчиков, например организовать логические операции над условиями препроцессирования, сгенерировать значения подстановок времени компиляции и другое.

DEFINED(mname)

Макро функция DEFINED возвращает значение 1 если макрос с именем mname был определен и 0 в ином случае. Имя макроса должно быть в двойных кавычках и задано по правилам языка MUMPS. В том числе могут использоваться операции вычисления имени.

Макро функция DEFINED позволяет организовать логические условия при компиляции макро рутин в зависимости от того, определен ли макрос или нет. Например:

```
#define MACRO1

#if $$DEFINED("MACRO1")&$$DEFINED("MACRO2")
 w "both macro1 and macro2 defined"
#else
 w "one of macro1 or macro2 not defined"
#endif

#define MACRO2

#if $$DEFINED("MACRO1")&$$DEFINED("MACRO2")
 w "both macro1 and macro2 defined"
#else
 w "one of macro1 or macro2 not defined"
#endif
```

Этот код приводит к генерации следующего INT кода:

```
w "one of macro1 or macro2 not defined"
w "both macro1 and macro2 defined"
```

GET(mname,default="")

Макро функция GET возвращает текущее значение подстановки макроса mname либо значение default если макрос не был определен или не имел подстановки.

Например, рутина:

```
#define APPVERSION 3
 ; ...
#if $$GET("APPVERSION")>2
 w "Code for application version greater than 2"
#else
 w "Code for compatibility with version 1"
#endif
```

генерирует код в зависимости от номера версии приложения.

QUOTE(str)

Макро функция QUOTE возвращает аргумент декорированный по правилам языка MUMPS в виде строки. Например, рутина

```
#execute d SET("LIST",$$QUOTE($lb(12,34,56)))
 w $$$LIST
```

вычисляет на этапе компиляции некое сложное выражение и преобразует результат в строковый константный вид:

```
 w $C(3,4,12,3,4)_""""_$C(3,4)_"8"
```

В этом примере сложность выражения весьма условна, а в реальных приложениях использование предвычисления на этапе компиляции может давать ощутимый эффект если такая оптимизация возможна.

SET(mname,msubst)

Макро функция SET создает или заменяет значение подстановки для макроса. Значение подстановки вычисляется при компиляции, что дает возможность вычислять выражения времени компиляции. Например, макрокод:

```
 w "Routine executed at "_$zd($h,3)
#execute d SET("COMPILED",$$QUOTE($zd($h,3)))
 w "Routine compiled at "_$$$COMPILED
```

приводит к генерации кода

```
 w "Routine executed at "_$zd($h,3)
 w "Routine compiled at "_"2011-12-25"
```

Здесь первая строка при выполнении рутины выводит время выполнения рутины, вторая строка выводит время компиляции рутины. Разработчики могут предвычислять множество констант, использовать собственные функции генерации подстановок.

EVAL(expr)

Макро функция, обратная к макрофункции QUOTE, вычисляет значение выражения заданного в виде строки языка MUMPS. Например, при компиляции макро кода

```
#define MACROEXPR 1+2+3
#execute d SET("EXPAND",$$EVAL($$GET("MACROEXPR")))
 w $$$EXPAND
```

генерируется INT код с подстановкой вычисленного значения макроса:

```
 w 6
```

В этом примере определяется подстановка макроса MACROEXPR в виде строки 1+2+3, далее берется значение подстановки (GET("MACROEXPR")), вычисляется как выражение языка MUMPS (EVAL), и результат записывается как подстановка макроса EXPAND (SET). В итоге макрокод может использовать макрос EXPAND содержащий уже вычисленное выражение макроса MACROEXPR.

В контексте вычисляемых директив препроцессора #if и #execute программист может использовать как сложные комбинирования макро функций, так и вызывать собственные функции генерации макроподстановок и имен макросов.

Глава 6

MiniMono

6.1 Архитектура MiniMono

MiniMono - это короткое название специальной однопользовательской редакции MiniM, полное название MiniM Embedded Edition. MiniM Embedded Edition представляет собой библиотеку динамической компоновки с модулями сервера MiniM. Предназначена для использования в персональных программах для внедрения в приложение возможностей языка MUMPS и функций MiniM.

В настоящее время MiniMono поддерживается в тех же редакциях, то и полный сервер MiniM Database Server - для Windows и Linux, включая консольные и графические средства. Примеры использования MiniM Embedded Edition входят в комплекты всех редакций для поддерживаемых операционных систем.

Приложение создает виртуальную машину MiniMono, обращается к ней для выполнения команд, вычисления выражений и для изменения переменных. В контексте хост процесса приложения MiniMono работает как процесс MiniM и несколько демонов. Все выполняемые MiniMono задачи работают в дочерних вторичных потоках. Потоки демонов запускаются в случае если это определено в данных инициализации. Виртуальная машина MiniMono может работать с одной базой данных хранящейся в том же формате что и в MiniM Database Server.

При работе MiniMono доступны все возможности полного MiniM Database Server за исключением запуска фоновых процессов и организации многофайловой базы данных.

В полном MiniM Database Server выполнены специальные средства улучшения стабильности базы данных и полный сервер работает на

уровне стабильности операционной системы плюс средства восстановления по предзаписи и восстановления из бекапа с журналом. Модуль MiniMono как библиотека ограниченная контекстом хост процесса работает только на уровне стабильности приложения плюс восстановление по предзаписи или восстановления из бекапа.

MiniMono содержит в своем составе код демонов расширения базы данных, демон записи измененных блоков и демон журнала. Разработчик хост приложения должен понимать, что если в полном MiniM Database Server работают средства защиты процессов и демонов от сбоев (гвардианы процессов), то для потоков MiniMono таких средств защиты нет и в случае сбоев хост процесса разработчик может использовать лишь средства автоматического восстановления по предзаписи или восстановление из бекапа. Также как и для полного MiniM Database Server, для MiniMono поддерживается горячий бекап.

Общая последовательность работы с модулем MiniMono: 1) получить значения инициализации виртуальной машины по умолчанию, 2) скорректировать значения по умолчанию в соответствии с настройками и потребностями приложения, 3) создать виртуальную машину MiniMono, 4) обращение к СУБД и 5) завершение работы модуля MiniMono.

MiniMono не хранит собственный настроек в каких-либо файлах или реестре. Считается что хранением настроек в соответствующем виде занимается хост приложение. Одна библиотека MiniMono может обслуживать несколько приложений если они используют разные базы данных и не дает возможность использовать одну базу данных нескольким приложениям одновременно. Каждая виртуальная машина MiniMono содержит в себе автономные модули сервера MiniM, собственные кеши и демоны записи, поэтому с одной базой данных несколько виртуальных машин работать не должны.

В одном хост приложении может быть создана только одна виртуальная машина MiniMono. Разработчику хост приложения необходимо понимать, что в адресном пространстве его процесса будет присутствовать множество вторичных объектов MiniMono и что при некорректной работе хост процесс может повредить внутренние структуры данных MiniMono.

MiniMono не реализует собственный интерфейс взаимодействия с пользователем, будь то алфавитно-цифровой или оконный режим. Библиотека MiniMono в качестве устройства ввода-вывода по умолчанию создает специальное устройство DLL, а приложение должно объявить

обработчики событий для этого устройства. Само хост приложение может быть консольным, оконным или иным и самостоятельно определять поведение устройства DLL.

Все данные и программы в MiniMono хранятся в одной базе данных и MiniMono работает с базой данных как сервер MiniM с базой данных "%SYS". К общим рекомендациям может относиться инсталляция отдельного инструментального экземпляра MiniM Database Server, подготовка на нем базы данных "%SYS" и дальнейшее использование файла данных совместно с хост приложением. Разработчик хост приложения может изменять состав системных рутин на свое усмотрение или совсем не использовать рутины.

Для MiniMono отладчик не поддерживается, поэтому при необходимости отладку рутин требуется проводить на отдельном инструментальном экземпляре MiniM Database Server.

Функции и возможности MiniMono соответствуют полному MiniM Database Server соответствующей версии. Также как в MiniM Database Server могут быть использованы внешние модули ZDLL и ZDEVICE, системные функции, транзакции, бекап, системные и пользовательские рутины. Введенные ограничения MiniMono перечислены в отдельной статье "Перечень отличий".

MiniMono устанавливается в комплекте с примерами и утилитами:

minimonostd.exe	Утилита командной строки получающая ввод через stdin и выдающая результат в stdout
minimonocon.exe	Консольная утилита, получающая ввод с консоли и выводящая на консоль с процессированием эскейп-последовательностей
minimonore.exe	MiniMono Routine Editor, утилита редактирования, компилирования, экспорта и импорта рутин, байткода и глобалов
minimonoge.exe	MiniMono Global Editor, утилита редактирования и просмотра глобалов
minimne.exe	Утилита редактирования файлов определения символов

Для MiniMono используется та же документация, что и для MiniM Database Server.

Для инсталляции библиотеки MiniMono достаточно скопировать на целевой компьютер файлы minimono.dll в каталог доступный для хост

приложения. Никаких дополнительных шагов по регистрации библиотеки не требуется.

6.2 Структуры данных

MiniMono использует те же структуры данных что и модули ZDLL и те же соглашения о передачи и кодировании данных.

Для передачи значений используется структура данных MINIM_STR. В ней может размещаться как последовательность байт так и числа. Тип размещаемых данных указывается служебным полем len или type. Хост приложение может аллокировать память не на всю длину структуры MINIM_STR, а только на ее значимую часть, например если используется лишь до 100 байт данных, то можно аллокировать 100 байт плюс место для служебного поля длины.

При передаче и приеме данных в контекст MiniMono хост приложение самостоятельно создает структуры в необходимом ему классе памяти. При вызове хост приложения из MiniMono виртуальная машина предоставляет указатели на свои внутренние структуры данных.

Для взаимодействия с виртуальной машиной MiniMono хост приложение использует ту же структуру данных для вызова контекста MiniM что и модуль ZDLL, структура ZDLLCB. Виртуальная машина MiniMono создает указатель на эту структуру при создании экземпляра виртуальной машины. Структура содержит набор указателей на функции для вызова контекста MiniMono. Если для контекста ZDLL модуля эта структура является структурой обратного вызова, то для хост приложения это структура прямого вызова.

Для инициализации виртуальной машины MiniMono используется структура MINIMONOVM. В ней хост приложение указывает параметры виртуальной машины. Все поля структуры кроме поля cbfunc заполняет хост приложение, а поле cbfunc заполняет сама виртуальная машина MiniMono, записывая реальные указатели на свои функции.

Для простоты использования библиотека MiniMono предоставляет функцию заполнения структуры MINIMONOVM значениями по умолчанию. Значения по умолчанию выбраны по характерным параметрам большинства используемых приложений. После получения значений по умолчанию хост приложение должно указать собственные особенности использования MiniMono, реальное расположение файлов данных и обработчики устройства DLL если они используются.

Обработчики устройства DLL полностью аналогичны таким же обработчикам устройств ZDEVICE за исключением: 1) устройство DLL может быть только одно и обработчики не имеют контекста текущего устройства и 2) устройство не имеет обработчиков open и close.

Перечень полей структуры инициализации MINIMONOVM:

DataFile	Имя файла базы данных, может быть указан полный или относительный путь. По умолчанию пусто, хост приложение должно указать имя файла данных.
ReadOnly	Указывает используется ли база данных в режиме только чтение (1) или разрешена запись (0). В случае использования базы данных только на чтение демоны записи вообще не запускаются. По умолчанию 0, запись разрешена.
JournalingEnabled	Разрешено ли журналирование операций с глобалами (1) или нет (0). Поле используется для баз данных для которых разрешена запись. При значении 0 демон журнала не запускается и откат изменения данных при откате транзакций невыполним. По умолчанию 1, журналирование используется.
LockAreaSize	Размер таблицы блокировок для команды lock в мегабайтах. Поскольку MiniMono работает в монопольном режиме, то оценка размера должна делаться на используемые в рутинах команды lock. Необходимости в них нет, поскольку приложение монопольное, но код может содержать такие команды для совместимости. В случае исчерпания таблицы процесс будет ожидать освобождения места но такое действие выполнимо лишь в контексте MiniM Database Server, поэтому разработчики хост приложений должны указывать достаточный размер для своего приложения. По умолчанию 1 MB.

RoutineCacheSize	Размер кеша байткода в мегабайтах. Если приложение использует небольшое число рутин, то может быть указано небольшое значение. Если в приложении используется много рутин, то увеличение этого значения может быть некоторый прирост скорости. По умолчанию 1 MB.
DeviceTableSize	Число устройств ввода-вывода которое может быть использовано в виртуальной машине MiniMono. По умолчанию 4.
DeviceNameSize	Длина имени устройства ввода-вывода, которое используется для различения различных устройств. По умолчанию 400 байт.
DBCacheSize	Размер кеша глобалов в мегабайтах. Рекомендуется использовать объем кеширования учитывающий что виртуальная машина MiniMono должна разместить свои структуры в адресном пространстве хост приложения а также реальный размер физической памяти на компьютере. При его существенном превышении может наблюдаться некоторое замедление работы из-за обращения к механизму подкачки. По умолчанию 100 MB.
NullSubscripts	Разрешено ли (1) или нет (0) использование пустых строк в качестве индексов. По умолчанию 0, MiniMono генерирует ошибку при использование пустых индексов.
TransactLevelLimit	Максимальное число вложенных транзакций, или предельный уровень вложения транзакций. По умолчанию 255.
TrapOnEof	Нужно ли (1) генерировать ошибку <ENDOFFILE> или нет (0) если используемое устройство достигло виртуального состояния "конец файла". По умолчанию 1, MiniMono генерирует ошибку при обнаружении состояния "конец файла".
FrameCount	Максимально возможное число стеков для вызова подпрограмм или xecute. По умолчанию 1024.

JournalCache	Размер кеша журнала в мегабайтах. По умолчанию 4 MB.
LocaleFileName	Имя файла определения символов или нулевое значения при использовании определения символов по умолчанию. По умолчанию NULL, хост приложение может указать собственное определение символов, использовать одно из стандартно устанавливаемых с MiniM или не использовать никакого, в этом случае MiniMono использует определение символов по умолчанию.
ProcessStorage	Размер пространства хранения локальных переменных в мегабайтах. По умолчанию 8 MB.

Обработчики виртуального устройства DLL по умолчанию инициализируются нулевыми указателями. Если хост приложение указывает обработчики, то они вызываются программой на MUMPS при обращении к устройству DLL.

6.3 Прямой вызов

Прямой вызов виртуальной машины MiniMono предназначен для выполнения последовательности команд, вычисления выражений языка MUMPS, преобразования данных, для операция с локальными или глобальными переменными и для вызова подпрограмм.

Хост приложение для каждого вызова виртуальной машины должно разместить структуры данных для параметров в своей памяти и в случаях когда принимает значения также разместить структуры данных для приема данных. Интерфейс MiniMono не предполагает отдельных функций для аллокации и освобождения памяти, все опрации производятся на структурах принадлежащих либо хост приложению либо виртуальной машине MiniMono.

Прямой вызов виртуальной машины программно совпадает с интерфейсом модулей ZDLL и номера ошибок возвращаемых функциями имеют префикс ZDLL.

Для получения структуры указателей на функции хост приложение должно создать виртуальную машину MiniMono. При этом виртуальная

машину устанавливает указатель на свои функции в поле cbfunc структуры MINIMONOVM.

После того как виртуальная машина MiniMono завершена, обращение по указателям этой структуры запрещено.

6.4 Обратный вызов

При выполнении прямого вызова виртуальная машина MiniMono выполняет указания и при работе программа на MUMPS может обратиться к устройству DLL. При этом производится так называемый обратный вызов.

При обратном вызове виртуальная машина MiniMono вызывает указанные хост приложением обработчики. В случае если обработчик не указан, виртуальная машина MiniMono выполняет действия по умолчанию.

Передаваемые обработчикам структуры данных принадлежат виртуальной машине MiniMono и хост приложение не должно их модифицировать или делать какие-либо предположения о доступности байт за пределами их реально используемого количества.

Все обработчики должны возвращать значение 0 при успехе выполнения либо не 0 при неудаче. При возврате ненулевого значения виртуальная машина MiniMono генерирует ошибку выполнения.

Все обработчики при своей работе также могут в свою очередь обращаться к контексту виртуальной машины MiniMono для преобразования данных или выполнения команд или вычисления выражений.

Поведение обработчиков, их наличие, поддерживаемые параметры команды use и трактовка состояния EOF полностью определяются хост приложением.

Обработчики чтения строки (тип dlldevreadstr_t) и чтения символа (тип dlldevreadchar_t) получают в качестве значения таймаута величину таймаута в миллисекундах. Если в командах чтения параметр таймаута не был указан, виртуальная машина MiniMono передает значение -1. Если длина чтения не была указана, передается значение -1.

Виртуальная машина MiniMono предполагает, что разработчик хост приложения самостоятельно и взаимосогласованно реализует реакцию

на состояние EOF в функциях чтения и в обработчике получения состояния EOF. Обработчик получения состояния EOF может быть вызван виртуальной машиной MiniMono по своему усмотрению неоднократно.

Среди обработчиков устройства DLL нет обработчиков команд open и close. Это устройство является устройством ввода-вывода по умолчанию и не может быть создано или закрыто, и не может иметь параметров закрытия. Хост приложение должно самостоятельно инициализировать управляющие структуры связанные с устройством DLL перез созданием виртуальной машины и самостоятельно деинициализировать после завершения MiniMono.

6.5 Перечень отличий

MiniM Embedded Edition содержит в себе всю функциональность полного варианта MiniM Database Server за исключением перечисленных далее.

MiniMono архитектурно не содержит средств повышения стабильности и защиты процессов (гвардианы процессов).

MiniMono не использует внешних настроек сохраняемых в файлах или в реестре, все настройки хост приложение должно хранить и использовать самостоятельно.

MiniMono не поддерживает устройства типов CON, STD, TNT, поскольку интерфейс взаимодействия с пользователем хост приложение выполняет самостоятельно. MiniMono не накладывает ограничений на тип хост приложения, это может быть как консольное так и оконное приложения.

MiniMono не поддерживает устройство типа MEM поскольку это устройство является ориентированным на межпроцессное взаимодействие, а MiniMono поддерживает только один процесс MUMPS.

MiniMono автоматически создает служебные файлы лога minim.log, предзаписи minim.bij и журналов в том же каталоге где находится обслуживаемый файл данных. Поэтому хост приложение должно размещать базу данных в отдельном каталоге. Размещение нескольких баз данных для разных приложений в одном каталоге допустимо если виртуальные машины MiniMono будут запускаться в режиме только на чтение данных.

Команда JOB не поддерживается, поскольку MiniMono является однопроцессной реализацией MiniM.

Отладчик MiniM не поддерживается, поскольку в MiniM Database Server отладчик архитектурно является многопроцессным.

Системная переменная $JOB возвращает номер потока операционной системы, а не номер процесса, как в MiniM Database Server.

Системная переменная $ZPARENT всегда возвращает значение 0, поскольку у MUMPS процесса виртуальной машины MiniMono нет другого родительского процесса MiniMono.

Системная переменная $ZVERSION содержит имя продукта MiniMono а не MiniM для того чтобы программы могли различить контекст работы.

Имя инсталляции MiniMono всегда равно "MINIMONO". Это виртуальное имя инсталляции, его имеют все одновременно работающие экземпляры MiniMono. При этом на компьютере может быть установлен полный MiniM Database Server с таким же именем инсталляции, и конфликта в работе виртуальных машин MiniMono между собой и с таким сервером MiniM не происходит.

Файл определения символов для MiniMono может размещаться в любом месте и иметь произвольное имя, в отличие от MiniM Database Server нет ограничений на его расположение.

MiniMono поддерживает только одну базу данных и она должна состоять только из первичного файла данных, экстенты не поддерживаются. Размер файла данных может быть произвольным. Имя базы данных всегда "%SYS".

MiniMono не подерживает отображение глобалов и рутин в другие базы данных, поскольку поддерживается только одна база данных.

Дополнительно к устройствам MiniM Database Server поддерживается специальное виртуальное устройство типа DLL. Устройство создается автоматически как устройство ввода-вывода по умолчанию. Хост приложение должно указать какие события этого устройства поддерживаются. Для неподдерживаемых событий MiniMono использует поведение по умолчанию, как для NULL устройства - запись в устройство игнорируется, чтение возвращает управление немедленно, системные переменные для устройства возвращают значения по умолчанию. Поведение обработчиков событий устройства DLL полностью определяется хост приложением и MiniMono не накладывает на характер ввода-вывода никаких ограничений.

Команда HALT в MiniMono не завершает процесс хост приложения, а завершает текущее обращение к MiniMono. После того как обращение к MiniMono, например, выполнение команд или вычисление выражения завершилось командой HALT, хост процесс может снова обратиться к виртуальной машине MiniMono.

MiniMono не экспонирует статистики виртуальной машины как счетчики производительности (в версии для Windows), но значения всех счетчиков производительности по-прежнему доступны через системные функции $view("perf").

Для работы виртуальной машины MiniMono не требуется лицензирование, это бесплатное программное обеспечение.

В общей документации по языку MiniM также приведены отличия MiniMono от полного сервера в тех функциях, командах и системных переменных где они есть.

6.6 MiniMono CHUI Tools

В комплект MiniMono Embedded Edition входит две CHUI утилиты, minimonostd.exe и minimonocon.exe. Это CHUI оболочки для библиотеки MiniMono.

Утилита minimonostd.exe для ввода-вывода использует стандартные каналы консольного процесса - stdin и stdout. Ввод получаемый на stdin разбивается на отдельные строки и направляется на выполнение как строки команд и вывод выполняемый командами write выполняется на стандартный выход stdout.

Утилита minimonostd.exe может быть использована для организации командных файлов с использованием перенаправления ввода-вывода.

По своим параметрам утилита полностью аналогична устройству |STD| полного MiniM Database Server.

Утилита minimonocon (minimonocon.exe для Windows и minimonocon для Linux) для ввода использует ввод с клавиатуры и выводит на консоль. При выводе используется процессирование эскейп-последовательностей. При старте процесса текущему устройству |DLL| назначается рутина обработки мнемоник "%CONX364". Для того, чтобы обработка мнемоник работала, необходимо, чтобы в используемую базу данных была импортирована с компиляцией эта рутина.

По своим параметрам утилита полностью аналогична устройству |CON| полного MiniM Database Server.

Для задания параметров работы утилит обе утилиты поддерживают параметры командной строки. В них задаются имя файла базы данных, начальные действия при старте и параметры процесса.

Опция -d datafile задает имя файла базы данных MiniM, с которым должен работать модуль MiniMono. Опция обязательна.

Опция -n natfile задает имя файла сравнения и перекодирования символов. Если опция не указана, то minimono.dll использует сравнение символов по умолчанию. Если программа используется для редактирования глобалов с национальными символами в индексах, эта опция необходима, иначе возможно формирование данных, некорректных для других программ, использующих определение символов.

Опция -r readonly указывает должен ли процесс только читать днные базы данных, значение readonly рассматривается как число и сравнивается с 0. Если 0, то база данных используется только на чтение. Опция необязательна, если не указана то запись в базу данных разрешается.

Опция -c cacheinmegbytes задает размер кеша базы данных в мегабайтах. Если опция не задана, то используется значение по умолчанию.

Опция -j journaling разрешает или запрещает журналирование изменений базы данных. Значение рассматривается как число и сравнивается с нулем. Если значение 0, то журналирование отключается. По умолчанию, если опция не задана, то журналирование разрешено. При отключении журналирования команда trollback технически не сможет выполнить откат изменений сделанных в транзакции.

Опция -p showprompt задает нужно ли при вводе очередной строки предварять ее стандартным промптом. По умолчанию промпт показывается. При выполнении пакетных команд из параметров промпт в текущий вывод не выводится.

Опция -x xecute задает строку команд которую нужно выполнить при старте процесса.

Опция -x @xecutefile задает имя файла, содержание которого рассматривается как последовательность строк, и эти строки выполняются одна за другой при старте процесса.

Опция -h задает нужно ли прекратить выполнение процесса после старта и выполнения параметров опции -x. Если опция не указана, то утилиты переходят к вводу строки команд.

Пример использования CHUI утилит - это создать командный файл и в нем указать опции запуска, например:

```
minimonostd.exe -d empty.dat
```

Или

```
start minimonocon.exe -d sys.dat -j 0 -x "d ^%aNC"
```

В отличие от полной версии MiniM Database Server, CHUI утилиты minimonostd.exe и minimonocon.exe начинают выполняют на уровне стека не 0, а 1, в силу архитектуры модуля MiniMono.

6.7 MiniMono GUI Tools

В комплект MiniMono Embedded Edition входит две GUI утилиты, minimonore.exe и minimonoge.exe. Это GUI оболочки для библиотеки MiniMono.

Утилита minimonore.exe это полный функциональный аналог MiniM Routine Editor, но для MiniMono, с отключенной поддержкой клиентской части отладчика MiniM, поскольку в архитектуре MiniMono отладчик не поддерживается.

В остальном утилита MiniMono Rouine Editor полностью аналогична утилите MiniM Routine Editor, включая редактирование и компиляцию рутин, препроцессор, экспорт и импорт рутин и данных.

Утилита MiniMono Routine Editor поддерживает ключи командной строки для задания имени файла данных и имени рутины.

Опция -d datafile задает имя файла базы данных MiniM, с которым должен работать модуль MiniMono. Опция необязательна. Если опция не указана, то утилита предлагает выбрать файл данных.

Опция -n natfile задает имя файла сравнения и перекодирования символов. Если опция не указана, то minimono.dll использует сравнение символов по умолчанию. Если программа используется для редактирования глобалов с национальными символами в индексах, эта опция необходима, иначе возможно формирование данных, некорректных для других программ, использующих определение символов.

Опция -r routine задает имя рутины, с которой должен работать MiniMono Routine Editor. Опция необязательна.

Пример:

```
minimonore.exe -d sys.dat -r %BACKUP
```

Утилита minimonoge.exe это MiniMono Global Editor, полный аналог утилиты MiniM Global Editor, но для модуля MiniMono.

Утилита MiniMono Global Editor поддерживает ключи командной строки для задания имени файла данных и имени глобала.

Опция -d datafile задает имя файла базы данных MiniM, с которым должен работать модуль MiniMono. Опция необязательна. Если опция не указана, то утилита предлагает выбрать файл данных.

Опция -g global задает имя глобала, с которой должен работать MiniMono Global Editor. Опция необязательна.

Опция -n natfile задает имя файла сравнения и перекодирования символов. Если опция не указана, то minimono.dll использует сравнение символов по умолчанию. Если программа используется для редактирования глобалов с национальными символами в индексах, эта опция необходима, иначе возможно формирование данных, некорректных для других программ, использующих определение символов.

Пример:

```
minimonoge.exe -d sys.dat -g ^ROUTINE
```

6.8 MiniMonoX

Библиотека MiniMonoX.dll представляет собой ActiveX интерфейс к библиотеке MiniMono и преобразует обращения OLE Automation в обращения к программному интерфейсу MiniMono. Для работы компонента на клиентском компьютере должны быть размещены одновременно в доступном каталоге компоненты minimonox.dll и minimono.dll. Библиотека MiniMonoX должна быть зарегистрирована после инсталляции

```
regsvr32 minimonox.dll
```

В зависимости от назначения инсталлятора могут быть использованы различные ключи команды regsvr32 а также специальные вызовы API, функционально аналогичные регистрации компонента.

Библиотека MiniMonoX выполнена как в варианте x86-32, так и в варианте x86-64. Для использования MiniMonoX должна быть установлена той же битности, что и используемая версия MiniMono.

Библиотека MiniMonoX реализует два объекта ActiveX:

```
MiniMono.ServerString
MiniMono.VM
```

Тип MiniMono.VM предназначен для соединения и работы с виртуальной машиной MiniMono, вызовов MiniMono, а также получения событий - обработчиков устройства по умолчанию. Внутренние объекты типа MiniMono.ServerString предназначены для хранения и преобразования данных между интерфейсом OLE Automation и внутренним форматом, используемым MiniMono. Часть функций MiniMono.VM имеет дублирование функций с суффиксом Str. Если основная функция работает с данными в формате MiniMono.ServerString, то функции с суффиксом Str работают с обычными строками OLE Automation (тип BSTR). Тип MiniMono.ServerString может хранить и использовать данные, недопустимые для стандартных строк, в частности содержать любые символы, включая строки в формате функций $lb(), нулевые байты, а также корректно преобразует данные из представления юникод, принятого в OLE Automation, в стандартные ANSI - последовательности байт.

Архитектура виртуальной машины MiniMono накладывает дополнительные ограничения на обращение к ней из клиентской программы - 1) объект MiniMono.VM может быть создан в клиентской программе только в единственном экземпляре и 2) с файлом данных может работать только одна виртуальная машина MiniMono.

Общая схема работы

Общая схема работы с MiniMonoX состоит из создания и удаления объекта типа MiniMono.VM, инициализации объекта указанием файла данных, локализации, настроек кешей и других, создания и завершения самой виртуальной машины и обращений к созданной виртуальной машине. Объекты типа MiniMono.ServerString, а также строки используются для передачи и преобразования данных.

Пример на VBS:

```
' create ActiveX object
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")
' initialize virtual machine
MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0
' create virtual machine
```

```
MiniMono.CreateVM
' ...
' calls to MiniMono virtual machine
' ...
' free virtual machine
MiniMono.FreeVM
' free ActiveX object
Set MiniMono = Nothing
```

Для использования в программе определенного поведения устройства по умолчанию нужно установить обработчики соответствующих событий:

```
Dim MiniMono
Set MiniMono=WScript.CreateObject("MiniMono.VM","MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM
MiniMono.ExecuteStr "w 123"
MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevWriteStr( Str)
  WScript.Echo str.Value
End Sub
```

Здесь функция WScript.CreateObject используется в двухаргументной форме с заданием префикса имен обработчиков событий. Имена обработчиков событий программы в VBS состоят из заданного префикса и самого имени события, например

```
DevWriteStr
DevWriteChar
DevWriteNL
```

В качестве префикса может быть использована любая строка на усмотрение разработчика. В иных средах исполнения обработчики событий устанавливаются соответствующим этим средам образом.

Виртуальная машина MiniMono может быть создана лишь одна на процесс и устройство по умолчанию может быть лишь одно, поэтому параметры обработчиков событий этого устройства по умолчанию не содержат идентификации самой виртуальной машины и устройства.

6.8.1 Свойства объекта MiniMono.VM

Свойства объекта MiniMono.VM должны устанавливаться между созданием самого объекта и созданием виртуальной машины MiniMono. Свойства используются для инициализации виртуальной машины. После создания свойства по-прежнему доступны на чтение и запись, но до завершения виртуальной машины не используются.

Пример:

```
Dim MiniMono
' create object
Set MiniMono = WScript.CreateObject( "MiniMono.VM")
' assign initial values
MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0
' create virtual machine
MiniMono.CreateVM
```

`DataFile`

Свойство DataFile имеет тип строка и указывает на файл данных, который должен использоваться виртуальной машиной MiniMono в качестве базы данных. Положение файла данных может задаваться как с указанием абсолютного пути, так и относительного относительно текущего каталога, либо файл данных должен находиться в одном из каталогов указанных в переменной окружения PATH.

Указание имени файла базы данных обязательно.

`LocaleFileName`

Свойство LocaleFileName имеет тип строка и указывает на файл локализации символов NAT, который должен использоваться для сортировки символов в индексах и операций поднятия - опускания регистра, а также для определения того, является ли символ буквой. Положение файла данных может задаваться как с указанием абсолютного пути, так и относительного относительно текущего каталога, либо файл данных должен находиться в одном из каталогов указанных в переменной окружения PATH.

Свойство необязательно, и если не указано, то используются операции сравнения по умолчанию.

`ReadOnly`

Свойство ReadOnly имеет тип целое число и если равно не нулю, то база данных используется в режиме только чтения.

Свойство необязательно для заполнения, по умолчанию база данных работает в режиме чтение - запись.

`JournalingEnabled`

Свойство JournalingEnabled имеет тип целое число и если равно нулю, то журналирование базы данных отключается.

Свойство необязательно для заполнения, по умолчанию журналирование включено. При отключении журналирования откат транзакций не выполняется, команда trollback ничего не выполняет.

`LockAreaSize`

Свойство LockAreaSize имеет тип целое число и указывает размер области памяти отводимый витуальной машиной для хранения информации о блокировках в мегабайтах.

Поскольку виртуальная машина MiniMono архитектурно является монопольной, использование блокировок в программах, ориентированных на MiniMono необязательно.

Свойство необязательно для заполнения, по умолчанию используется 1 мегабайт. Пределы значения находятся между 1 и 64 мегабайт.

`RoutineCacheSize`

Свойство RoutineCacheSize имеет тип целое число и указывает размер кеша байткода рутин в мегабайтах.

Свойство необязательно для заполнения, по умолчанию используется 1 мегабайт. Пределы значения находятся между 1 и 64 мегабайт.

`DeviceTableSize`

Свойство DeviceTableSize имеет тип целое число и указывает количество устройств, которое может одновременно открыть один процесс. Свойство необязательно для заполнения, значение по умолчанию 16. Допустимые значения - от 4 до 1000.

`DeviceNameSize`

Свойство DeviceNameSize имеет тип целое число и задает максимальную длину имени устройства, которое может использовать процесс, в символах. Свойство необязательно для заполнения, значение по умолчанию 400.

`DBCacheSize`

Свойство DBCacheSize имеет тип целое число и указывает размер кеша глобалов в мегабайтах. Значение по умолчанию 100 мегабайт. Минимальное значение 1 мегабайт. Для 32-битных версий максимальное значение 1 гигабайт, для 64-битных версий максимальное значение не ограничивается.

`NullSubscripts`

Свойство NullSubscripts имеет тип целое число и указывает разрешены ли пустые строки в качестве значений индексов глобалов и локальных переменных. Если указано значение 0, то не разрешены. Иное значение означает что разрешены. По умолчанию для совместимости программ пустые строки в индексах не разрешены.

`TransactLevelLimit`

Свойство TransactLevelLimit имеет типа целое число и задает число максимально допустимых вложений транзакций. Значение по умолчанию 255, минимальное 1, максимальное 32000.

TrapOnEof

Свойство TrapOnEof имеет тип целое число и задает поведение устройств по умолчанию при обнаружении конца чтения. Если значение не ноль, то генерируется ошибка. Если значение 0, то взводится системная переменная $zeof. По умолчанию для совместимости программ при достижении конца чтения генерируется ошибка. Поведение также зависит от типа устройства и соглашение применяется только к устройствам, физический смысл которых допускает такое понятие как конец чтения, например файл или устройство определенное во внешней DLL / SO (ZDEVICE).

FrameCount

Свойство FrameCount имеет тип целое число и задает максимальное число уровней стека при исполнении программ. Свойство необязательно для указания, по умолчанию равно 1024. Минимальное значение 16, максимальное 131072.

JournalCache

Свойство JournalCache имеет тип целое число и задает размер кеша журнала в мегабайтах. Свойство необязательно для заполнения, значение по умолчанию 8 мегабайт. Минимальное значение 1 мегабайт, максимальное для 32-битных систем 64 мегабайт. Для 64-битных систем максимальное значение не ограничивается.

ProcessStorage

Свойство ProcessStorage имеет тип целое число и задает размер области для хранения локальных переменных в мегабайтах. Свойство необязательно для заполнения, значение по умолчанию 8 мегабайт. Минимальное значение 1 мегабайт, максимальное для 32-битных версий 64 мегабайт, для 64-битных версий максимальное значение не ограничивается.

6.8.2 Функции объекта MiniMono.VM

`CreateVM`

Функция CreateVM создает виртуальную машину MiniMono согласно значениям указанным в инициализирующих свойствах. Виртуальная машина MiniMono может быть создана лишь одна на процесс операционной системы и лишь одна на файл данных.

Возвращаемые значения: 0 если виртуальная машина создана успешно, 1 если виртуальная машина для указанной базы данных уже создана (в этом или ином процессе) и 2 при других ошибках - недостаток оперативной памяти или объектов ядра операционной системы, или иных необходимых ресурсов.

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.FreeVM
Set MiniMono = Nothing
```

После вызова функции CreateVM значения свойств более не используются.

`FreeVM`

Функция FreeVM завершает работу виртуальной машины MiniMono. Возвращаемое значение отсутствует. Если виртуальная машина не была создана, то функция ничего не выполняет.

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.FreeVM
Set MiniMono = Nothing
```

EvalStr

Функция EvalStr вычисляет значение аргумента как выражения языка MUMPS в диалекте MiniMono. Аргумент - строка (тип BSTR), возвращаемое значение - строка (тип BSTR) с результатом вычисления.

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

WScript.Echo MiniMono.EvalStr( "$zv")

MiniMono.FreeVM
Set MiniMono = Nothing
```

Eval

Функция Eval вычисляет значение аргумента как выражения языка MUMPS в диалекте MiniMono. Аргумент - объект типа MiniMono.ServerString, возвращаемое значение объект типа MiniMono.ServerString.

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

Dim Tmp
Set Tmp = WScript.CreateObject( "MiniMono.ServerString")
Tmp.Value = "$zv"

Dim Res
Set Res = MiniMono.Eval( Tmp)

WScript.Echo Res.Value

MiniMono.FreeVM
Set MiniMono = Nothing
```

```
GetError
```

Функция GetError возвращает значение последней ошибки, тип возвращаемого значения MiniMono.ServerString.

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

WScript.Echo "Undefined: " & MiniMono.EvalStr( "unknown")

Dim ServerString
```

```
Set ServerString = MiniMono.GetError

WScript.Echo ServerString.Value

MiniMono.FreeVM
Set MiniMono = Nothing
```

GetErrorStr

Функция GetErrorStr возвращает значение последней ошибки, тип возвращаемого значения строка (BSTR).

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

WScript.Echo "Undefined: " & MiniMono.EvalStr( "unknown")

WScript.Echo MiniMono.GetErrorStr()

MiniMono.FreeVM
Set MiniMono = Nothing
```

Execute

Функция Execute выполняет строку команд заданную аргументом типа MiniMono.ServerString. Возвращаемое значение целое число с кодом ошибки:

0	Функция завершена успешно
1	Аргумент не может быть использован из-за синтаксической ошибки
5	Сбой базы данных или процесса
6	Вызванные действия привели к выполнению команды HALT

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

Dim Commands
Set Commands = WScript.CreateObject( "MiniMono.ServerString")

Commands.Value = "s list=$lb(123,456,$h)"

MiniMono.Execute Commands

Dim Tmp
Set Tmp = WScript.CreateObject( "MiniMono.ServerString")
Tmp.Value = "list"

Dim List
Set List = MiniMono.Eval( Tmp)

Dim Element
Set Element = WScript.CreateObject( "MiniMono.ServerString")

MiniMono.ListGet List, 1, Element
WScript.Echo "Before: " & Element.Value

Element.Value = "next"

MiniMono.ListSet List, 1, Element

MiniMono.ListGet List, 1, Element
WScript.Echo "After: " & Element.Value

MiniMono.FreeVM
Set MiniMono = Nothing
```

`ExecuteStr`

Функция ExecuteStr выполняет строку команд заданную аргументом типа строка (BSTR). Возвращаемое значение целое число с кодом ошибки:

0	Функция завершена успешно
1	Аргумент не может быть использован из-за синтаксической ошибки
5	Сбой базы данных или процесса
6	Вызванные действия привели к выполнению команды HALT

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s list=$lb(123,456,$h)"

Dim Tmp
Set Tmp = WScript.CreateObject( "MiniMono.ServerString")
Tmp.Value = "list"

Dim List
Set List = MiniMono.Eval( Tmp)

Dim Element
Set Element = WScript.CreateObject( "MiniMono.ServerString")

MiniMono.ListGet List, 1, Element
WScript.Echo "Before: " & Element.Value

Element.Value = "next"
```

```
MiniMono.ListSet List, 1, Element

MiniMono.ListGet List, 1, Element
WScript.Echo "After: " & Element.Value

MiniMono.FreeVM
Set MiniMono = Nothing
```

SetTest

Функция SetTest устанавливает значение системной перемнной $TEST. Аргумент типа целое число, если указано значение 0, то системная переменная $TEST устанавливается в значение 0, иначе в значение 1.

Возвращаемое значение - нет.

Функция используется в контексте обратного вызова, если необходимо по соглашениям языка изменить значение переменной $TEST, например если при чтении с указанным таймаутом истекло время ожидания то обработчик события чтения должен изменить значение системной переменной $TEST.

Пример:

```
MiniMono.SetTest 1
```

SetCtrlBreak

Функция SetCtrlBreak устанавливает внутреннее состояние виртуальной машины в контекст прерывания ввода. Виртуальная машина MiniMono при исполнении проверяет индикатор прерывания ввода и генерирует ошибку INTERRUPT.

Аргумент - целое число, если указано значение 0, то состояние прерывания не взводится, иначе взводится.

Функция SetCtrlBreak используется в обработчиках ввода-вывода или вызывается из обработчика нажатия Ctrl+Break или при ином обнаружении во входной последовательности данных указывающих на необходимость такого прерывания, например, получение символа с кодом 3 от телнет-клиента.

Пример:

```
MiniMono.SetCtrlBreak 1
```

ListGet

Функция ListGet возвращает один элемент списковой структуры.

Первый аргумент, типа MiniMono.ServerString, задает значение списка, второй аргумент типа целое число задает номер позиции в списке, третий аргумент задает объект типа MiniMono.ServerString, в котором должен быть возвращен элемент списка.

Возвращаемое значение - нет.

Для неопределенного значения списка возвращается значение пустая строка.

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s list=$lb(123,456,$h)"

Dim Tmp
Set Tmp = WScript.CreateObject( "MiniMono.ServerString")
Tmp.Value = "list"

Dim List
Set List = MiniMono.Eval( Tmp)

Dim Element
Set Element = WScript.CreateObject( "MiniMono.ServerString")

MiniMono.ListGet List, 1, Element
WScript.Echo Element.Value
```

```
MiniMono.ListGet List, 2, Element
WScript.Echo Element.Value

MiniMono.ListGet List, 3, Element
WScript.Echo Element.Value

MiniMono.FreeVM
Set MiniMono = Nothing
```

ListSet

Функция ListSet устанавливает в списке элемент в указанное значение.

Первый аргумент, тип MiniMono.ServerString, задает список, в котором необходимо изменить значение. Второй аргумент, целое число, задает номер позиции элемента. Третий аргумент, тип MiniMono.ServerString, задает значение элемента списка.

Возвращаемое значение - нет.

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s list=$lb(123,456,$h)"

Dim Tmp
Set Tmp = WScript.CreateObject( "MiniMono.ServerString")
Tmp.Value = "list"

Dim List
Set List = MiniMono.Eval( Tmp)

Dim Element
```

```
Set Element = WScript.CreateObject( "MiniMono.ServerString")

MiniMono.ListGet List, 1, Element
WScript.Echo "Before: " & Element.Value

Element.Value = "next"

MiniMono.ListSet List, 1, Element

MiniMono.ListGet List, 1, Element
WScript.Echo "After: " & Element.Value

MiniMono.FreeVM
Set MiniMono = Nothing
```

ListLength

Функция ListLength возвращает число элементов списка.

Аргумент типа MiniMono.ServerString задает значение списка.

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s list=$lb(123,456,$h)"

Dim Tmp
Set Tmp = WScript.CreateObject( "MiniMono.ServerString")
Tmp.Value = "list"

Dim List
Set List = MiniMono.Eval( Tmp)
```

```
WScript.Echo MiniMono.ListLength( List)

MiniMono.FreeVM
Set MiniMono = Nothing
```

Text

Функция Text выполняет декорирование строки по правилам языка MUMPS, чтобы результат был синтаксически корректной строкой. Первый аргумент задает исходное значение строки, тип MiniMono.ServerString. Второй аргумент задает объект типа MiniMono.ServerString, в который необходимо вернуть декорированную строку.

Функция Text выполняет преобразование аналогично функции $ZQUOTE для образования синтаксически корректных аргументов для строк, которые могут содержать непечатные символы. Функция представляет непечатные символы используя операторы конкатенации и функцию $CHAR, а также корректно удваивая при необходимости двойные кавычки.

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s list=$lb(123,456,$h)"

Dim Tmp
Set Tmp = WScript.CreateObject( "MiniMono.ServerString")
Tmp.Value = "list"

Dim Res
Set Res = MiniMono.Eval( Tmp)
```

```
WScript.Echo Res.Value

Dim Str
Set Str = WScript.CreateObject( "MiniMono.ServerString")

MiniMono.Text Res, Str

WScript.Echo Str.Value

MiniMono.FreeVM
Set MiniMono = Nothing
```

TextStr

Функция TextStr выполняет декорирование строки по правилам языка MUMPS, чтобы результат был синтаксически корректной строкой. Аргумент задает исходное значение строки, тип строка (BSTR). Возвращаемое значение тип строка (BSTR), содержащая декорированное представление строки. Функция представляет непечатные символы используя операторы конкатенации и функцию $CHAR, а также корректно удваивая при необходимости двойные кавычки.

Пример:

```
Dim MiniMono
Set MiniMono = WScript.CreateObject( "MiniMono.VM")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

WScript.Echo MiniMono.TextStr( "str""str")

MiniMono.FreeVM
Set MiniMono = Nothing
```

6.8.3 События объекта MiniMono.VM

События объекта MiniMono.VM вызываются из контекта виртуальной машины MiniMono при наступлении события или при запросе определенного программой поведения для устройства по умолчанию. События могут быть установлены независимо друг от друга, и если обработчик события не установлен, то виртуальная машина MiniMono использует вместо него поведение по умолчанию.

Для возвращаемых значений обработчиков событий используется тип целое число, ноль при успехе работы обработчика или иное число при ошибке. Если среда исполнения не поддерживает возвращаемое значение обработчика события, то виртуальная машина считает что обработчик завершился успешно.

```
DevUse(PairsCount,ParamPairs)
```

Обработчик события DevUse вызывается при вызове команды USE для устройства по умолчанию.

Первый аргумент события целое число, указывает число пар ключ / значение указанных в опциях команды USE. Второй аргумент - массив значений в последовательности ключ, за ним значение параметра. Массив передается как VARIANT типа массив, элементы массива - объекты типа VARIANT. Если имя опции или значение опции пропущены, то передается пустая строка.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

Dim Val
Dim Expr
```

```
Set Expr =
  WScript.CreateObject( "MiniMono.ServerString")
Expr.Value = "$zv"

Set Val = MiniMono.Eval( Expr)

WScript.Echo Val.Value

MiniMono.ExecuteStr "u $p:(/P1=123:/P2=456)"

MiniMono.FreeVM

Set Expr = Nothing
Set MiniMono = Nothing

Sub MiniMono_DevUse( PairsCount, Pairs)
  For Counter = 0 to PairsCount - 1
    ParamName = Pairs( Counter * 2)
    ParamValue = Pairs( Counter * 2 + 1)
    WScript.Echo
      "DevUse, Parameter = " & ParamName &
      " Value = " & ParamValue
  Next
End Sub


DevWriteStr(Value)
```

Обработчик события DevWriteStr вызывается при выполнении команды записи строки в устройство по умолчанию.

Аргумент события Value имеет тип MiniMono.ServerString и содержит данные строки которую необходимо вывести в текущее устройство. Характер и способ вывода данных определяется программистом.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")
```

```
MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "w 123"

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevWriteStr( Str)
  WScript.Echo
    "Event DevWriteStr fired, str = " &
      str.Value
End Sub


DevWriteChar(Value)
```

Обработчик события DevWriteChar вызывается при выполнении команды вывода в текущее устройство кода символа.

Аргумент события Value имеет тип целое число и позволяет передавать как положительные, так и отрицательные коды. Характер и способ вывода данных определяется программистом. В большинстве случаев вывод кода символа означает вывод символа с этим кодом, но в некоторых случаях разработчики используют отдельные соглашения о выводе отрицательных значений, которые для определенного типа устройства могут означать дополнительные действия, заменяя таким образом отдельную команду USE.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
```

```
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "w *122,*123"

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevWriteChar( Symbol)
  WScript.Echo
    "Event DevWriteChar fired, Symbol = " & Symbol
End Sub


DevWriteNL()
```

Обработчик события DevWriteNL вызывается при выполнении команды форматного вывода перевода строки. Аргументов не имеет.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "w !!"

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevWriteNL
  WScript.Echo "Event DevWriteNL fired"
End Sub
```

```
DevWriteFF()
```

Обработчик события DevWriteFF вызывается при выполнении команды форматного вывода перевода формата (новая страница). Аргументов не имеет.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "w ##"

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevWriteFF
  WScript.Echo "Event DevWriteFF fired"
End Sub
```

```
DevWriteTAB(TabCount)
```

Обработчик события DevWriteTAB вызывается при выполнении команды форматного вывода табуляции. Аргумент TabCount имеет тип целое число и значение указанное в опции табуляции.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")
```

```
MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "w ?5,?12"

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevWriteTAB( TabCount)
  WScript.Echo
    "Event DevWriteTAB fired, TabCount = " &
      TabCount
End Sub


DevReadStr(Len,TimeOut,ReadString)
```

Обработчик события DevReadStr вызывается при выполнении команды чтения (READ) из устройства по умолчанию.

Аргумент Len имеет тип целое число и указывает сколько байт было указано для чтения в аргументах команды. Если число байт не было указано, то передается значение -1. Максимальное число байт которые могут быть прочитаны, ограничивается общей длиной строки в MiniM и составляет 32 килобайт.

Аргумент TimeOut имеет тип целое число и указывает число миллисекунд времени ожидания. В команде чтения READ указывается число секунд и может быть использовано дробное число, но виртуальная машина MiniMono производит пересчет этого значения в миллисекунды. Если время ожидания в команде чтения не было указано, то передается значение -1.

Аргумент ReadString имеет тип MiniMono.ServerString и в его значение Value необходимо поместить результат выполнения команды READ. Эта последовательность байт будет использована в контексте виртульной машины MiniMono как результат выполнения команды.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "r str"

WScript.Echo "str after read is " &
  MiniMono.EvalStr( "str")

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevReadStr( Length, TimeOut, Str)
  WScript.Echo "Event DevReadStr fired"
  Str.Value = "answer"
End Sub


DevReadChar(TimeOut,ReadChar)
```

Обработчик события DevReadChar вызывается при выполнении команды чтения кода символа из устройства по умолчанию.

Аргумент TimeOut имеет тип целое число и указывает число миллисекунд времени ожидания. В команде чтения READ указывается число секунд и может быть использовано дробное число, но виртуальная машина MiniMono производит пересчет этого значения в миллисекунды. Если время ожидания в команде чтения не было указано, то передается значение -1.

Аргумент ReadChar имеет тип VARIANT. В этот параметр должно быть присвоено значение кода прочитанного символа. Коды символов рассматриваются как целые числа от 0 до 255 включительно.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "r *ch"

WScript.Echo
  "ch after read is " & MiniMono.EvalStr( "ch")

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevReadChar( TimeOut, ReadCode)
  WScript.Echo "Event DevReadChar fired"
  ReadCode = 123
End Sub


DevGetX(Value)
DevGetY(Value)
```

Обработчики событий DevGetX и DevGetY вызываются при выполнении чтения системных переменных $X и $Y для устройства по умолчанию.

Аргумент Value имеет тип VARIANT и обработчик должен присвоить этому аргументу значение соответственно $X или $Y.

Трактовка как положительных, так и отрицательных значений задается разработчиком устройства.

Пример:

```
Dim MiniMono
Set MiniMono =
```

```
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s x=$x,y=$y"

WScript.Echo "values are: x = " &
  MiniMono.EvalStr( "x") &
  ", y = " & MiniMono.EvalStr( "y")

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevGetX( XValue)
  WScript.Echo "Event DevGetX fired"
  XValue = 123
End Sub

Sub MiniMono_DevGetY( YValue)
  WScript.Echo "Event DevGetY fired"
  YValue = 456
End Sub


DevSetX(Value)
DevSetY(Value)
```

Обработчики событий DevSetX и DevSetY вызываются при выполнении присваивания системных переменных $X и $Y для устройства по умолчанию.

Аргумент Value имеет тип целое число и передается то значение которое было указано в команде присваивания. В случае если присваиваемое значение было не целым числом, оно приводится к целому по правилам языка MUMPS.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s $x=123,$y=456"

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevSetX( XValue)
  WScript.Echo
    "Event DevSetX fired, XValue = " & XValue
End Sub

Sub MiniMono_DevSetY( YValue)
  WScript.Echo
    "Event DevSetY fired, YValue = " & YValue
End Sub


DevGetKEY(KeyValue)
```

Обработчик события DevGetKEY вызывается при чтении значения системной переменной $KEY при текущем устройстве ввода-вывода по умолчанию.

Аргумент KeyValue имеет тип MiniMono.ServerString, в его значение Value необходимо записать последовательность байт, которая соответствует значению $KEY для устройства по умолчанию. Разработчик устройства самостоятельно определяет, какие значения в нем должны содержаться. Традиционно в значении $KEY отражается чем завершилась операция чтения, например каким из разделителей чтения.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s key=$key"

WScript.Echo "key have value " &
  MiniMono.EvalStr( "key")

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevGetKEY( Key)
  WScript.Echo "Event DevGetKEY fired"
  Key.Value = "abcd"
End Sub



DevSetKEY(KeyValue)
```

Обработчик события DevSetKEY вызывается когда в контексте виртуальной машины MiniMono производится присваивание значения системной переменной $KEY для устройства по умолчанию.

Аргумент KeyValue имеет тип MiniMono.ServerString и его свойство Value передает новое значение, присваиваемое системной переменной $KEY.

Как именно необходимо трактовать присваивание системной переменной $KEY для устройства по умолчанию - определяет разработчик этого устройства.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s $key=123"

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevSetKEY( Key)
  WScript.Echo
    "Event DevSetKEY fired: " & Key.Value
End Sub
```

```
DevZEOF(Value)
```

Обработчик события DevZEOF вызывается при чтении системной переменной $ZEOF для устройства по умолчанию.

Аргумент Value имеет тип VARIANT и в его значение обработчик должен записать значение 0 если не достигнуто окончание чтения текущего устройства или не ноль если достигнуто.

Поскольку внешние обработчики событий не могут быть в точности интегрированы в виртуальную машину MiniMono в части генерации встроенной стандартной ошибки достижения конца чтения, разработчикам рекомендуется использовать системную переменную $ZEOF, если разрабатываемое устройство по смыслу имеет состояние окончания чтения.

Пример:

```
Dim MiniMono
Set MiniMono =
```

```
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s zeof=$zeof"

WScript.Echo "zeof value is: " &
  MiniMono.EvalStr( "zeof")

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevZEOF( ZEOF)
  WScript.Echo "Event DevZEOF fired"
  ZEOF = 1
End Sub



DevGetZA(KeyValue)
DevGetZB(KeyValue)
```

Обработчики событий DevGetZA и DevGetZB вызываются при чтении значений системных переменных $ZA и $ZB для устройства по умолчанию.

Аргумент KeyValue имеет тип MiniMono.ServerString и в его свойтво Value необходимо поместить последовательность байт, соответствующую значениям соответственно системных переменных $ZA и $ZB.

Значения системных переменных $ZA и $ZB полностью определяются разработчиком устройства. Традиционно они используются либо для совместимости с другими MUMPS системами либо для получения специфической для устройств аинформации или его состояния.

Пример:

```
Dim MiniMono
Set MiniMono =
  WScript.CreateObject( "MiniMono.VM", "MiniMono_")

MiniMono.DataFile = "empty.dat"
MiniMono.DBCacheSize = 100
MiniMono.JournalingEnabled = 0

MiniMono.CreateVM

MiniMono.ExecuteStr "s za=$za,zb=$zb"

WScript.Echo "za value is: " &
  MiniMono.EvalStr( "za")
WScript.Echo "zb value is: " &
  MiniMono.EvalStr( "zb")

MiniMono.FreeVM

Set MiniMono = Nothing

Sub MiniMono_DevGetZA( ZA)
  WScript.Echo "Event DevGetZA fired"
  ZA.Value = 123
End Sub

Sub MiniMono_DevGetZB( ZB)
  WScript.Echo "Event DevGetZB fired"
  ZB.Value = 456
End Sub
```

6.8.4 Свойства объекта MiniMono.ServerString

`Value`

Value является свойством объекта MiniMono.ServerString, имеет тип строка. Свойство доступно на чтение и запись.

`Length`

Length является свойством типа число и определяет длину данных в объекте MiniMono.ServerString. Свойство доступно на чтение и запись. При присваивании значения меньше 0 значение устанавливается в 0, при присваивании больше допустимого (32767) длина устанавливается в максимально допустимое значение.

6.8.5 Функции объекта MiniMono.ServerString

`GetAt( Pos)`

Функция возвращает целое число, код символа в позиции Pos, где позиция Pos задается целым числом. Если Pos указывает за пределы данных, то возвращается значение -1. Коды символов возвращаются как числа от 0 до 255.

`SetAt( Pos, Code)`

Функция записывает символ с кодом Code в позиции Pos. Значения Code и Pos задаются целыми числами. Если значение Pos указывает за пределы данных, то никаких изменений не выполняется.

`Add( Val )`

Функция конкатенирует к текущему значению объекта значение объекта Val типа MiniMono.ServerString. Значение объекта Val не изменяется. При успехе функция возвращает значение 1, иначе 0.

`AddStr( Val)`

Функция конкатенирует к текущему значению объекта строку Val. Значение строки Val не изменяется. При успехе функция возвращает значение 1, иначе 0.

Глава 7

MiniMono для Android

7.1 Состав SDK

MiniMono SDK для Android представляет собой набор файлов, используемых для разработки приложения для Android с включением в приложение MiniM Embedded Edition.

MiniMono SDK не имеет инсталлятора и деинсталлятора, не содержит утилит, запускаемых на компьютере разработчика и не зависит от операционной системы разработки. Для использования MiniMono SDK для построения приложений для Android он используется совместно с Android SDK. Если разработчику необходимо редактировать специфические для MiniMono файлы, то необходимо использовать инсталлятор MiniMono как средства разработки, в частности для создания баз данных с определенным составом рутин и глобалов, для редактирования файлов определения символов, для тестирования MUMPS кода. Также может быть использован полный MiniM Database Server как средство разработки программных элементов, специфических для MiniMono.

MiniMono SDK распространяется в виде архива ZIP, после распаковки которого в подкаталоге minimono создаются подкаталоги:

db	Файлы баз данных MiniM, по умолчанию содержит по крайней мере один файл empty.dat, пустой файл данных
doc	Документация на MiniM в формате pdf
examples	Примеры для Android на языке Java для использования совместно с Android SDK

include	Заголовочные файлы для самостоятельной интеграции в приложения на основе языков C и Pascal (FreePascal + Lazarus)
libs	Скомпилированные библиотеки libminimono.so + libminimonoj.so + libzini.so для различных поддерживаемых архитектур
routines	Стандартные системные рутины MiniM
src	Модули minimonoj и utils на языке Java для разработки приложений с использованием Android SDK
zdevice	Примеры разработки внешних модулей ZDEVICE на языках C и Pascal
zdll	Примеры разработки внешних модулей ZDLL на языках C и Pascal

В настоящее время MiniMono SDK поддерживает следующие процессорные архитектуры для Android:

armeabi	Семейство процессоров архитектуры ARM, 32-битные.
x86	Семейство процессоров архитектуры Intel/AMD x86-32, от i386 и старше, 32-битные.
arm64-v8a	Семейство процессоров архитектуры ARM, 64-битные.
x86_64	Семейство процессоров архитектуры Intel/AMD x86-64, 64-битные.

В настоящее время распространенность устройств на базе Android такова, что большинство устройств используют архитектуру arm, менее распространены устройства на архитектуре x86 и очень редки 64-битные устройства на ARM и x86-64. В большинстве случаев для корректной разработки вполне достаточно включать в разрабатываемое приложение лишь библиотеки для архитектур armeabi и x86.

В настоящее время MiniMono SDK не поддерживает архитектуры MIPS и MIPS64, хотя их поддерживает Android SDK, поскольку эти процессоры используют очередность байт big-endian, а на такие архитектуры процессоров код MiniM не был портирован. К процессорам big-endian также относятся такие процессоры, как Itanium, SPARC и некоторые другие.

Файл libminimono.so содержит модуль MiniMono и является основным исполняемым. По внутреннему устройству и набору функций он в точности такой же как и модули MiniMono для других операционных систем. Этот файл необходим для работы MiniMono для Android.

Файл libminimonoj.so представляет собой переходник с Java кода на код MiniMono и является JNI модулем. Этот файл необходим при использовании MiniMono из среды выполнения Java. В примерах для MiniMono SDK приводятся примеры на языке Java и они используют этот модуль. Если разработчики используют иные средства разработки (на языке C/C++ или Pascal или иные), то этот файл не требуется.

Файл libzini.so представляет собой ZDLL модуль с расширенными функциями для рутины %INI. Рутина входит в стандартный набор системных рутин MiniM, но в базе данных empty.dat по умолчанию ее нет, это пустая база. Если разработчики не используют рутину %INI в своих приложениях для Android, то файл libzini.so не требуется.

7.2 Построение приложения

Применение MiniMono для Android иллюстрируется с использованием стандартных средств Android SDK и основным руководством по разработке для Android можно считать ресурс:

```
http://developer.android.com
```

На этом ресурсе можно найти все необходимые утилиты, информационные материалы по разработке для Android, правила сборки проектов и правила их распространения.

Построение приложения иллюстрируется на приложении основанном на языке разработки Java. В случае применения иных средств разработки нужно использовать только модули libminimono.so из состава MiniMono SDK для соответствующих процессоров.

Для создания приложения необходимо, чтобы каталог tools из каталога установки Android SDK находился в путях поиска (в значении переменной окружения PATH). Используется утилита android с параметрами создания проекта:

```
android create project
```

Можно рекомендовать создать командный файл, которые эти операции автоматизирует, например для создания заготовки проекта для примера Example1 (вариант для Windows x64):

```
set PATH=C:\Program Files (x86)\Android\
   android-sdk\tools;%PATH%
md example1
android create project --target 1
  --name example1 --path example1
  --activity Example1
  --package android.database.minimono
```

Здесь создается подкаталог example1 и в нем создается набор каталогов по умолчанию для приложения example1, в котором Activity Example1 входящее в пакет

```
android.database.minimono
```

После создания заготовки необходимо добавить в проект файлы библиотек libminimono.so и libminimonoj.so из соответствующих подкаталогов libs из состава MiniMono SDK в подкаталог проекта libs таким образом, чтобы получилась следующая структура каталогов:

```
/example1
   /libs
      /armeabi
         libminimono.so
         libminimonoj.so
      /x86
         libminimono.so
         libminimonoj.so
```

Файлы so из состава MiniMono SDK используются в подкаталогах /libs совместно с остальными файлами so, используемыми приложением. В этих же подкаталогах нужно размещать ZDLL и ZDEVICE модули, если они входят в разрабатываемое приложение.

При сборке инсталлятора приложения (файл apk) эти библиотеки будут внесены в инсталлятор и при использовании на устройстве под

управлением Android будут использоваться компилированные модули соответствующие процессору устройства.

В подкаталоге src проекта необходимо разместить подкаталог android, в нем подкаталог database, в нем подкаталог minimono, в нем файлы minimonoj.java и utils.java таким образом:

```
/src
   /android
      /database
         /minimono
            minimonoj.java
            utils.java
```

Такая структура каталогов определяется соглашениями языка Java. Там же необходимо разместить файл примера ExampleXXX.

Далее в файле /res/layout/main.xml нужно задать структуру интерфейса приложения. Можно просто заменить этот файл из файла main.xml из состава MiniMono SDK.

Для работы СУБД MiniMono необходимо добавить в приложение файл данных. В состав MiniMono SDK в подкаталог /db входит пустой файл данных empty.dat. Примеры используют перенос этого файла через подкаталог /assets/db. Файлы, размещенные в подкаталоге /assets, входят в сборку apk как есть, с сохранением структуры каталогов, и доступны через специальный интерфейс AssetManager как псевдофайлы. Утилиты из файла

```
/android/database/minimono/utils.java
```

используются примерами для синхронизации файлов из хранилища Assets приложения для Android в локальные файлы операционной системы, с которыми уже может работать СУБД MiniMono.

После подготовки каталогов проект готов к сборке. Для сборки проекта разработчики Android рекомендуют использовать билд-систему ANT. При ее использовании необходимо чтобы каталоги bin системы ANT и JDK находились в путях поиска (переменная окружения PATH). Можно рекомендовать создать командный файл, который автоматизирует эти операции (вариант для Windows):

```
set ANT_HOME=c:\tools\apache-ant
set PATH=%ANT_HOME%\bin;%PATH%
set JAVA_HOME=C:\Program Files\Java\jdk1.8.0_20
set PATH=%JAVA_HOME%\bin;%PATH%
cd example1
ant debug
```

В случае инсталляции JDK других версий или использования других каталогов необходимо указать их правильные значения.

После сборки проекта в подкаталоге /bin генерируется файл apk, например example1-debug.apk.

Для получения релизной сборки необходимо использовать утилиты и правила от Google для подписания сборки.

Для размещения и инсталляции на устройстве используется утилита adb с параметром

```
adb install
```

а для переустановки с параметрами

```
adb install -r
```

с указанием на какое устройство устанавливать и какой apk файл использовать.

Все примеры для MiniMono SDK используют одинаковый простой интерфейс, приведены для языка Java и для всех процедура создания и сборки приложения одинакова.

Если необходима поддержка не только процессорных архитектур arm и x86, но и 64-битных, то также нужно добавить библиотеки из 64-битных подкаталогов arm64-v8a и x86_64.

Примеры не используют настраиваемые файлы определения сравнения символов nat, все примеры используют сравнение символов по умолчанию. При использовании определения символов отличного от определения по умолчанию файл такого определения также необходимо перенести с приложением через хранилище /assets, например в подкаталоге /nat и после синхронизации указать реальное имя файла в начальной инициализации виртуальной машины MiniMono.

7.3 Утилита синхронизации Assets

Assets - это добавляемое к приложению для Android структурированное хранилище файлов, содержание которых не используются сборщиком проекта, они не подвергаются никакой трансформации и переносятся в составе инсталляционного файла apk как есть, в сыром виде. Это хранилище используется и рекомендуется использовать для переноса дополнительных файлов приложения.

В случае использования в приложении модуля MiniMono необходимо по меньшей мере перенести файл данных. Опционально, на усмотрение разработчиком, могут понадобиться файлы определения символов /nat, исходные тексты рутин, иные файлы данных.

Вместо пустой базы данных, как это сделано в примерах, могут переноситься заранее подготовленные разработчиком базы данных MiniMono, с необходимым набором рутин и начальным состоянием глобалов.

При переносе сырых файлов через хранилище assets инсталлятором apk необходимо иметь в виду, что разработчики Android могут самостоятельно поддерживать ограничение на размер одного файла, входящего в assets. Это ограничение может накладываться разработчиками Google. В случае обнаружения, что приложение превышает такое ограничение, необходимо принять меры к разделению информации между несколькими файлами или использовать дополнительный источник данных для приложения.

Файлы, переносимые через хранилище Assets, доступны приложению как псевдофайлы, через объект AssetManager. Для того, чтобы разместить файлы в виде обычных файлов, пригодных для использования СУБД MiniMono, их надо скопировать в доступный приложению локальный подкаталог по правилам безопасности Android. Это выполняет утилитный класс utils.

Утилиты синхронизации assets читают список файлов, входящих в заданный подкаталог /assets, и проверяют существует ли соответствующий файл в локальной файловой системе. Если существует, то этот файл не изменяется. Если еще не существует, то он копируется из хранилища assets в локальную файловую систему.

Разработчики приложения для Android могут использовать свои собственные утилиты синхронизации и правила разрабатываемого приложения.

Для использования утилит синхронизации assets необходимо файл utils.java разместить в подкаталоге приложения перед его сборкой.

Функция объекта класса utils

```
public void syncAssetDir( String dir,
  Context context, AssetManager assets)
```

выполняет синхронизацию всех файлов входящих в указанный подкаталог хранилища assets.

Функция объекта класса utils

```
public String fileName( String dir,
  Context context, String name)
```

возвращает реально используемое имя файла в локальной файловой системе, соответствующее указанному файлу из указанного подкаталога assets.

7.4 Примеры

Примеры для MiniMono для Android выполнены как приложения на языке Java использующие Java объект для JNI модуля (libminimonoj.so), обращающегося к модулю MiniMono (libminimono.so):

```
import android.database.minimono.minimonoj;
import android.database.minimono.utils;

...

minimonoj MiniMono =
  new minimonoj( CodePage);
minimonoj.MiniMonoVM init =
  new minimonoj.MiniMonoVM( CodePage);
int ret = MiniMono.CreateMiniMono( init);
```

Экран состоит из двух элементов - текст и кнопка. В элемент отображения текста выводится информация о работе приложения, обработчик нажатия кнопки завершает работу приложения.

В примерах используется одинаковая инициализация виртуальной машины MiniMono:

```
MiniMono.GetDefaultSettings( init);
init.DataFile =
  u.fileName( dbDir,
  getApplicationContext(), dbFileName);
init.JournalingEnabled = 0;
```

Предварительно выполняется синхронизация файла базы данных из хранилища assets:

```
utils u = new utils();
String dbDir = "db";
String dbFileName = "empty.dat";
u.syncAssetDir( dbDir,
  getApplicationContext(), getAssets());
```

И для виртуальной машины MiniMono используется то имя файла, которое получено по правилам синхронизации:

```
init.DataFile =
  u.fileName( dbDir,
  getApplicationContext(), dbFileName);
```

Далее в примерах выполняется последовательность обращений к виртуальной машине MiniMono, после чего завершается ее работа.

```
MiniMono.FreeMiniMono();
```

Работа самого приложения завершается по нажатию кнопки:

```
final Button btnClose =
 (Button)findViewById(R.id.btnClose);
btnClose.setOnClickListener(
  new View.OnClickListener() {
    public void onClick(View v) {
      finish();
    }
});
```

Пример Example1.java показывает вычисление выражения на языке MUMPS и чтение результата. Результат выводится в текстовый элемент на экран.

```
txtOut.append( "Value of $zversion: " +
  MiniMono.Read( "$zversion") + "\n");
txtOut.append( "Now is: " +
  MiniMono.Read( "$zdate($h,3)") + "\n");
```

Пример Example2.java выполняет последовательность команд, которой присваиваются значения локальным переменным, после чего выполняется чтение этих локальных переменных:

```
int n = 5;
MiniMono.Execute( "f i=1:1:" +
  n + " s a(i)=i*i");

for( int i = 1; i <= n; i++)
{
  txtOut.append(
    "Value of a(" + i + ") is: " +
    MiniMono.Read( "a(" + i + ")") +
    "\n");
}
```

Пример Example3.java показывает работу с переменными списковой структуры. Пример генерирует в виртуальной машине значение списковой структуры, после чего читает эту переменную и обращается к функциям определения длины и получения каждого из элементов списка:

```
MiniMono.Execute(
  "s var=$lb($zv,$sy,$zd($h,8))");
String list = MiniMono.Read( "var");

int n = MiniMono.ListLength( list);
txtOut.append(
  "Actual length of the list is: " +
  n + "\n");

for( int i = 1; i <= n; i ++)
```

```
{
  txtOut.append( "Item " + i + " : " +
    MiniMono.ListGet( list, i) + "\n");
}
```

Далее пример показывает как на стороне Java производится изменение элементов списковой структуры:

```
list = "";
// set 1 list item
list = MiniMono.ListSet(
  list, 1, "123456");
// set 2 list item
list = MiniMono.ListSet(
  list, 2, "123.456");
// set 3 list item
list = MiniMono.ListSet(
  list, 3, "Hello");
```

По правилам языка Java нельзя изменить строку, передавая ее по ссылке или указателю, поэтому новое значение списка возвращается функциями изменения списковой структуры и переменная полностью переприсваивается.

Далее пример показывает применение функции декорирования строки по правилам языка MUMPS. Возвращаемая функцией MiniMono.Text строка декорируется так, что ее можно использовать в выражениях по правилам языка MUMPS. Функция преобразует если это необходимо строку в последовательность символов, допустимых в строке, операторов конкатенации, использует функцию $CHAR() и там где это необходимо добавляет двойные кавычки.

Пример Example4.java показывает как на языке Java задать поведение устройства ввода-вывода по умолчанию (principal device). Для этого нужно наследовать класс minimonoj и переопределить виртуальные функции для устройства - вывод строки, кода одного символа, перевода строки и другие, которые необходимы в разрабатываемом приложении.

```
class minimonodev4 extends minimonoj
{
  private TextView txtOut;
```

```
  minimonodev4( String _CodePage,
    TextView _txtOut)
   throws java.lang.Exception
  {
    super( _CodePage);
    txtOut = _txtOut;
  };

  // overload functions for device handlers
  public int DevWriteStr( String str)
  {
    txtOut.append( "Fired WriteStr(\"" +
      str + "\") event\n");
    return 0;
  };
  public int DevWriteChar( int _Char)
  {
    txtOut.append( "Fired WriteChar(" +
      _Char + ") event\n");
    return 0;
  };
  public int DevWriteNL()
  {
    txtOut.append( "Fired WriteNL event\n");
    return 0;
  };
  ...
};
```

Далее именно этот класс с переопределенными обработчиками используется в приложении:

```
minimonodev4 MiniMono =
  new minimonodev4( CodePage, txtOut);
```

Далее в примере выполняются команды вывода в устройство по умолчанию:

```
MiniMono.Execute(
  "write \"string\",*46,?12,!,#");
```

Пример Example5 показывает обращение к индексированным переменным на примере чтения и записи глобала. Предварительно выполняется заполнение значениями:

```
MiniMono.Execute(
  "f i=0:1:" + limit + " s ^var(i,i*i)=i*i*i");
```

Для задания глобала и его индексов используется задание имени и массива индексов:

```
// read global values using direct call
String[] Indices = new String[ 2];

for( int i = 0; i < limit; i++)
{
  Indices[ 0] = Integer.toString( i);
  Indices[ 1] = Integer.toString( i * i);
  txtOut.append(
    MiniMono.ReadGlobal( "var", Indices) + "\n");
}
```

Для задания неиндексированного имени глобала используется пустой массив индексов:

```
MiniMono.KillGlobal( "var", null);
```

Этот же способ используется для множества функций с индекированными переменными - ReadLocal, WriteLocal, KillLocal, OrderLocal, и других, а также для раздельной передачи параметров функций и подпрограмм UserFunc и UserDo.

Объект массив строк в языке Java самостоятельно несет в себе информацию о количестве входящих в него элементов и задает их упорядочение. В случае отсутствия индексов или параметров функций и подпрограмм передается значение null.

Пример Example6 показывает запись в базу данных исходного текста рутины

```
// kill routine before create new
String global_name = "ROUTINE";
String routine_name = "Example6";
String[] indices = new String[ 1];
indices[ 0] = routine_name;
MiniMono.KillGlobal( global_name, indices);

// create routine as a series of direct global sets
indices = new String[ 2];
indices[ 0] = routine_name;

String[] routine_text =
{
  "Func(arg)",
  " w \"arg = \",arg,!",
  " n expr=$zd($h,8)",
  " q expr"
};
for( int i = 0; i < routine_text.length; i++)
{
  indices[ 1] = Integer.toString( i + 1);
  MiniMono.WriteGlobal(
    global_name, indices, routine_text[ i]);
}
```

После этого пример выполняет компиляцию исходного текста в байт-код

```
// compile routine
MiniMono.Read(
  "$v(\"rou\",\"c\",\"" + routine_name + "\")");
```

После этого обращается к рутине, выполняя ее код в виртуальной машине MiniMono:

```
// evaluate expression
String func_result =
  MiniMono.Read( "$$Func^" +
    routine_name + "(\"any data\")");
```

Работа MUMPS кода в этом примере показывается как выводом на экран результата вычисления значения функции $$Func, так и выводом на экран результата команд write, которые эта функция вызывает.

Пример Example7 показывает обработку ошибок. Объект JNI интерфейса для MiniMono при происхождении ошибки генерирует исключение. Это исключение перехватывается оператором catch и на экран выводится текущая диагностика ошибки. В примере используется две ошибки - деление на ноль и обращение к неопределенной переменной.

```
try
{
  // execute division by zero
  MiniMono.Execute( "w 1/0");
}
catch( Throwable e)
{
  // catch execution error and display last error
  txtOut.append( "Catched an error: " +
    MiniMono.GetLastError() + "\n");
  // revert value of $ec to an empty
  // string for next code
  MiniMono.Execute( "s $ec=\"\"");
}

try
{
  // read value of undefined variable
  MiniMono.Read( "abcdef");
}
catch( Throwable e)
{
  // catch execution error and display last error
  txtOut.append( "Catched an error: " +
    MiniMono.GetLastError() + "\n");
  // revert value of $ec to an empty
  // string for next code
  MiniMono.Execute( "s $ec=\"\"");
}
```

Здесь оба обработчика ошибок кроме вывода на экран диагностического сообщения дополнительно сбрасывают значение системной пере-

менной $ECODE в значение пустая строка, чтобы следующее обращение к виртуально машине MiniMono происходило в чистом контексте обработки ошибки. Эта очистка значения $ECODE необязательна, но при следующем происхождении ошибки ее значение будет дополняться по правилам языка MUMPS для обработчика ошибок это может оказаться лишней информацией, не относящейся к текущему обращению MiniMono. Значение функции MiniMono.GetLastError это просто значение системной переменной $ZERROR, это значение не накапливается при происхождении последовательности ошибок и отображает диагностическое сообщение только о последней произошедшей ошибке. Разработчики приложения с использованием MiniMono должны самостоятельно определить свои обработчики ошибок и поведение приложения в случае происхождения ошибки.