(редакция от 2013-08-17)

kIT Universal Presets

универсальный конфигуратор,
пакетный шелл

Язык UP

Автор:
Шаповалов Арсений
aka АрсеН Hypocrite

Введение (набросок)

Программа kIT Universal Presets служит для применения изменений в конфигурации любых программ. Программа позволяет вносить комплексные изменения в INI-файлы, в реестр, в XML-файлы. Программа способна производить все мыслимые операции с файлами. Программа также позволяет выполнять различные действия вроде перезапуска программы, посылки клавиатурного сочетания, посылки определенного сообщения и т.п.

Программа написана на языке Object Pascal с использованием библиотеки Key Object Library в среде Delphi 7.

Программа распространяется бесплатно для использования в любой среде (в том числе в коммерческих организациях). Разрешается включать программу в любые пакеты и сборки программ и использовать в любых целях без уведомления автора.

Всю ответственность за применение пресетов несёт лицо, применяющее пресеты, а не авторы пресетов или авторы программы kIT Universal Presets.

Системные требования: Windows 2000/XP/2003/Vista/7.

Аппаратные требования: дополнительно не накладываются.

Спецификация синтаксиса пресетов и сопутствующих конфигурационных файлов и каталогов находится в открытом доступе по адресу:

http://www.kitvision.ru/kitup/specification.

Последняя версия программы доступна по адресу:

http://www.kitvision.ru/kitup/latestversion.

Основу иерархии пресетов составляют категории — группа пресетов, направленных на изменение одной совокупности настроек (формально). Каждая категория хранится в одной папке. Файлы пресетов хранятся в этих папках. Они имеют синтаксис, основанный на синтаксисе INI-файлов, но значительно расширенный.

Пресет может иметь специальную секцию [Configuration] с собственными особыми настройками. Например, есть возможность задать путь к базовой папке (DefaultDirectory), от которой будут отсчитываться относительные пути.

Для удобства кодирования пресетов предусмотрено каскадное автоматическое применение ряда переменных. Например, переменная DefaultDirectory (так называемая базовая папка), использующаяся везде, где применяются относительные пути к файлам. Эта переменная может быть объявлена и в самом пресете, и в конфиге категории, и в конфиге программы. Приоритет, естественно, у значения, объявленного в самом пресете, затем в категории.

Более того, для обеспечения максимальной гибкости можно каскадно складывать пути базовых папок, если они заданы в относительном виде. Если даже на самом верху, в конфиге программы указан относительный путь, то он отсчитывается от папки с программой.

Область применения программы ничем не ограничена. Программа, например, крайне полезна в следующих ситуациях.

Изменение цветовой схемы TC: нужно заменить секцию Colors, а также, возможно, некоторые параметры некоторых плагинов, например, TWinKey.

Изменение языка сборника программ: в большом количестве различных конфигов и в реестре нужно изменять значение обычно одного ключа для каждой программы.

Размеры значков в TC: придется изменить значения нескольких ключей в паре конфигов, а, главное, нужно заменить некоторые bar-файлы.

Определения и понятия

Пресет (Preset)

Это текстовый файл (в любой кодировке), содержащий инструкции для выполнения определённых действий программой kIT Universal Presets. Инструкции записываются на языке UP, описание которого и содержится в последующих главах.

Категория (Category)

Это набор пресетов, обычно объединённых общей направленностью действия (рекомендовано, но не обязательно). Категория определяется папкой, в которой содержатся её пресеты, и конфигурационным файлом в этой папке. Конфигурационный файл содержит настройки, учитывающиеся при применении любого из пресетов данной категории, а также действия, выполняющиеся до или после выполнения любого из пресетов данной категории.

Свободный пресет (Free preset)

Это пресет, не включенный в категорию и даже в общую схему применения пресетов. Таким образом, настройки категории и даже программы игнорируются при применении свободного пресета. Это означает, что внутри такого пресета должно быть достаточно информации для корректного его применения, т. е., например, пути к файлам будут указаны абсолютно.

1) (±) Служебные файлы (в разработке)

Технически программа состоит лишь из одно исполняемого файла, и, как сказано в лицензионном соглашении, конфигурационные файлы и пресеты не являются частью программы. Однако для практического применения потребуется правильно настроить следующие конфигурационные файлы.

1.1) (±) Config.ini в папке программы

Содержит основные настройки самой программы. Во всех случаях, где требуется указать путь и указан относительный путь, он отсчитывается от папки программы.

1.1.1) (±) Секция [Configuration]

1.1.1.1) (+) PresetsDirectory=

Папка с категориями пресетов (которые содержат сами пресеты).

По умолчанию: Presets

1.1.1.2) (+) DefaultDirectory=

Папка по умолчанию при расчете относительных путей.

Параметр может быть переопределён в конфиге категории или в пресете.

По умолчанию: Папка программы

1.1.1.3) (–) LogFile=

Файл для ведения журнала (см. главу 18).

Параметр может быть переопределён в конфиге категории или в пресете.

По умолчанию: kitup.log

1.1.1.4) (–) LogLevel=

Степень подробности ведения журнала (см. главу 18).

Параметр может быть переопределён в конфиге категории или в пресете.

По умолчанию: 3

1.1.1.5) (+) TopMost=

Поверх остальных окон.

По умолчанию: 1

1.1.1.6) (+) AutoClose=

Автоматически закрывать программу после применения пресета.

По умолчанию: 0

1.1.1.7) (–) MessageShowTime=

Закрывать диалоговые окна по истечении указанных секунд. 0 — не показывать вообще. 1 — отключить автоматическое закрытие.

По умолчанию: −1

1.1.1.8) (+) LastCategory=

Хранит имя последней использовавшейся категории. При открытии графического интерфейса эта категория будет развёрнута, и, если задан параметр LastPreset в конфиге этой категории, то этот пресет будет выделен, в

противном случае будет выделен заголовок категории.

По умолчанию: пусто

1.1.1.9) (+) Language=

Определяет язык программы и предпочтительный язык строк в пресетах. Значение задаёт имя файла перевода без расширения ".lng". Например: "Language=Русский".

По умолчанию: Не задано. Если этот параметр не задан, то везде используются строки по умолчанию, даже если имеются альтернативы.

1.1.1.10) (+) LanguagesDirectory=

Определяет папку, где хранятся файлы перевода (с расширением ".lng").

По умолчанию: Languages

1.1.1.11) (+) ShowProgress=

Показывать прогресс применения. При применении пресета через графический интерфейс прогресс показывается в любом случае, однако автоматические закрывается, если ShowProgress=0. Если AutoClose=0 и ShowProgress=1, то пользователю придётся самому закрыть окно прогресса.

По умолчанию: 0

1.1.1.12) (+) ExternalEditor=

Определить редактор файлов Config.ini и up-файлов. Этот редактор можно запустить из контекстного меню на пунктах дерева пресетов.

По умолчанию: notepad.exe

1.1.1.13) (+) CloseProcessWait=

При выполнении команды [p?q|...] ожидать завершения процесса. Время указывается в миллисекундах. Особые значения: 0 — не ждать, –1 — ждать бесконечно.

Реальное время ожидания может быть большим, так как это же значение применяется в SendMessageTimeout при попытке закрыть очередное окно процесса. Никакого принудительного закрытия окон не осуществляется, но по окончании перебора окон процесса снова осуществляется ожидание в течение CloseProcessWait посредством WaitForSingleObject. Дальнейшее поведение зависит от значения TerminateAfterWait.

Параметр может быть переопределён в конфиге категории или в пресете.

По умолчанию: 3000 (т. е. 3 секунды)

1.1.1.14) (+) TerminateAfterWait=

По окончании ожидания завершения процесса в рамках команды [p?q|...], если процесс так и не завершился самостоятельно, завершить его принудительно вызовом TerminateProcess.

Параметр может быть переопределён в конфиге категории или в пресете.

По умолчанию: 0

1.1.1.15) (+) FreePresetsDirectory=

Папка со свободными пресетами.

По умолчанию: равна PresetsDirectory

1.1.1.16) (+) ShowFreePresets=

Показывать свободные пресеты в дереве.

По умолчанию: 1

1.1.1.17) (–) PreferSystemDialogs=

Использовать по возможности системные диалоги при обработке секций [d...]. Чуть ускоряет обработку данных секций и экономит память, так как используются стандартные системные диалоговые окна.

Параметр может быть переопределён в конфиге категории или в пресете.

По умолчанию: 1

1.1.1.18) (+) RussianSpeakingLCIDs=

Список LCID, для которых следует использовать русские варианты непереводимых с помощью файлов локализации строк.

По умолчанию: 1049,1058,1059,1064,1067,1087,1088,1092,2073,2092,2115 (!не окончательно!)

1.1.1.19) (+) ForbiddenActions=

Определяет, какие разделы иерархии пресетов будут игнорироваться при применении пресета. Число является маской: 1 — PreActions в конфиге программы, 2 — PreActions в конфиге категории, 4 — PreActions в пресете, 8 — PostActions в пресете, 16 — PostActions в конфиге категории, 32 — PostActions в конфиге программы.

Данный параметр имеет более низкий приоритет, чем у параметра запуска 2.3.

По умолчанию: 0 (т. е. все разделы разрешены)

1.1.1.20) (+) ForbiddenSections=

Список типов секций, которые будут игнорироваться при применении пресета. Строка содержит перечисленные через пробел, запятую или точку с запятой префиксы заголовков секций. Можно указать как полностью уточнённый тип секции (например, «pet» — запретить снимать процессы), так и целый класс типов секций, указав общий префикс их названия (например, «r» — запретить любую работу с реестром).

Данный параметр имеет более низкий приоритет, чем у параметра запуска 2.4.

По умолчанию: пусто (т. е. все типы секций разрешены)

1.1.1.20.1.1) (+) Секция [PreActions]

В секции можно определить действия, выполняемые в момент применения любого пресета ДО всех нижезаданных действий.

По умолчанию: пусто

1.1.2) (+) Секция [PostActions]

В секции можно определить действия, выполняемые в момент применения любого пресета ПОСЛЕ всех нижезаданных действий.

По умолчанию: пусто

1.2) (±) Config.ini в папке категории

Содержит параметры данной категории.

1.2.1) (±) Секция [Configuration]

1.2.1.1) (+) DefaultDirectory=

Папка по умолчанию при расчете относительных путей.

По умолчанию: пусто, не переопределяет вышезаданную папку

1.2.1.2) (+) DefaultFile=

Файл по умолчанию. К нему будут применяться изменения, если в заголовке соответствующих секций файл будет опущен. Если задан относительным именем, в расчёт берётся текущее значение DefaultDirectory (причём не имеет значения, какой ключ встречается раньше, DefaultDirectory должен быть рассчитан на этом уровне до DefaultFile).

По умолчанию: пусто, файл не задан

1.2.1.3) (+) PreviewsDirectory=

Папка с изображениями для пресетов (скриншотами), которые отображаются в графическом интерфейсе программы, когда выделен данный пресет. В дальнейшем в самом пресете можно указывать только имя файла-изображения.

По умолчанию: Previews в папке категории

1.2.1.4) (+) Name=

Название категории, отображаемое в дереве в графическом интерфейсе.

По умолчанию: <имя_папки_категории>

1.2.1.5) (+) Description=

Описание категории в свободной форме. Отображается в графическом интерфейсе, когда данная категория выделена. Длина не ограничена. В интерфейсе должна быть возможность прокрутки и выделения/копирования данного текста.

По умолчанию: пусто

1.2.1.6) (+) LastPreset=

Хранит имя последнего применённого пресета. При открытии графического интерфейса этот пресет будет выделен, если его категория будет развёрнута (а это будет так, если она задана параметром LastCategory в конфиге программы).

По умолчанию: пусто. То есть, если LastCategory в конфиге программы задаёт данную категорию, выбран будет заголовок категории.

1.2.1.7) (–) LogFile=

Файл для ведения журнала (см. главу 18).

Этот параметр может быть переопределен для отдельного пресета.

По умолчанию: пусто, не переопределяет вышезаданное значение

1.2.1.8) (–) LogLevel=

Степень подробности ведения журнала (см. главу 18).

Этот параметр может быть переопределен для отдельного пресета.

По умолчанию: пусто, не переопределяет вышезаданное значение

1.2.1.9) (+) CloseProcessWait=

См. 1.1.1.13.

Этот параметр может быть переопределен для отдельного пресета.

1.2.1.10) (+) TerminateAfterWait=

См. 1.1.1.14.

Этот параметр может быть переопределен для отдельного пресета.

1.2.1.11) (–) PreferSystemDialogs=

См. 1.1.1.17.

Этот параметр может быть переопределен для отдельного пресета.

1.2.1.12) (+) PreviewTemplate=

Файл шаблона предпросмотра. Относительный путь разрешается от папки категории (PreviewsDirectory НЕ используется).

1.2.2) (+) Секция [PreActions]

В секции можно определить действия, выполняемые в момент применения пресета данной категории ДО всех обычным образом заданных действий.

По умолчанию: пусто

1.2.3) (+) Секция [PostActions]

В секции можно определить действия, выполняемые в момент применения пресета данной категории ПОСЛЕ всех обычным образом заданных действий.

По умолчанию: пусто

Примеры

Пример №1

2) (±) Параметры запуска

Программа применима как в графическом, так и в консольном режимах.

В консольном режиме задачи выбираются посредством параметров командной строки. Некоторые ключи послужат для молчаливого применения какого-либо действия, другие для запуска графического интерфейса в определённом режиме.

Применим общепринятую в Windows нотацию и будем записывать параметры через прямой слеш.

2.1) (+) /c <имя_папки_категории> ...

Category. Открыть графический интерфейс только с указанной категорией (или несколькими) в дереве. Если открыта лишь одна категория, то дерево вырождается в список пресетов данной категории. Если указано несколько категорий, среди которых имеются не существующие, программа запускается с теми категориями, которые существуют, а о не существующих заносится запись в журнал на уровне 1. Если ни одна из указанных категорий не существует, то запускается обычный интерфейс со всеми категориями.

Ключ /c взаимоисключаем с ключами /C, /a и /A. Ключ должен быть последним в списке параметров.

2.2) (+) /a <имя_папки_категории>\<имя_файла_пресета> ...

Apply. Молчаливо применить указанный пресет (или несколько в указанном порядке). Графический интерфейс ни в каком случае не появляется. Если указано несколько пресетов, среди которых имеются не существующие, программа применяет существующие, а о не существующих заносится запись в журнал на уровне 1. Если ни один пресет не существует, программ завершает работу, не выполнив никаких действий, кроме записи в журнал.

Ключ /a взаимоисключаем с ключами /A, /c и /C. Ключ должен быть последним в списке параметров.

2.3) (+) /!N

Запрещает выполнение указанных секций Pre/PostActions. N — это десятичное число, соответствующее маске: 1 — PreActions в конфиге программы, 2 — PreActions в конфиге категории, 4 — PreActions в пресете, 8 — PostActions в пресете, 16 — PostActions в конфиге категории, 32 — PostActions в конфиге программы.

Параметр имеет более высокий приоритет, чем 1.1.1.19. Однако изменение данной настройки с помощью графического интерфейса имеет наивысший приоритет.

2.4) (+) /!*

Подавить обработку указанных типов секций. Звездочка — одна или несколько букв-ключей. Буквы ключей совпадают с первыми буквами соответствующих секций.

Например, запретить любую работу с реестром:

```
/!r
```

Запретить файловую операцию перемещение:

```
/!fm
```

Так можно подавить и действия, определенные в секциях [p...], но не в Pre/PostActions. Для их подавления требуется другой синтаксис, так как эти секции могут встретиться, как в пресете, так и в конфиге категории, так и в конфиге программы.

Запретить убивать процессы по PID-у:

```
/!pit
```

Параметр имеет более высокий приоритет, чем 1.1.1.20. Однако изменение данной настройки с помощью графического интерфейса имеет наивысший приоритет.

2.5) (+) /C <имя_папки_категории> ...

Аналогично /c, но при поиске категорий игнорируется соответствующие параметры из конфига программы, а значит <имя_папки_категории> должно быть задано либо абсолютно, либо папка должна быть доступна относительно рабочей папки программы.

Ключ /C взаимоисключаем с ключами /c, /a и /A. Ключ должен быть последним в списке параметров.

2.6) (+) /A <имя_файла_пресета> ...

Apply free preset. Аналогично /a, но при обработке пресета игнорируются все вышестоящие конфиги. Таким образом <имя_файла_пресета> должен непосредственно задавать up-файл: путь должен быть задан абсолютно или быть доступен относительно рабочей папки программы. Кроме того в имени обрабатываются переменные окружения.

Ключ /A взаимоисключаем с ключами /a, /c и /C. Ключ должен быть последним в списке параметров.

2.7) (~) <имя_файла_пресета> ...

Режим применения пресетов с распознаванием их свободности. Такой формат запуска предназначен для ассоциирования в системе типа файла ".up" с программой.

Количество применяемых пресетов ограничено лишь максимальной длиной командной строки.

Перед применением очередного пресета из переданного списка проверяется значение ключа FreePreset в разделе [Configuration] up-файла. Если FreePreset=1, то пресет применяется как свободный, т. е. настройки категории и программы игнорируются. Если FreePreset=0 или ключ отсутствует, сначала производится попытка найти конфиг категории и применить её настройки. Отсутствие конфига категории не является ошибкой, настройки программы в любом случае применяются.

В данном формате запуска нельзя указывать никакие параметры. Это означает, что при применении свободного пресета все настройки программы будут иметь значения по умолчанию. Если же применить пресет, не помеченный как свободный, но не поместить его в папку категории, то настройки программы, в том числе запрещённые действия, будут задействованы.

Примеры

Пример №1

3) (±) Локализация

Программа должна быть полностью мультиязычной. Это означает в том числе, что и конфиги и up-файлы могут храниться в UNICODE-кодировке, включая UTF-8.

Перевод требуется не только для графического интерфейса программы, но и для строк, содержащихся в пресетах (если автор посчитает это нужным), таких как Author, Description, Name. При этом не реально хранить альтернативные строки в другом файле. Поэтому синтаксис этих параметров в пресетах нужно расширить.

3.1) Глобальные настройки

3.1.1) (+) Папка с переводами

Файлы переводов хранятся в папке переводов, определяемой параметром в конфиге программы:

```
[Configuration]
LanguagesDirectory=<папка>
```

По умолчанию это папка Languages в папке программы.

3.1.2) (+) Язык программы

В конфиге программы может иметься параметр, определяющий её язык и предпочтительный язык строк в пресетах.

```
[Configuration]
Language=<язык>
```

Язык — имя файла перевода без расширения. Например:

```
Language=Русский
```

Если указанный файл (в примере — «Русский.lng») не существует, это равносильно отсутствию ключа Language.

3.1.2.1) (+) В графическом интерфейсе

Данный параметр можно установить через графический интерфейс.

Там язык можно выбрать из ниспадающего списка, пункты которого составляются из реально имеющихся переводов интерфейса программы.

При этом обязательно должен присутствовать вариант «не задано», который соответствует отсутствию данного ключа.

3.1.2.2) (+) Отсутствие ключа Language

предписывает всегда использовать не уточнённые языком ключи.

Будет использоваться не уточнённый языком вариант ключа, даже если уточнённые языком варианты ключа присутствуют, а не уточнённый — отсутствует. В этом случае значение ключа будет пустой строкой.

3.1.3) Прочие настройки

3.1.3.1) (+) Ключ RussianSpeakingLCIDs

Не все элементы интерфейса нужно переводить на все языки, и не все элементы можно разрешить переводить с помощью файлов локализации (например, лицензионное соглашение). Однако, учитывая страну происхождения программы, было решено встроить поддержку для выборочного включения русского перевода для подобных ключей вместо английского

варианта (по умолчанию). Известно, что по-русски могут читать народы бывшего СССР. Их LCID-ы записаны в ключе RussianSpeakingLCIDs через запятую.

По умолчанию: 1049,1058,1059,1064,1067,1087,1088,1092,2073,2092,2115 (т. е. русский, украинский, белорусский, таджикский, армянский, казахский, киргизский, татарский, молдавский (кириллица), азербайджанский (кириллица), узбекский (кириллица)).

Серьёзные лингвистические исследования не проводились, поэтому данный список по умолчанию может быть изменён впоследствии. Можно изменить список посредством ключа в секции Configuration:

```
[Configuration]
RussianSpeakingLCIDs=1049,1058,1059
```

3.2) (±) Файлы перевода

Файлы перевода хранятся в папке, заданной параметром LanguagesDirectory, и имеет расширение «.lng». Имя файла может быть любым, хотя рекомендуется называть файлы перевода по названию языка перевода, причём на этом языке. Например:

```
English.lng
Русский.lng
Українська.lng
```

Именно эти названия будут пунктами списка выбора перевода в графическом интерфейсе.

3.2.1) (+) Свойства файла перевода

Файлы переводов имеют синтаксис INI-файлов.

В безымянной секции, располагающейся до первой секции и не имеющей заголовка, содержатся основные сведения о переводе.

3.2.1.1) (+) LCID=<LCID>

Задаёт стандартный идентификатор локализации в виде десятичного числа (например, русский — 1049).

Полный список идентификаторов можно найти, например, тут.

Этот идентификатор используется для выборки локализованных значений некоторых ключей в конфигах категорий и самих пресетах, таких как название и описание.

Это значение отображается в графическом интерфейсе настройки рядом со списком переводов при выборе данного перевода.

3.2.1.2) (+) Name=

Название перевода.

Здесь следует указать более дружественное и описательное название перевода. Рекомендуется записывать данную строку на языке перевода.

Эта строка отображается в графическом интерфейсе настройки рядом со списком переводов при выборе данного перевода.

Необходимость такого ключа объясняется тем, что не все файловые системы позволят задать имя файла с использованием любых символов, определённых в UNICODE. В такой ситуации имя файла перевода может быть записано, например, на английском языке, а параметр Name будет содер-

жать локализованное название языка (и перевода).

3.2.1.3) (+) Author=

Автор перевода.

Эта строка отображается в графическом интерфейсе настройки рядом со списком переводов при выборе данного перевода.

3.2.1.4) (+) Modified=<YYYY-MM-DD>

Дата последнего изменения перевода.

Служит в качестве версии перевода.

Это значение отображается в графическом интерфейсе настройки рядом со списком переводов при выборе данного перевода.

3.2.2) (±) Секции файла перевода (в разработке)

Графический интерфейс, система логирования и другие элементы системы находятся в разработке, поэтому стандартизировать конкретные локализуемые строки преждевременно.

3.2.3) (±) Содержимое секций файла перевода

В каждой тематической секции строки перевода содержатся в ключах с именами-номерами. Номера растут по мере добавления новых строк, начиная с нуля. Нулевой ключ обычно задаёт локализованную строку для самого высокоуровневого элемента, хотя это зависит от тематики секции.

Например:

```
[UI-Presets]
0=Пресеты
1=Применить
2=Редактировать
3=Создать...
4=Перечитать
5=Открыть файл...
6=Открыть папку...
7=Свободные пресеты
```

3.3) (±) Локализация конфигов категорий и пресетов

Теоретически локализация может потребоваться для любого элемента пресета или конфига категории. Если используются диалоги-запросы, то их надписи потребуется локализовать. Целевые значения, например, в INI-файле может потребоваться изменять в зависимости от текущей локализации. В то же время существуют типичные для всех пресетов/конфигов ключи, которые желательно переводить.

Таким образом, имеется как упрощённый способ локализации, применимый к ограниченному списку ключей, так и универсальный подход, позволяющий переводить не только значения ключей, но и сами имена ключей, части заголовков секций и вообще что-угодно.

Даже в конфиге категории может потребоваться использовать универсальный подход к локализации, например, при составлении каркаса пресетов данной категории.

3.3.1) (+) Упрощенный подход

Упрощённый подход состоит в том, чтобы кроме обычного ключа со значе-

нием, можно указать локализованную версию ключа:

```
<ключ>=<не локализованное значение>
<ключ>.<язык>=<локализованное значение>
```

Не локализованный вариант используется тогда, когда язык не задан в конфиге программы или нужный локализованный вариант ключа отсутствует.

3.3.2) (–) Универсальный подход

Все локализуемые строки можно задать в виде псевдонимов (см. 17). Каждый псевдоним поддерживает упрощённый подход к локализации своего значения. Например:

```
[Aliases]
Hello=Hello world!
Hello.1049=Привет, мир!
LocalizableKey=EngKey
LocalizableKey.1049=RusKey
```

Данная секция анализируется самой первой, поэтому практические везде (даже в последующих ключах в секции [Aliases]) можно вставлять ссылки на ключи. Они будут заменены на соответствующие значения до дальнейшего разбора. Ссылка вставляется путём помещения имени псевдонима в тупые кавычки. К предыдущему примеру:

```
[dm|Dialog header|`Hello`]
[im|Some.ini|SomeSection]
`LocalizableKey`=Value
```

3.3.3) (+) Локализация конфига категории

Список ключей, поддерживающих упрощённый подход к локализации, ограничен в соответствии со степенью востребованности их локализации.

3.3.3.1) (+) Name.<LCID>=

Локализованное название категории.

3.3.3.2) (+) Description.<LCID>=

Локализованное описание категории.

3.3.4) (+) Локализация пресета

Список ключей, поддерживающих упрощённый подход к локализации, ограничен в соответствии со степенью востребованности их локализации.

3.3.4.1) (+) Name.<LCID>=

Локализованное название пресета.

3.3.4.2) (+) Description.<LCID>=

Локализованное описание пресета.

3.3.4.3) (+) Author.<LCID>=

Локализованное перечисление имён авторов.

Локализация данного ключа совершенно не обязательна, хотя поддерживается.

Примеры

Пример №1

4) (±) Структура пресета (в разработке)

Пресет — текстовый файл с расширением «.up», имеющий ini-подобный синтаксис. Расширение пресетов вне категорий (свободных) может быть любым, так как для их применения потребуется явно указать up-файл. Всё содержимое up-файла, кроме секции Configuration, записывается на языке UP. Секция Configuration позволяет инициализировать некоторые переменные и записывается в обычном ini-синтаксисе.

Пресет способен содержать всю необходимую информацию для применения любых изменений в системе. В нём же содержится некоторая служебная информация, предназначенная для графического интерфейса.

На практике же пресет редко будет одиноким. Пресеты логично объединяются в категории. Благодаря этому многие настройки можно вынести из пресета и сделать общими для всех пресетов заданной категории, а некоторые настройки можно сделать общими вообще для всех пресетов всех категорий.

4.1) (±) Секция [Configuration]

4.1.1) (+) DefaultDirectory=

Папка по умолчанию при расчете относительных путей в этом пресете.

По умолчанию: пусто, не переопределяет вышезаданную папку

4.1.2) (+) DefaultFile=

Файл по умолчанию. К нему будут применяться изменения, если в заголовке соответствующих секций файл будет опущен. Если задан относительным именем, в расчёт берётся текущее значение DefaultDirectory (причём не имеет значения, какой ключ встречается раньше, DefaultDirectory должен быть рассчитан на этом уровне ДО DefaultFile).

По умолчанию: пусто, файл не задан, но не переопределяет вышезаданный файл

4.1.3) (+) Preview=

Имя файла-картинки, которая отображается в графическом интерфейсе программы, когда выделен данный пресет. Поддерживаются bmp, png, jpeg, gif (анимированный), размер не ограничен.

По умолчанию: <имя_пресета>.png, однако отсутствие этого файла не является ошибкой и не отмечается в журнале ни на каком уровне

4.1.4) (+) Name=

Название пресета, отображаемое в дереве в графическом интерфейсе.

По умолчанию: <имя_пресета_без_расширения_.up>

4.1.5) (+) Description=

Описание пресета в свободной форме. Отображается в графическом интерфейсе, когда данный пресет выделен. Длина не ограничена. В интерфейсе должна быть возможность прокрутки и выделения/копирования данного текста.

По умолчанию: пусто

4.1.6) (+) Author=

Информация об авторе пресета. Отображается в графическом интерфейсе, когда данный пресет выделен. Длина не ограничена, но рекомендуется краткость. В интерфейсе должна поддерживаться возможность выделения/копирования данного текста, но отображён он будет в одной строке. Здесь же могут присутствовать координаты обратной связи — URL должен обрабатываться автоматически.

По умолчанию: пусто

4.1.7) (–) Created=

Дата создания или начала разработки пресета в ISO-формате (YYYY-MM-DD). Задаётся программой только при первом сохранении или же автором вручную.

По умолчанию: не задано, в интерфейсе не отображается

4.1.8) (–) Modified=

Дата последней модификации пресета в ISO-формате (YYYY-MM-DD). Задаётся программой при каждом сохранении или же автором вручную.

По умолчанию: не задано, в интерфейсе не отображается

4.1.9) (–) LogFile=

Файл для ведения журнала (см. главу 18).

По умолчанию: пусто, не переопределяет вышезаданное значение

4.1.10) (–) LogLevel=

Степень подробности ведения журнала (см. главу 18).

По умолчанию: пусто, не переопределяет вышезаданное значение

4.1.11) (+) CloseProcessWait=

См. 1.1.1.13.

4.1.12) (+) TerminateAfterWait=

См. 1.1.1.14.

4.1.13) (–) PreferSystemDialogs=

См. 1.1.1.17.

4.1.14) (–) FreePreset=

Определяет свободность пресета.

Если FreePreset=1, то пресет используется как свободный в сценариях 2.6 и 2.7, а также во время обычного применения пресета посредством графического интерфейса, то есть конфиги категории и программы игнорируются.

Если FreePreset=0, то пресет не является свободным, и должен применяться в составе категории. Если несвободный пресет применяется безотносительно категории (посредством 2.7), то программа всё равно попытается найти конфиг категории рядом с пресетом и применить настройки из него, и в любом случае применит настройки из конфига программы. При попытке применения несвободного пресета посредством 2.6 пресет не применяется,

а пользователю выдаётся сообщение о том, что пресет не является свободным.

По умолчанию: 0

4.2) (+) Секции [PreActions] и [PostActions]

В секциях можно определить действия, выполняемые в момент применения пресета соответственно ДО и ПОСЛЕ всех действий, заданных с помощью секций (см. 12).

По умолчанию: пусто

4.3) (+) Секция [Actions]

Позволяет объединить несколько простых действий (см. 12). Данная секция может встретиться в пресете сколько угодно раз.

По умолчанию: не существует

4.4) (±) Секции-действия

Секции-действия, собственно, определяют, что делает пресет. Каждый тип секций описан далее отдельно. Тип секции определяется первой буквой заголовка секции. В зависимости от типа секции синтаксис её заголовка и ключей может меняться.

Данные секции никак не влияют на графический интерфейс программы, а лишь служат для неё планом алгоритма применения пресета.

Предусмотрены следующие типы секций.

4.4.1) (±) [i...]

Различные операции с INI-файлами.

4.4.1.1) (±) [I...]

Особые действия с расширенными INI-файлами.

4.4.2) (±) [r...]

Различные операции с реестром.

4.4.3) (–) [x...]

Различные операции с XML-файлами.

4.4.4) (±) [f...]

Различные файловые операции.

4.4.5) (±) [p...]

Различные действия над процессами, окнами и окружением.

4.4.6) (–) [t...]

Различные действия над текстовыми файлами.

4.4.7) (–) [b...]

Различные действия над бинарными файлами.

4.4.8) (–) [d...]

Средства взаимодействия с пользователем посредством различных диалоговых окон.

4.4.9) (–) [w...]

Средства взаимодействия окнами: посылка сообщений, изменение свойств.

4.5) (–) Секции условий

Данные секции применяются в тех случаях, когда требуется проверить некоторое условие. Они применяются внутри некоторых метаконструкций, а также могут объединяться в процедуры.

4.6) (±) Метасекции

Метасекции служат для управления процессом применения пресета. Данные секции в том числе позволяют воспользоваться рядом алгоритмических конструкций, таких как условия и циклы.

4.7) (–) Секции отката

Такие секции служат для извлечения имеющихся настроек и сохранения их в up-файлы отката.

Примеры

Пример №1

5) (±) INI-файлы (в разработке)

Секции для работы с INI-файлами предваряются буквой «**i**», например:

```
[ir|c:\tc\wincmd.ini|Colors]
InverseCursor=1
InverseSelection=0
BackColor=788488
...
```

5.1) (±) Заголовок

Заголовок секции имеет формат:

5.1.1) (+) [i<буква_действия>|<INI-файл>|<секция_в_INI-файле>]

Имя INI-файла задаётся любым способом. Берутся в расчёт текущие значения DefaultDirectory и DefaultFile, то есть имя INI-файла вообще может отсутствовать:

```
[i<буква_действия>||<секция_в_INI-файле>]
```

Имя секции может отсутствовать, а заголовок имеет вид:

```
[i<буква_действия>|<INI-файл>|]
```

В таком случае действие применяется к гипотетической безымянной секции, ключи которой расположены в начале INI-файла до объявления первой нормальной секции.

При этом последний символ «|» должен присутствовать, иначе заголовок превращается в вариант, описанный в 5.1.3.

Если интересуют настоящие секции без имени, т. е. имеющие заголовок вида «[]» (хотя таких секций по стандарту быть не должно), то потребуется воспользоваться шаблоном имени секции (см. 5.1.2.3 и 5.1.2.4).

5.1.2) (–) Шаблоны секций

Имя секции должно совпадать с таковой в целевом INI-файле. При этом регистр букв не учитывается.

Вместо имени секции могут присутствовать некоторые спец. символы:

5.1.2.1) (–) [i<буква_действия>|<INI-файл>|?]

Секция не известна. Следует найти ПЕРВОЕ вхождение указанных ниже ключей и применить указанное действие.

5.1.2.2) (–) [i<буква_действия>|<INI-файл>|*]

Секция не известна. Следует найти ВСЕ вхождения указанных ниже ключей и применить указанное действие.

5.1.2.3) (–) [i<буква_действия>|<INI-файл>|?<шаблон>]

Имя секции определяется заданным шаблоном. Следует найти ПЕРВУЮ секции, чьё имя удовлетворяет шаблону и применить указанное действие. Шаблон здесь имеет синтаксис расширенных регулярных выражений.

5.1.2.4) (–) [i<буква_действия>|<INI-файл>|*<шаблон>]

Имя секции определяется заданным шаблоном. Следует найти ВСЕ секции, чьё имя удовлетворяет шаблону и применить указанное действие.

5.1.3) (+) [i<буква_действия>|<INI-файл>]

Секция в заголовке не указывается. Не указывается и последний символ «|», иначе конструкция превращается в 5.1.1.

Нужная секция INI-файла должна быть указана в каждом ключе в виде:

```
<имя_секции>]<имя_ключа>=<значение>
```

Это удобная альтернатива записи множества секций-действий, обрабатывающих один INI-файл, но разные секции в нём.

Например:

```
[im|usercmd.ini|em_BackupSelected]
menu=Создать резервные копии выделенных файлов/папок
[im|usercmd.ini|em_BAR_Addon]
menu=Панель дополнительных кнопок
[im|usercmd.ini|em_BAR_Editors]
menu=Редакторы
```

можно заменить на:

```
[im|usercmd.ini]
em_BackupSelected]menu=Создать резервные копии выделенных файлов/папок
em_BAR_Addon]menu=Панель дополнительных кнопок
em_BAR_Editors]menu=Редакторы
```

Кроме оптимизации синтаксиса данная конструкция оптимизирует операции доступа к файлам, так как открытие и сохранение INI-файла в первом случае произойдёт 3 раза (по разу для каждой секции), а во втором — всего 1 раз, а все операции будут происходить с программной моделью в памяти.

Имя INI-файла задаётся любым способом. Берутся в расчёт текущие значения DefaultDirectory и DefaultFile, то есть имя INI-файла вообще может отсутствовать.

Данная конструкция применима не со всеми типами действий. В каждом пункте в разделе 5.2 и 5.3 будет особо оговорена возможность применения в данной конструкции.

Очевидно, для каждой целевой секции будет применяться одинаковое действие, указанное в заголовке. Также очевидно, что действие применяется к одному ключу за раз. В некоторых ситуациях применение данной конструкции наоборот усложнит синтаксис и увеличит многословность пресета. При выборе способа записи рекомендуется также ознакомится с конструкцией use/unuse (10.4).

5.2) (±) Действия

5.2.1) (+) r

Обычная перезапись. Указанная секция в INI-файле удаляется, а вместо неё вставляется полностью секция из пресета.

5.2.1.1) (+) Если ключи отсутствуют,

то в целевом INI-файле создаётся пустая секция с указанным именем. Если в целевом INI-файле существовала одноименная секция, то её содержимое уничтожается полностью.

В случае применения в конструкции 5.1.3, секция игнорируется.

5.2.1.2) (+) Возможно ограниченное применение в конструкции 5.1.3.

Ограничение заключается в том, что строки выполняются последователь-

но по одной, а значит, затронутые секции в целевом файле останутся лишь с одним ключом: с тем, который устанавливается последним.

Если в строке не упоминается ключ со значением (т. е. строка заканчивается символом «]»), в целевом INI-файле создаётся пустая секция с указанным именем. Если в целевом INI-файле существовала одноименная секция, то её содержимое уничтожается полностью.

5.2.2) (+) m

Обычное слияние. Уже имеющиеся ключи, не упоминаемые в пресете, остаются без изменения. Совпадающие ключи обновляются значениями из пресета. Ключи из пресета, отсутствующие в целевой секции, вставляются в неё.

5.2.2.1) (+) Если ключи отсутствуют,

а в целевом INI-файле отсутствует указанная секция, то в нём создаётся пустая секция с указанным именем.

5.2.2.2) (+) Возможно применение в конструкции 5.1.3.

5.2.3) (+) x

Взаимная перезапись (обмен). Содержимое секции в целевом файле и секции в пресете целиком меняются местами.

Запрещено применение в конструкции 5.1.3.

5.2.4) (+) a

Добавление. Из пресета берутся и вставляются только отсутствующие в целевом INI-файле ключи.

5.2.4.1) (+) Если ключи отсутствуют,

то в целевом INI-файле создаётся пустая секция с указанным именем, даже если одноимённая секция уже существует.

5.2.4.2) (?) Возможно применение в конструкции 5.1.3.

5.2.5) (+) d

Удаление ключей. Из целевого INI-файла удаляются ключи, присутствующие в пресете.

Если задано только имя ключа (без знака «=»), то происходит безусловное удаление ключа. Если знак «=» присутствует, то удаление произойдёт только если данному ключу присвоено указанное значение (пусть даже пустое).

5.2.5.1) (+) Если ключи отсутствуют,

секция игнорируется.

5.2.5.2) (+) Возможно применение в конструкции 5.1.3.

Например:

```
[id|some.ini]
section1]key1
section1]key2=value
section2]key3
```

5.2.6) (+) D

Удаление секции. Из целевого INI-файла удаляются указанная секция це-

ликом.

Если задан хотя бы один ключ, это воспринимается как условие удаления. Секция удаляется, только если в целевой секции имеются все указанные ключи (возможно, с указанными значениями), т. е. применяется операция «И».

Имеющиеся ключи, не упоминаемые в пресете, ни на что не повлияют.

5.2.6.1) (+) Если ключи отсутствуют,

произойдет безусловное удаление секции.

5.2.6.2) (+) Возможно ограниченное применение в конструкции 5.1.3.

Ограничение заключается в том, что в качестве условия можно указать не более одного ключа для каждой целевой секции. Однако, если в конструкции несколько строк будут указывать на одну целевую секцию, они сработают как комбинация условий по «ИЛИ», т. е. первая удачная проверка условия приведёт к удалению секции, а дальнейшие упоминания удалённой секции будут проигнорированы. Например:

```
[iD|some.ini]
section1]
section2]key1=or-cond1
section2]key2=or-cond2
section3]key3
```

5.2.7) (+) c<режим>

Комментирование ключей. В целефом INI-файле комментируются ключи, указанные в пресете. Комментирование заключается в добавлении знака «;» в начало строки.

Если задано только имя ключа (без знака «=»), то происходит безусловное комментирование ключа. Если знак «=» присутствует, то комментирование произойдёт только если данному ключу присвоено указанное значение (пусть даже пустое).

Пример см. в 5.2.8.

Режим определяет, собственно, действие:

+ (плюс)

Закомментировать. Если уже закомментировано, ничего не происходит.

– (минус)

Раскомментировать. Если и так не закомментировано, ничего не происходит.

любой другой символ (в том числе его отсутствие)

Инверсия, то есть закомментировать, если не было закомментировано, и раскомментировать, если было.

5.2.7.1) (+) Если ключи отсутствуют,

секция игнорируется.

5.2.7.2) (+) Возможно применение в конструкции 5.1.3.

5.2.8) (+) C<режим>

Комментирование секции. В целевом INI-файле комментируется указанная секция целиком. Комментирование заключается в добавлении знака «;» в начало каждой строки.

Если задан хотя бы один ключ, это воспринимается как условие комментирования. Секция комментируется, только если в целевой секции имеются все указанные ключи (возможно, с указанными значениями), т. е. применяется операция «И».

Если задано только имя ключа (без знака «=»), то значение ключа не учитывается. Если знак «=» присутствует, то учитывается значение (пусть даже пустое).

Режимы аналогичны пункту 5.2.7.

Примеры:

```
[iC+|wincmd.ini|Associations]
[ic-|wincmd.ini|Configuration]
DriveLib
IconLib
[ic^|wincmd.ini|Configuration]
RestrictInterface=37
```

5.2.8.1) (+) Если ключи отсутствуют,

произойдет полное безусловное комментирование секции.

5.2.8.2) (+) Возможно ограниченное применение в конструкции 5.1.3.

Ограничение аналогично описанному в 5.2.6.

5.2.9) (+) n

Переименование ключей. В целевом INI-файле ключи, указанные в пресете, переименовываются в заданные значения.

Значение не может быть опущено. В противном случае указанный ключ игнорируется, а в лог заносится сообщение на уровне 2 — неверные синтаксис секции.

Например:

```
[in|some.ini|Section]
OldKey1=NewKey1
…
```

5.2.9.1) (+) Если ключи отсутствуют,

произойдет безусловное переименование секции.

5.2.9.2) (+) Возможно применение в конструкции 5.1.3.

Следует учитывать, что строки секции применяются по-отдельности по порядку, т. е. при желании можно переименовать один и тот же целевой ключ несколько раз в одной секции (хотя в этом не будет никакого смысла, т. к. изменения произойдут только в программной модели в памяти). Например:

```
[in|some.ini]
section1]OldKey1=NewKey1
section2]OldKey2=NewKey2
section2]OldKey3=NewKey3
section2]NewKey2=NewKey4
```

5.2.10) (+) N

Переименование секции. В целевом INI-файле переименовывается указанная секция целиком. Новое имя указывается в последнем параметре заголовка:

```
[iN|some.ini|OldName|NewName]
```

Если задан хотя бы один ключ, это воспринимается как условие переименования. Секция переименовывается, только если в целевой секции имеются все указанные ключи (возможно, с указанными значениями), т. е. применяется операция «И».

Если задано только имя ключа (без знака «=»), то значение ключа не учитывается. Если знак «=» присутствует, то учитывается значение (пусть даже пустое)

5.2.10.1) (+) Если ключи отсутствуют,

произойдет безусловное переименование секции.

5.2.10.2) (+) Возможно ограниченное применение в конструкции 5.1.3.

Ограничение аналогично описанному в 5.2.6.

5.2.11) (+) M

Управляемое слияние. Совпадающие ключи обновляются значениями из пресета. Ключи из пресета, отсутствующие в целевой секции, вставляются в неё. Ключи из пресета без значений (и без знака «=») остаются в целевой секции нетронутыми. Все остальные ключи в целевой секции удаляются.

Если в целевой секции имеются несколько ключей с одинаковым именем, а в пресете ключ с таким именем упоминается (без значения и без знака «=») также несколько раз, то в результирующей секции останется столько таких ключей, сколько их в пресете, притом в том же порядке. Впрочем одноимённых ключей в одной секции по стандарту быть не должно.

Если ключ упомянут в пресете без значения и без знака «=», а в целевой секции такой ключ отсутствует, то ничего не происходит.

Запрещено применение в конструкции 5.1.3. Запрет связан с тем, что такое применение нарушало бы главную идею управляемого слияния: описание шаблона для целевой секции.

....

5.3) (±) Особые действия

Существует ряд ситуаций, в которых невозможно или неудобно использовать обычные действия. Кроме того, некоторые программы используют различные расширения синтаксиса INI-файлов.

Типичным примером может быть Total Commander, который использует ключ RedirectSection для размещения секций в отдельных файлах. Поддержка такого расширения со стороны kIT Universal Presets определённо будет очень полезна.

Особые действия, связанные с INI-файлами, предваряются буквой «**I**».

5.3.1) (+) Ir

Аналогично «**ir**», но с поддержкой RedirectSection.

Возможно применение в конструкции 5.1.3.

5.3.2) (+) Im

Аналогично «**im**», но с поддержкой RedirectSection.

Возможно применение в конструкции 5.1.3.

5.3.3) (+) Ix

Аналогично «**ix**», но с поддержкой RedirectSection.

Запрещено применение в конструкции 5.1.3.

5.3.4) (+) Ia

Аналогично «**ia**», но с поддержкой RedirectSection.

Возможно применение в конструкции 5.1.3.

5.3.5) (+) Id

Аналогично «**id**», но с поддержкой RedirectSection.

Возможно применение в конструкции 5.1.3.

5.3.6) (+) ID

Аналогично «**iD**», но с поддержкой RedirectSection, т. е. удаляется секция, на которую указывается с помощью ключа RedirectSection.

Если RedirectSection=0, то будет секция будет очищена от всех других ключей, но не будет удалена совсем, т. е. останется только RedirectSection=0.

Если указанный файл отсутствует, ничего не происходит.

5.3.7) (+) Ic

Аналогично «**ic**», но с поддержкой RedirectSection.

Возможно применение в конструкции 5.1.3.

5.3.8) IC

?

5.3.9) In

?

5.3.10) (+) IN

Замкнуто переименовывает секцию. В третьем и четвёртом параметрах заголовка можно указать соответственно префикс и суффикс имени. Префикс и суффикс добавляются к имени секции или же удаляются из него, если они уже имелись в составе имени секции.

Ключи не ожидаются и игнорируются.

И префикс, и суффикс могут быть независимо опущены. Опущенный параметр равен пустой строке. Если опущены оба параметра, префикс получает значение «-» (минус).

Если в целевом файле уже существует секция с именем, в которое требуется переименовать исходную, то содержимое данной секции становится содержимым исходной секции, а содержимое исходной — содержимым данной, то есть секции обмениваются местами.

Поддержка RedirectSection алгоритмически избыточна, а потому не предусмотрена.

```
[IN|wincmd.ini|Associations] ;переименование в "-Associations"
[IN|some.ini|SomeSection|P|S] ;переименование в "PSomeSectionS"
[IN|some.ini|SomeSection||!] ;переименование в "SomeSection!"
```

Запрещено применение в конструкции 5.1.3. Это слишком специфичное действие, с особым синтаксисом заголовка.

Примеры

Пример №1

6) (±) Реестр (в разработке)

Секции для работы с реестром предваряются буквой «**r**»:

```
[r...]
```

Строение реестра очень похоже на строение INI-файлов. Однако первое же отличие в терминологии: ключом называется ветка реестра (аналог секции), а внутри ключей находятся параметры со своими значениями. В реестре параметры (INI-ключи) также могут иметь единственное значение. Каждый параметр (INI-ключ) располагается в своём ключе реестра (INI-секции). Другим отличием от INI-файлов является то, что ключ (INI-секция) может содержать другие ключи (INI-секции) внутри. Таким образом все ключи образуют иерархическую структуру.

В системе предусмотрен экспорт/импорт ключей реестра. Вся необходимая для этого информация сохраняется в reg-файлы. Синтаксис reg-файлов расширяет синтаксис INI-файлов:

```
[HKEY_CURRENT_USER\Software\Company\Product]
"Editor"="notepad.exe"
"ShowSystemMenu"=dword:00000001
"FolderHistory"=hex:43,00,3a,00,5c,00,00,00,1a,04,4c,04,4e,\
  04,42,04,35,04,40,04,5c,00,00
```

Для удобства составления пресетов следует сохранить синтаксис reg-файлов хотя бы в части описания ключей и значений.

6.1) (±) Заголовок

Одна r-секция пресета обычно работает только с одним ключом реестра. Для однозначного указания ключа всегда можно указать «путь» в системном реестре.

6.1.1) (+) [r<буква_действия>|<путь_к_ключу>]

Вариант, приближенный к виду reg-файлов. Внутри могут непосредственно адресоваться параметры заданного ключа реестра в любом синтаксисе.

Даже если указать путь к ключу в заголовке, можно использовать полные пути к параметрам, даже адресуя ключи и параметры, не являющиеся потомками ключа, заданного в заголовке.

6.1.2) (+) [r<буква_действия>|]

Путь к ключу могут отсутствовать. Тогда внутри r-секции придётся указывать полные пути к параметрам. В этом случае внутри секции нельзя использовать reg-синтаксис для имён ключей (с двойными кавычками) без указания пути, т. е. строка секции не может начинаться с двойной кавычки.

6.2) (±) Ключи

Каждый ключ секции (строка) затрагивает один параметр в каком-то ключе реестра (в некоторых случаях — целый ключ реестра). Так как в заголовке не обязательно задавать путь к ключу реестра, в ключах секции именах параметров можно задавать абсолютно. Однако в этом случае следует отказаться от reg-синтаксиса, строка секции не должна начинаться с двойной кавычки.

6.2.1) (±) Экранирование

В именах параметров и в текстовых значениях допустимо применение двойных кавычек, но синтаксис присваивает им особый смысл — в них заключаются имена параметров и строка текста в параметрах типа REG_SZ. Поэтому необходимы правила экранирования.

6.2.1.1) (±) \"

Двойная кавычка экранируется обратным слешем.

6.2.1.2) (±) \\

Обратный слеш удваивается.

6.2.1.3) (±) \<перевод строки>

Если слеш находится в конце строки, значение параметра продолжается с новой строки.

6.2.2) (±) Имя параметра

Имя параметра, как обычно, предшествует знаку "=" и может быть задано двумя способами. Функционально данные способы не различимы.

6.2.2.1) (+) "имя"=

Как принято в reg-файлах.

6.2.2.2) (+) @=

Значение по умолчанию для данного ключа реестра. Может иметь тип только RES_SZ.

6.2.2.3) (+) имя=

Как принято в INI-файлах. При этом внутри имени может встретиться двойная кавычка, но она не может быть первым символом имени. Соответственно в данном варианте записи не работает экранирование (не работает только в имени параметра). Запрещённое имя — @.

Этот вариант использовать не рекомендуется. Он полезен при переносе значений из INI-файлов в реестр.

6.2.3) (+) Значение параметра

Так как в реестре могут храниться параметры различных типов, для значений ключей секции предусмотрен особый синтаксис, полностью совпадающий с синтаксисом reg-файлов.

6.2.3.1) (+) <имя_параметра>="<текстовые_данные>"

Это особый и самый распространённый тип данных — строка (REG_SZ). Строка заключается в двойные кавычки.

6.2.3.2) (+) <имя_параметра>=<тип_данных>:<данные>

Для всех остальных типов данных.

6.3) (+) Описание пути

Так как реестр представляет собой иерархическую структуру, однозначно указать на любой ключ реестра или параметр можно с помощью некоторого пути.

Путь состоит из имён ключей (а также имени куста), разделённых обрат-

ным слешем.

6.3.1) (+) <куст>\<ключ>[\<ключ>]...

Такая форма применима в заголовке, так как позволяет задать абсолютный путь к ключу реестра.

В имени ключа реестра допускается использовать символ "]". Программа должна правильно отрабатывать такой случай.

6.3.2) (+) <куст>\<ключ>[\<ключ>]...\<параметр>

Такая форма применима внутри r-секции для задания абсолютного имени параметра. Имя параметра формируется по правилам 6.2.2.1 и 6.2.2.2 (но не 6.2.2.3), то есть кавычки вокруг имени параметра обязательны (если не параметр по умолчанию.

6.3.3) (+) Определение куста

Кусты это стандартные глобальные разделы реестра, определяемые системой. У каждого куста своё назначение и особенности функционирования. Определены 6 кустов:

HKEY_CLASSES_ROOT	**HKCR**
HKEY_CURRENT_USER	**HKCU**
HKEY_LOCAL_MACHINE	**HKLM**
HKEY_USERS	**HKU**
HKEY_CURRENT_CONFIG	**HKCC**
HKEY_PERFORMANCE_DATA	**HKPD**

Куст можно задавать в двух формах. Функционально формы эквивалентны.

6.3.3.1) (+) полная

Например:

```
HKEY_CURRENT_USER
```

6.3.3.2) (+) сокращённая

Например:

```
HKCU
```

6.4) (+) Типы данных

В реестре могут хранится данные различных типов. В зависимости от указанного типа данные интерпретируются различным образом.

6.4.1) (+) "<текстовые_данные>"

REG_SZ (1). Текстовая строка в формате, удобном для восприятия человеком. Значениям, представляющим собой описания компонентов, обычно присваивается именно этот тип данных. Имеет фиксированную длину.

6.4.2) (+) hex(2):<hex-байт>[,<hex-байт>]...

REG_EXPAND_SZ (2). Расширяемая строка данных. Эта строка представляет собой текст, содержащий переменную, которая может быть заменена при вызове со стороны приложения.

6.4.3) (+) hex:<hex-байт>[,<hex-байт>]...

REG_BINARY (3). Необработанные двоичные данные. Большинство сведений об аппаратных компонентах хранится в виде двоичных данных и вы-

водится в редакторе реестра в шестнадцатеричном формате.

Каждый байт записывается двумя hex-цифрами и может быть в диапазоне от `00` до `ff`. Регистр hex-букв не учитывается (в reg-файлах принято использовать нижний).

В reg-файлах принято добавлять лидирующие нули.

6.4.4) (+) dword:<hex-число>

REG_DWORD (4). Данные, представленные целым числом (4 байта, 32 бита). Многие параметры служб и драйверов устройств имеют этот тип и отображаются в двоичном, шестнадцатеричном или десятичном форматах.

Hex-число может быть в диапазоне от `00000000` до `ffffffff`. Регистр hex-букв не учитывается (в reg-файлах принято использовать нижний).

В reg-файлах принято добавлять лидирующие нули.

6.4.5) (+) hex(5):<hex-байт>[,<hex-байт>]...

REG_DWORD_BIG_ENDIAN (5). Данные, представленные целым числом (4 байта, 32 бита) с обратным порядком байт.

Hex-число может быть в диапазоне от `00000000` до `ffffffff`. Регистр hex-букв не учитывается (в reg-файлах принято использовать нижний).

В reg-файлах принято добавлять лидирующие нули.

6.4.6) (+) hex(6):<hex-байт>[,<hex-байт>]...

REG_LINK (6). Символическая ссылка в формате Юникод.

6.4.7) (+) hex(7):<hex-байт>[,<hex-байт>]...

REG_MULTI_SZ (7). Многострочное поле. Значения, которые фактически представляют собой списки текстовых строк в формате, удобном для восприятия человеком, обычно имеют именно этот тип данных. Записи разделяются пробелами, запятыми или другими символами.

6.4.8) (+) hex(8):<hex-байт>[,<hex-байт>]...

REG_RESOURCE_LIST (8). Последовательность вложенных массивов. Служит для хранения списка ресурсов, которые используются драйвером устройства или управляемым им физическим устройством. Обнаруженные данные система сохраняет в разделе \ResourceMap. В окне редактора реестра эти данные отображаются в виде двоичного параметра в шестнадцатеричном формате.

6.4.9) (+) hex(9):<hex-байт>[,<hex-байт>]...

REG_FULL_RESOURCE_DESCRIPTOR (9). Последовательность вложенных массивов. Служит для хранения списка ресурсов, которые используются физическим устройством. Обнаруженные данные система сохраняет в разделе \HardwareDescription. В окне редактора реестра эти данные отображаются в виде двоичного параметра в шестнадцатеричном формате.

6.4.10) (+) hex(a):<hex-байт>[,<hex-байт>]...

REG_RESOURCE_REQUIREMENTS_LIST (10). Последовательность вложенных массивов. Служит для хранения списка драйверов аппаратных ресурсов, которые могут быть использованы определенным драйвером устрой-

ства или управляемым им физическим устройством. Часть этого списка система записывает в раздел \ResourceMap. Данные определяются системой. В окне редактора реестра они отображаются в виде двоичного параметра в шестнадцатеричном формате.

6.4.11) (+) hex(b):<hex-байт>[,<hex-байт>]...

REG_QWORD (11). Данные, представленные в виде 64-разрядного целого. Начиная с Windows 2000, такие данные отображаются в окне редактора реестра в виде двоичного параметра.

6.4.12) (+) hex(0):<hex-байт>[,<hex-байт>]...

REG_NONE (0). Данные, не имеющие определенного типа. Такие данные записываются в реестр системой или приложением. В окне редактора реестра отображаются в виде двоичного параметра в шестнадцатеричном формате.

6.4.13) (+) hex(HH):<hex-байт>[,<hex-байт>]...

Пользовательский тип.

Номер типа записывается в шестнадцатеричном виде и может быть в диапазоне от `0c` (12) до `ff` (255). Регистр hex-букв не учитывается (в reg-файлах принято использовать нижний).

В reg-файлах принято опускать лидирующие нули.

6.5) (±) Действия

6.5.1) (+) r

Обычная перезапись. Указанный ключ реестра (или целая ветка) удаляется, а вместо него вставляется ключ, описанный в пресете.

6.5.1.1) (+) Если ключи отсутствуют,

то в реестре создаётся пустой ключ с указанным именем (с созданием всех промежуточных ключей). Если в реестре существовал одноименный ключ, то он уничтожается (или ветка полностью).

Параметры можно задавать в синтаксисе 6.3.2, т. е. указывать абсолютный путь. В этом случае всё содержимое данного ключа удаляется, а затем создаётся данной параметр.

Ключ в заголовке вообще можно не определять, задавая его в каждой строке секции. В этом случае в каждом затронутом ключе реестра останется по одной записи — последней по порядку в секции. См. аналогичное поведение в 5.2.1.2.

Например:

```
[rr|HKCU\Some\Key]
"Item1"="Value1"
"Item2"=dword:00000001
[rr|]
HKCU\More\Key\@="Value1"
HKCU\More\Key\"Item2"="Value2"
HKCU\Some\Key\"Item3"=hex:0a,1b,2c
```

Содержимое ключей HKCU\Some\Key и HKCU\More\Key будет потеряно. В ключе HKCU\Some\Key будут созданы три параметра, а в HKCU\More\Key — один параметр и параметр по умолчанию.

6.5.2) (+) m

Обычное слияние. Уже имеющиеся параметры в ключе реестра, не упоминаемые в пресете, остаются без изменения. Совпадающие параметры обновляются значениями из пресета. Параметры из пресета, отсутствующие в целевом ключе реестра, вставляются в него.

6.5.2.1) (+) Если ключи отсутствуют,

а в реестре отсутствует указанный ключ, то в нём создаётся пустой ключ с указанным именем (с созданием всех промежуточных ключей).

Например:

```
[rm|HKCU\Some\Key]
"Item1"="Value1"
"Item2"=dword:00000001
[rm|]
HKCU\More\Key\@="Value1"
HKCU\More\Key\"Item2"="Value2"
HKCU\Some\Key\"Item3"=hex:0a,1b,2c
```

В ключе HKCU\Some\Key будут созданы (если отсутствовали) или переопределены три параметра, а в HKCU\More\Key — один параметр и параметр по умолчанию.

6.5.3) (–) x

Взаимная перезапись (обмен). Содержимое указанного ключа в реестре и r-секция в пресете целиком меняются местами. Вложенные ключи при этом остаются неизменными.

Например:

```

```

6.5.4) (+) a

Добавление. Из пресета берутся и вставляются только отсутствующие в целевом ключе реестра параметры.

Если ключи отсутствуют, r-секция игнорируется. Она лишена смысла, так как в реестре запрещается создавать одноимённые ключи.

Например:

```
[ra|HKCU\Some\Key]
"Item1"="Value1"
"Item2"=dword:00000001
[ra|]
HKCU\Some\Key\"Item3"=hex:0a,1b,2c
```

Если в ключе HKCU\Some\Key отсутствовал параметр Item1 (или Item2, или Item3), они соответственно будут созданы.

6.5.5) (+) d

Удаление параметров. Из целевого ключа реестра удаляются параметры, присутствующие в пресете.

Если ключи отсутствуют, r-секция игнорируется.

Если задано только имя параметра (без знака «=»), то происходит безусловное удаление параметра. Если знак «=» присутствует, то удаление произойдёт только если данному параметру присвоено указанное значение (пусть

даже пустое, "").

Например:

```
[rd|HKCU\Some\Key]
Item1
"Item2"
Item3="Value"
"Item4"=dword:00000001
[rd|]
HKCU\Some\Key\"Item5"=hex:0a,1b,2c
HKCU\Some\Key\Item6
```

Из ключа HKCU\Some\Key будут безусловно удалены параметры Item1, Item2 и Item6, а параметры Item3, Item4 и Item5 — только если содержали указанное значение (и имели тот же тип).

6.5.6) (+) D

Удаление ключа. Из реестра удаляется указанный ключ (или целая ветка).

Ключ в заголовке не может быть опущен. В противном случае секция игнорируется, а в лог заносится сообщение на уровне 2 — неверный синтаксис секции.

6.5.6.1) (+) Если ключи отсутствуют,

происходит безусловное удаление.

Если задан хотя бы один параметр, это воспринимается как условие удаления. Ключ реестра удаляется, только если существуют указанные ключи/параметры (возможно, с указанными значениями).

Ключ в данном контексте задаётся с помощью завершающего обратного слеша.

При этом вполне можно проверить наличие ключей/параметров, не являющимися потомками удаляемого ключа. Для этого можно использовать абсолютные пути.

Например:

```
[rD|HKCU\Some\DeletingKey]
ExistedParam1
"ExistedParam2"=dword:00000000
HKCU\Some\ExistedKey\
HKCU\More\Key\ExistedParam
```

Ключ (или ветка) HKCU\Some\DeletingKey будет удалён, только если: в нём имеется параметр ExistedParam1, в нём имеется параметр ExistedParam2 с указанным значением, имеется ключ HKCU\Some\ExistedKey, имеется параметр ExistedParam в ключе HKCU\More\Key.

6.5.7) (+) n

Переименование параметров. В реестре параметры, указанные в пресете, переименовываются в заданные значения.

Значение не может быть опущено. В противном случае указанный ключ игнорируется, а в лог заносится сообщение на уровне 2 — неверный синтаксис секции.

Имена не должны быть заключены в двойные кавычки (в отличие от синтаксиса r-секций с другими действиями.

Например:

```
[rn|HKCU\Some\Key]
OldItem1=NewItem1
...
```

6.5.8) (–) N

Переименование ключа. В реестре ключ, указанный во втором параметре заголовка, переименовывается. Ключ снабжается полным путём. Новое имя указывается в последнем параметре заголовка. Новое имя может быть снабжено путём, тогда произойдёт не просто переименование, но и перемещение ключа в структуре реестра.

Старый и новый ключи в заголовке не могут быть опущены. В противном случае секция игнорируется, а в лог заносится сообщение на уровне 2 — неверный синтаксис секции.

Если задан хотя бы один параметр, это воспринимается как условие переименования. Ключ реестра переименовывается, только если существуют указанные ключи/параметры (возможно, с указанными значениями).

Ключ в данном контексте задаётся с помощью завершающего обратного слеша.

При этом вполне можно проверить наличие ключей/параметров, не являющимися потомками удаляемого ключа. Для этого можно использовать абсолютные пути.

Например:

```
[rN|HKCU\Some\Old1|New1]
[rN|HKCU\Some\Old2|HKCU\More\New2]
ExistedParam1
"ExistedParam2"=dword:00000000
HKCU\Some\ExistedKey\
HKCU\More\Key\ExistedParam
```

Ключ HKCU\Some\Old1 переименуется безусловно в HKCU\Some\New1. А HKCU\Some\Old2 будет перенесён в ветку HKCU\More под новым именем New2, только если: в нём имеется параметр ExistedParam1, в нём имеется параметр ExistedParam2 с указанным значением, имеется ключ HKCU\Some\ExistedKey, имеется параметр ExistedParam в ключе HKCU\More\Key.

Примеры

Пример №1:

```
[rm|HKCU\Some\Key]
"Item1"="Value1"
"Item2"=dword:00000001
HKEY_CURRENT_USER\Some\Key\"Item3"="same"
HKCU\More\Key\"Item"="wow!!!"
[rm|]
HKEY_CURRENT_USER\Some\Key\"Item4"=hex(7):0a,1b,2c
HKCU\Some\Key\@="hello!"
HKCU\More\Key\@="world!"
```

Обратите внимание, первые три строки в первой секции и первые две строки во второй секции относятся к одному и тому же ключу реестра, хоть и записаны совершенно разными способами. Последние строки обеих секций также манипулируют одним ключом реестра.

7) (–) XML-файлы (в разработке)

Секции для работы с XML-файлами предваряются буквой «**x**»:

```
[x...]
```

Структура XML-файлов может быть очень сложной, но всегда регулярной. То есть всегда удастся однозначно определить "путь" к нужному элементу или его атрибуту.

Обрабатываются лишь корректно сформированные XML-документы. При этом проверка действительности документов не производится, DTD игнорируется, пространства имён ничего не означают.

За основу синтаксиса указания путей принят синтаксис XPath. Для начала следует поддерживать лишь простейшие конструкции.

Каждый ключ секции соответствует одному элементу или атрибуту. Значение записывается через "=":

```
path/element=text
path/element/@attribute=text
```

Программа не осуществляет анализ каких-либо значений, воспринимает и обрабатывает все значения как текст.

7.1) (–) Описание пути (избранное из XPath)

7.1.1) (–) "/"

Корневой элемент. Корнем является не элемент-документ, а сам документ, как это и принято в XPath. Таким образом, в последствие будет возможно работать и с комментариями и инструкциями по обработке вне элемента-документа.

Путь, начинающийся с "/", является абсолютным.

7.1.2) (–) "имя"

Определяет элемент с заданным именем. В XML регистр символов не игнорируется.

7.1.3) (–) "@имя"

Определяет атрибут с заданным именем.

7.1.4) (–) ".."

Псевдо-элемент пути, адресующий родительский элемент.

7.1.5) (–) путь

Путь к произвольному элементу/атрибуту может быть задан с помощью последовательного перечисления его предков слева направо. Имя каждого предка отделяется прямым слешем.

У атрибутов не бывает потомков, поэтому они могут встретиться только в конце пути.

Абсолютный путь применим и в заголовке секции и в именах ключей, а относительный — только в именах ключей, так как нарочно не предусмотрен параметр вроде DefaultElement.

7.2) (–) Заголовок

В заголовке секции как обычно нужно указать файл, над которым осуществляется действие, а также базовый элемент, применяемый, если для конкретного элемента задан относительный путь. Каждый ключ секции соответствует одному элементу или атрибуту. Значение записывается через "=":

7.2.1) (–) [x<действие>|<XML-файл>|<базовый элемент>]

Поиск файла осуществляется по общей схеме с учётом DefaultDirectory и DefaultFile.

Поле "базовый элемент" может содержать только наипростейший XPath-адрес и только абсолютный. Если поле пусто, подразумевается "/", то есть корень дерева XML-документа.

7.3) (–) Действия

В силу того что структура XML-файлов гораздо сложнее структуры INI-файлов, набор действия здесь не столь богат, а смысл одноимённых иногда сильно различается.

7.3.1) (–) r

Обычная перезапись. Ветка документа, начиная от базового элемента уничтожается, а вместо неё создаётся новая ветка на основании ключей секции.

Так как синтаксис этих секций гораздо скуднее синтаксиса XML-файлов, данная операция может привести к потери некоторых данных (например, комментариев и инструкций по обработке).

Если базовый элемент задан, например, как "/" (или просто опущен), такая x-секция, очевидно, приведёт к полному пересозданию документа.

При создании элементов/атрибутов при необходимости создаются и все промежуточные элементы.

7.3.2) (–) m

Обычное слияние. Уже имеющиеся элементы и атрибуты, не упоминаемые в пресете, остаются без изменения. Совпадающие элементы и атрибуты обновляются значениями из пресета. Элементы и атрибуты из пресета, отсутствующие в целевом XML-файле, вставляются в него (после имеющихся).

При создании элементов/атрибутов при необходимости создаются и все промежуточные элементы.

7.3.3) (–) x

Взаимная перезапись (обмен). Содержимое указанных элементов и значения указанных атрибутов в целевом файле и в секции пресета целиком меняются местами.

Если базовый элемент задан, например, как "/" (или просто опущен), такая x-секция, очевидно, приведёт к полному пересозданию документа, причём пресет будет вставлена почти полная опись XML-файла (только элементы и их атрибуты).

При этом никакой оптимизации не производится. То есть количество строк

в полученной x-секции будет равно сумме элементов и атрибутов.

7.3.4) (–) a

Добавление. Из пресета берутся и вставляются только отсутствующие в целевом XML-файле элементы и атрибуты. Новые элементы и атрибуты вставляются после имеющихся.

При создании элементов/атрибутов при необходимости создаются и все промежуточные элементы.

7.3.5) (–) d

Удаление. Из целевого XML-файла удаляются элементы/атрибуты, указанные в пресете.

Если задано только имя элемента/атриута (без знака «=»), то происходит безусловное удаление. Если знак «=» присутствует, то удаление произойдёт только если данный элемент/атрибут содержит указанное значение (пусть даже пустое).

7.3.6) (–) D

Условное удаление элемента. В целевом XML-файле удаляется элемент, заданный в базовом элементе в заголовке.

Ключи рассматриваются как условия удаления. Если ключи отсутствуют вообще, произойдёт безусловное удаление (то есть действие аналогично [xd.., но для единственного элемента).

Если задан хотя бы один ключ, это воспринимается как условие удаления. Элемент удаляется, только если указанные элементы/атрибуты имеются в целевом файле (возможно, с указанными значениями). При этом вполне можно проверить наличие элементов/атрибутов, не являющимися потомками комментируемого элемента. Для этого можно использовать абсолютные пути или псевдо-элемент пути "..".

Если задано только имя ключа (без знака «=»), то значение элемента/атрибута не учитывается. Если знак «=» присутствует, то учитывается значение (пусть даже пустое).

7.3.7) (–) c<режим>

Комментирование элемента. В целевом XML-файле комментируется указанный элемент целиком. Комментирование заключается в добавлении "<!--" перед открывающим тегом элемента и "-->" — после закрывающего тега.

Комментировать можно только элементы.

Если ключи отсутствуют вообще, секция лишена смысла и игнорируется.

Режимы аналогичны пункту 5.2.7.

7.3.8) (–) C<режим>

Условное комментирование элемента. В целевом XML-файле комментируется элемент, заданный в базовом элементе в заголовке. Комментирование заключается в добавлении "<!--" перед открывающим тегом элемента и "-->" — после закрывающего тега.

Ключи рассматриваются как условия комментирования. Если ключи отсут-

ствуют вообще, произойдёт безусловное комментирование (то есть действие аналогично [xc.., но для единственного элемента).

Если задан хотя бы один ключ, это воспринимается как условие комментирования. Элемент комментируется, только если указанные элементы/атрибуты имеются в целевом файле (возможно, с указанными значениями). При этом вполне можно проверить наличие элементов/атрибутов, не являющимися потомками комментируемого элемента. Для этого можно использовать абсолютные пути или псевдо-элемент пути "..".

Если задано только имя ключа (без знака «=»), то значение элемента/атрибута не учитывается. Если знак «=» присутствует, то учитывается значение (пусть даже пустое).

Режимы аналогичны пункту 5.2.7.

7.3.9) (–) n

Переименование. В целевом XML-файле элементы и/или атрибуты, указанные в пресете, переименовываются в заданные значения.

Значение не может быть опущено. В противном случае указанный ключ игнорируется, а в лог заносится сообщение на уровне 2 — неверный синтаксис секции.

```
[xn|some.xml|path/baseelement]
path/element=newname
path/element/@attribute=newname
...
```

Примеры

Пример №1

8) (±) Файловые операции

Секции для работы с файлами предваряются буквой «**f**».

Файловые операции (f-секции): особые секции с особым синтаксисом записи "ключей". В таких секциях можно определить различные действия над файлами: удаления, перемещение, копирование и т.д.

Файловые секции позволяют работать не только с файлами, но и с папками (практически с тем же синтаксисом).

Так как количество секций ни чем не ограниченно, с помощью данных операций можно запрограммировать любые мыслимые манипуляции с файловой системой.

Для удобства предусмотрены два вида f-секций: общие и конкретизированные. Например:

```
[f|%COMMANDER_PATH%\kITUP\Files\Bars|%COMMANDER_PATH%\Bars]
...
[fC|%COMMANDER_PATH%\kITUP\Files\Bars|%COMMANDER_PATH%\Bars]
...
```

Кроме того, некоторые действия не оперируют "исходной" и "целевой" папками, им достаточно указания одной, "рабочей" папки. Такие секции и действия назовём стационарными, а остальные — направленными.

Внимание. Если в общей направленной f-секции задано стационарное действие, то "рабочей" папкой будет считаться "целевая" папка! Использовать направленные действия в стационарных f-секциях запрещено.

8.1) (±) Заголовок

В заголовке секции нужно указать пути к папкам источника и целевой папки. Здесь пути обрабатываются по общей схеме, то есть могут быть относительными (принимается во внимание DefaultDirectory).

8.1.1) (+) [f|<исходная_папка>|<целевая_папка>]

Общая направленная f-секция. Внутри такой секции могут описываться любые манипуляции с файлами, где участвуют две папки. При этом в каждой строке нужно будет отдельно указывать, какое действие нужно выполнить.

Внутри таких секций используется особый синтаксис ключей (без "=").

8.1.2) (+) [f|<рабочая_папка>]

Общая стационарная f-секция. Внутри такой секции могут описываться любые манипуляции с файлами, где участвует одна папка. При этом в каждой строке нужно будет отдельно указывать, какое действие нужно выполнить.

Внутри таких секций используется особый синтаксис ключей (без "=").

8.1.3) (+) [f<действие>|<исходная_папка>|<целевая_папка>]

Конкретизированная направленная f-секция. Действие задаётся однажды для всех файлов, перечисленных внутри секции.

Внутри таких секций используется обычный ini-синтаксис. Но в ряде случаев не придётся или будет запрещено указывать что-либо после символа

"=", да и сам этот символ.

8.1.4) (+) [f<действие>|<рабочая_папка>]

Конкретизированная стационарная f-секция. Действие задаётся однажды для всех файлов, перечисленных внутри секции.

Внутри таких секций используется обычный ini-синтаксис. Но в ряде случаев не придётся или будет запрещено указывать что-либо после символа "=", да и сам этот символ.

Некоторые типы действий могут использоваться в разных видах секций: и в направленных, и в стационарных. Обобщая, можно сказать, что стационарная секция — это направленная секция, в которой исходная и целевая папка совпадают.

8.1.5) (+) Папки по умолчанию

Как исходную, так и целевую (или рабочую) папку можно опустить. В этом случае можно полагаться либо на DefaultDirectory, либо задавать пути с соответствующей стороны абсолютно.

Если имена файлов в ключах снабжены путями, папки, указанные в заголовке секции будут игнорироваться. Более того, они вообще могут быть опущены в таком случае.

Это относится и к общим, и к конкретизированным f-секциям.

8.1.5.1) (+) [f<возможно,действие>|]

Рабочая папка равна текущему содержимому DefaultDirectory.

8.1.5.2) (+) [f<возможно,действие>|<исходная_папка>|]

Целевая папка равна текущему содержимому DefaultDirectory.

8.1.5.3) (+) [f<возможно,действие>||<целевая_папка>]

Исходная папка равна текущему содержимому DefaultDirectory.

8.1.5.4) (+) [f<возможно,действие>||]

Исходная и целевая папки равны текущему содержимому DefaultDirectory.

8.2) (±) Генерация имён и переменные окружения

В именах исходных файлов можно использовать не только абсолютные и относительные пути, но и маски.

Маски позволяют в одной строке описать манипуляцию над множеством файлов.

8.2.1) (+) ?

Соответствует любому одиночному символу.

8.2.2) (+) *

Соответствует любому (в том числе нулевому) количеству любых символов.

Маски можно применять лишь при указании исходных файлов, но не целевых. Это значит, что все конструкции определённым образом видоизменяются, если в них применяются маски. Обычно это означает запрет на указание целевого имени, например:

```
[f|C:\Src|D:\Dest]
*|C|
[fm|C:\Src|D:\Dest]
oldarch.r??
[fd|C:\Sources|D:\Headers]
*.cpp|*.h
```

Путь и имя файла/папки может содержать любое количество переменных окружения в любом месте (даже включая обратный слеш — он будет правильно учтён).

8.2.3) (+) %EXISTING_VARIABLE%

Заменяется на значение существующей переменной окружения. Переменные окружения, как обычно, заключаются в символы "%". Замена производится за один проход.

Регистр в переменных не учитывается.

8.2.4) (–) %NOT_EXISTING_VARIABLE%

Если указанная переменная не зарегистрирована в окружении, то вместо конструкции вставляется пустая строка.

8.2.5) (–) %%

Сочетание "%%" заменяется на "%" и лишается особого смысла.

Переменные окружения берутся из актуального окружения программы — анализатора up-файлов. Если требуется задать какие-либо дополнительные пользовательские/системные переменные, это нужно сделать в вызывающем процессе ДО запуска программы-анализатора. Программа-анализатор не реагирует на системный сигнал об изменении переменных окружения. В качестве альтернативы в языке UP предусмотрены конструкции для модификации окружения.

8.3) (+) Ключи

Синтаксис ключей f-секции зависит от её вида: общая или конкретизированная.

В ключах вместо разделителя "=" придётся использовать "|", так как символ "=" допустим в именах файлов и папок.

8.3.1) (+) Общие направленные действия

В первом и последнем поле должны фигурировать объекты одинакового рода: либо два файла, либо две папки.

8.3.1.1) (+) <имя1>|<действие>|<имя2>

8.3.1.2) (+) <папка1>\|<действие>|<папка2>\

Крайнее левое поле строки указывается имя исходного файла (ищется в исходной папке), крайнее правое — целевого (ищется в целевой папке).

Символ, определяющий действие над указанными файлами, указывается между двумя разделителями, что обеспечивает наглядность, простоту парсинга и расширяемость.

Имена файлов могут как совпадать, так и нет. То есть, например, при копировании файла можно его заодно переименовать.

Если имя заканчивается на "\", значит речь идёт о папке. Слеш должен присутствовать и в исходном, и в целевом имени.

Смысл действий для папок в целом совпадает со смыслом аналогичных действий над файлами. Даже термин "файл" обычно относится и к папкам тоже. Где это не так, будет упомянуто отдельно.

8.3.1.3) (+) <маска>|<действие>|

8.3.1.4) (+) <маска>|<действие>|<папка>

Если в исходном имени использована маска, то указывать целевое имя запрещено. Маски применимы только к файлам. Можно указать папку справа. В таком случае целевая папка, указанная в заголовке, игнорируется и может быть опущена.

8.3.2) (+) Общие стационарные действия

Стационарные действия оперируют одним файлом/папкой, поэтому в строке могут присутствовать, как оба имени, так и только одно из них. При этом действие применяется к отдельными файлам/папкам независимо, а значит, одно имя может задавать файл, а другое — папку.

Таким образом, возможны следующие варианты:

8.3.2.1) (+) <имя1>|<действие>|

8.3.2.2) (+) <папка1>\|<действие>|

8.3.2.3) (+) <имя1>|<действие>|<имя2>

8.3.2.4) (+) <имя1>|<действие>|<папка2>

8.3.2.5) (+) <папка1>\|<действие>|<имя2>

8.3.2.6) (+) <папка1>\|<действие>|<папка2>

8.3.2.7) (+) |<действие>|<имя2>

8.3.2.8) (+) |<действие>|<папка2>

8.3.2.9) (+) <маска1>|<маска2>

Так как файлы слева и справа обрабатываются независимо, маски могут присутствовать с обеих сторон или только с одной из сторон.

8.3.3) (+) Конкретизированные направленные действия

Так как, в заголовке уже указано выполняемое действие, а для многих действий имя целевого файла можно опустить, оставляя подразумеваемым, такие f-секции являются самыми компактными.

8.3.3.1) (+) <имя>

8.3.3.2) (+) <имя>

Такие строки функционально совпадают со строками в общей f-секции с опущенным целевым именем, но символ "|" тоже должен быть опущен.

Если это допустимо для конкретного действия, целевое имя совпадает с исходными именем (но, возможно, имеет другой путь).

8.3.3.3) (+) <имя>|<имя>

8.3.3.4) (+) <имя>\|<имя>

Такие строки, функционально совпадают со строками в общей f-секции с

заданным целевым именем, но символ "|" тоже должен быть опущен.

Если это имеет смысл для конкретного действия, второе имя задаёт результирующее имя заданного объекта.

8.3.3.5) (+) <маска1>

8.3.3.6) (+) <маска1>|<папка>\

Если в исходном имени использована маска, то указывать целевое имя запрещено. Маски применимы только к файлам. Можно указать папку справа. В таком случае целевая папка, указанная в заголовке, игнорируется и может быть опущена.

8.3.4) (+) Конкретизированные стационарные действия

Некоторые виды стационарных действий подразумевают, а другие не подразумевают указания кроме имени исходного файла и некой дополнительной информации. Таким образом, возможны два варианта синтаксиса.

8.3.4.1) (+) <имя1>

8.3.4.2) (+) <имя1>\

Если действие не требует указание дополнительной информации для выполнения, то должно быть указано только имя файла, должен быть опущен символ "|".

8.3.4.3) (+) <имя1>|<информация>

8.3.4.4) (+) <имя1>\|<информация>

Если действие подразумевает использование дополнительной информации, она должна быть указана во втором поле через разделитель "|".

8.4) (±) Действия

8.4.1) (+) C

Копирование с перезаписью. Направленное действие.

8.4.1.1) (+) В общей секции

```
[f|source|target]
file|C|file
folder\|C|folder\
```

Если целевой файл не указан, исходный файл копируется под своим именем.

8.4.1.2) (+) В конкретизированной секции

```
[fC|source|target]
file
folder\
file|newname
folder\|newname\
```

Второе поле позволяет задать новое имя, под которым файл будет скопирован в целевую папку.

8.4.2) (+) c

Копирование без перезаписи. Направленное действие.

8.4.2.1) (+) В общей секции

```
[f|source|target]
file|c|file
```

```
folder\|c|folder\
```

Если целевой файл не указан, исходный файл копируется под своим именем.

8.4.2.2) (+) В конкретизированной секции

```
[fc|source|target]
file
folder\
file|newname
folder\|newname\
```

Второе поле позволяет задать новое имя, под которым файл будет скопирован в целевую папку.

8.4.3) (+) M

Перемещение с перезаписью. Направленное действие.

8.4.3.1) (+) В общей секции

```
[f|source|target]
file|M|file
folder\|M|folder\
```

Если целевой файл не указан, исходный файл перемешается под своим именем.

8.4.3.2) (+) В конкретизированной секции

```
[fM|source|target]
file
folder\
file|newname
folder\|newname\
```

Второе поле позволяет задать новое имя, под которым файл будет перемещен в целевую папку.

8.4.4) (+) m

Перемещение без перезаписи. Направленное действие.

Если целевой файл существовал, ничего не происходит, исходный файл не удаляется.

8.4.4.1) (+) В общей секции

```
[f|source|target]
file|m|file
folder\|m|folder\
```

Если целевой файл не указан, исходный файл перемешается под своим именем.

8.4.4.2) (+) В конкретизированной секции

```
[fm|source|target]
file
folder\
file|newname
folder\|newname\
```

Второе поле позволяет задать новое имя, под которым файл будет перемещен в целевую папку.

8.4.5) (+) d

Удаление файла или папки. Стационарное действие.

Папка удаляется со всем содержимым.

8.4.5.1) (+) В общей секции

```
[f|source|target]
file|d|
folder\|d|
file|d|file
|d|file
```

Операция распространяется на оба файла, если они указаны (то есть отсутствие файла с любой из сторон не является ошибкой).

Так как в данном действии "исходный" и "целевой" файлы никак не связаны, вполне возможно в одной строке указать файл и папку.

```
folder\|d|file
```

8.4.5.2) (+) В конкретизированной секции

```
[fd|work]
file
folder\
```

8.4.6) (+) e

Создание пустого файла, только если его нет. Стационарное действие.

Если указана папка, а она уже существует, то ничего не происходит. Если отсутствует, то создаётся.

8.4.6.1) (+) В общей секции

```
[f|source|target]
file|e|
folder\|e|
file|e|file
|e|file
```

Операция распространяется на оба файла, если они указаны (то есть отсутствие файла с любой из сторон не является ошибкой).

Так как в данном действии "исходный" и "целевой" файлы никак не связаны, вполне возможно в одной строке указать файл и папку.

```
folder\|e|file
```

8.4.6.2) (+) В конкретизированной секции

```
[fe|work]
file
folder\
```

8.4.7) (+) E

Создание пустого файла или урезание до нуля. Стационарное действие.

Если указана папка, а она уже существует, то всё её содержимое удаляется. Если отсутствует, то создаётся.

8.4.7.1) (+) В общей секции

```
[f|source|target]
file|E|
folder\|E|
file|E|file
|E|file
```

Операция распространяется на оба файла, если они указаны (то есть отсутствие файла с любой из сторон не является ошибкой).

Так как в данном действии "исходный" и "целевой" файлы никак не связа-

ны, вполне возможно в одной строке указать файл и папку.

```
folder\|E|file
```

8.4.7.2) (+) В конкретизированной секции

```
[fE|work]
file
folder\
```

8.4.8) (+) a

Дозапись. Направленное действие.

Для файлов: дозапись исходного файла в конец целевого файла, только если целевой файл существует. Нельзя опускать имя целевого файла.

Для папок: копирование содержимого исходной папки в целевую папку без замены файлов только если целевая папка существует. Нельзя опускать имена папок, но можно задать их в виде, например, ".\", чтобы действие относилось к папкам, указанным в заголовке.

8.4.8.1) (+) В общей секции

```
[f|source|target]
file|a|file
folder\|a|folder\
```

8.4.8.2) (+) В конкретизированной секции

```
[fa|source|target]
file|file
folder\|folder\
```

8.4.9) (+) A

Дозапись или создание. Направленное действие.

Для файлов: дозапись исходного файла в конец целевого файла. Если его нет, то происходит копирование. Нельзя опускать имя целевого файла.

Для папок: копирование содержимого исходной папки в целевую папку без замены файлов. Если целевая папка не существовала, она будет создана, т. е действие сводится к копированию. Нельзя опускать имена папок, но можно задать их в виде, например, ".\", чтобы действие относилось к папкам, указанным в заголовке.

8.4.9.1) (+) В общей секции

```
[f|source|target]
file|A|file
folder\|A|folder\
```

8.4.9.2) (+) В конкретизированной секции

```
[fA|source|target]
file|file
folder\|folder\
```

8.4.10) (+) u

Обновление. Направленное действие.

Для файлов: копирование файла с заменой, только если целевой файл существовал и его время модификации меньше времени модификации исходного файла.

Для папок: применение аналогичного действия для каждого файла и папки, вложенных в исходную папку, рекурсивно.

8.4.10.1) (+) В общей секции

```
[f|source|target]
file|u|file
folder\|u|folder\
```

8.4.10.2) (+) В конкретизированной секции

```
[fu|source|target]
file|file
folder\|folder\
```

8.4.11) (+) U

Обновление или копирование. Направленное действие.

Для файлов: копирование файла с заменой, только если время модификации целевого файла меньше времени модификации исходного файла. А если файл не существовал, происходит копирование.

Для папок: применение аналогичного действия для каждого файла и папки, вложенных в исходную папку, рекурсивно.

8.4.11.1) (+) В общей секции

```
[f|source|target]
file|U|file
folder\|U|folder\
```

8.4.11.2) (+) В конкретизированной секции

```
[fU|source|target]
file|file
folder\|folder\
```

Примеры

Пример №1

```
[f|%COMMANDER_PATH%\kITUP\Files\Bars|%COMMANDER_PATH%]
big_default.bar|C|default.bar
program.bar|C|
|d|*.br2
```

Особый файл основной панели Total Commander заменяет имеющийся (меняя имя). Файл другой панели копируется под своим именем. Затем удаляются все файлы кеша.

9) (±) Процессы (в разработке)

Секции для работы с процессами предваряются буквой «**p**»:

```
[peq|c:\totalcmd\totalcmd.exe]
```

Параметры позволяют уточнить форму указания процесса и действие, именно в этом порядке. В отличие от других видов секций p-секции имеют гибкий синтаксис заголовков, зависящий от конкретного действия. Кроме того, не все сочетания действий и форм указания на процесс допустимы.

Одна p-секция относится к одному процессу (или к ряду процессов одной программы).

9.1) (±) Заголовок

Некоторые действия (например, посылка сообщения окну) могут требовать задать значительный объём информации, причем в довольно разнообразных форматах.

В заголовке указывается только первично необходимая информация, а все дополнительные данные передаются через ключи со строго определёнными именами.

9.1.1) (+) [p<форма_идентификации><действие>|<идентификатор_процесса>]

Здесь "идентификатор процесса" не означает "PID" (в общем случае). Формат идентификатора зависит от буквы "форма идентификации" процесса.

9.1.2) (+) [p<форма_идентификации><действие>|<идентификатор_процесса>|<параметры_действия>]

Дополнительные параметры может потребоваться передать, например, программе при её запуске. Для каждого отдельного действия указано, как интерпретируется это поле.

9.1.3) (–) [p<форма_идентификации><действие>|<идентификатор_процесса>|<параметры_идентификации>]

Некоторые формы идентификации могут потребовать уточнения некоторых данных.

9.1.4) (–) [p<форма_идентификации><действие>|<идентификатор_процесса>|<параметры_идентификации>|<параметры_действия>]

Если требуются оба типа дополнительных параметров, то они следуют в указанном порядке.

9.2) (±) Идентификация целевого процесса

9.2.1) (–) i

По идентификатору (Process ID).

Идентификатор: PID, десятичное число.

Параметры: нет.

```
[piq|1234]
```

9.2.2) (–) n

По внутреннему имени процесса.

Идентификатор: внутреннее имя процесса, строка.

Параметры: ApplyTo.

```
[pnt|TOTALCMD.EXE]
```

9.2.3) (+) e

По имени исполняемого файла.

Идентификатор: имя исполняемого файла, строка.

Параметры: ApplyTo.

```
[pee|c:\programs\myprog.exe]
```

9.2.4) (–) p

Непосредственный родитель программы.

Идентификатор: нет (должен быть пустым).

Параметры: нет.

```
[ppr|]
```

9.2.5) (–) a

Предок по отношению к программе.

Идентификатор: внутреннее имя процесса, строка. Не может быть опущен.

Параметры: ApplyTo.

```
[paq|totalcmd.exe]
ApplyTo=1
```

9.2.6) (+) s

Особый вид обращения к процессу — к среде Windows. Запуск осуществляется с помощью ShellExecute.

Третья буква в этом случае задаёт действие, передаваемое функции ShellExecute, а не действие, описанное в 9.3.

Идентификатор: имя файла, возможно с путём (обрабатываются переменные среды). Не может быть опущен.

В третьем поле заголовка можно задать параметры (например, параметры запуска). Переменные среды также обрабатываются.

В ShellExecute поддерживаются следующие действия.

9.2.6.1) (+) e

Редактирование (Edit). Запускает редактор и открывает документ для редактирования (отсутствует команда Edit в контекстном меню). Если файл не является документом, вызов не удастся.

9.2.6.2) (+) x

Обзор (Explore). Открывает в проводнике папку с заданным путём.

9.2.6.3) (+) f

Поиск. Инициирует начало поиска проводником в указанном каталоге.

9.2.6.4) (+) o

Открытие (Open). Выполняет подразумеваемое действие для файла или папки. По сути эмулируется Двойной ЛКМ (или Enter) в проводнике.

9.2.6.5) (+) p

Печать (Print). Печатает документ принтером по умолчанию. Если файл не является документом (отсутствует команда Print в контекстном меню), вызов не удастся.

9.2.6.6) (+) s

Запуск (Start). Выполняется действие по умолчанию, если оно задано в реестре. Иначе, выполняется действие Open. Если не задано действие по умолчанию и отсутствует действие Open, выполняется первое действие из списка действий в реестре.

```
[pss|calc.exe]
[pso|%COMMANDER_PATH%\TOTALCMD.CHM]
[psp|win.ini]
[pss|notepad.exe|%COMMANDER_PATH%\Hist_rus.txt]
```

9.3) (±) Действия

9.3.1) (+) q

Послать сигнал завершения.

Параметры: Wait.

9.3.2) (+) t

Завершить принудительно (снять задачу).

Параметры: Wait.

9.3.3) (+) e

Запустить.

Поддерживаются параметры запуска: передаются как аргументы.

Параметры: Wait, WorkDir, Params, ...

9.3.4) (+) r

Перезапустить (то же, что q+e).

Поддерживаются параметры запуска: передаются как аргументы при запуске программы после завершения.

Параметры: Wait (относится к поддействию q), WorkDir, Params, ...

9.3.5) m

послать сообщение

9.3.6) k

послать сочетании клавиш

9.3.7) c

кликнуть мышью

...

9.4) (–) Именованные параметры

9.4.1) (–) ApplyTo

Количество одноимённых:

9.4.1.1) пусто, по умолчанию

к первому найденному.

9.4.1.2) десятичное число

к указанному количеству первых найденных.

9.4.1.3) *

ко всем.

9.4.2) (–) Wait

Ожидание результата действия (например, закрытия процесса) или заданное максимальное время. Если не задан, не переопределят глобальное значение (CloseProcessWait).

9.4.2.1) пусто

не ждать.

9.4.2.2) десятичное число

максимальное время ожидания в миллисекундах.

9.4.2.3) *

ждать бесконечно.

9.4.3) (–) WorkDir

Задаёт рабочую папку для запускаемой программы.

9.4.3.1) пусто, по умолчанию

совпадает с папкой запускаемой программы.

9.4.3.2) строка

определяет рабочую папку. Никаких проверок на корректность не производится.

9.4.4) (–) Params

Задаёт строку с параметрами для запускаемой программы. Использовать этот способ задания параметров запуска не обязательно, так как их можно указать прямо в заголовке после имени программа. Если использованы оба способа задания параметров, данная строка дописывается в конец.

9.4.4.1) пусто, по умолчанию

Пустая строка.

9.4.4.2) строка

Дописывается в конец команды запуска (строки-идентификатора).

...

Примеры

Пример №1

10) (±) Метасекции

В рамках одного пресета вполне может потребоваться выполнить определенные действия только если некоторое условие соблюдается. В ряде случаев такую проверку может выполнить и сам пользователь. Тогда можно просто создать два пресета.

Однако, например, для тоглинга необходимы средства выбора нужных действий прямо в пресете.

Очевидно, проверить может потребоваться что угодно: наличие или отсутствие ключа в ini, в реестре или в xml, наличие или отсутствие файла или папки, выполнение определенной программы... К тому же, условия могут быть составными.

Таким образом требуется контейнерная конструкция. При этом подобная же конструкция может использоваться для организации циклов и даже вызова "процедур"...

10.1) (–) Общие конструкции

Для группировки секций и поддержки вложенных конструкций предусмотрено понятие блока.

10.1.1) (–) Блок

```
[{]
секции для выполнения
[}]
```

Блок может быть вырожденным, то есть содержать одну секцию. Тогда ограничители [{] и [}] можно опустить, хотя это не рекомендуется.

10.1.2) (–) Объявление процедуры

```
[#<НомерПроцедуры>]
<блок>
```

Совершенно не важно содержимое конструкции. В частности, в "процедуру" можно заключить набор часто проверяемых условий, а не только действий. Интерпретация содержимого процедуры зависит от места её применения.

Номер процедуры — целое число, возможно отрицательное. Рекомендуется для обычных процедур задавать отрицательные номера, а для блоков условий — положительные. В этом случае логические выражения будут выглядеть более понятно.

...

10.2) (±) Служебные операторы

10.2.1) (–) [#apply|<опции>|<up-файл>]

Применяет указанный пресет. При этом нужно уточнить многие моменты с помощью флагов в опциях. Опции — это десятичное число — массив флагов. Отсутствие значения подразумевает 0. По умолчанию вызываемый пресет не воспринимается как самостоятельный. То есть не обрабатываются его секции Configuration и Pre/PostActions. А значит, переменная DefaultDirectory и ей подобные сохраняют свое значение, полученное для вызывающего пресета. Кроме того, никакие явные или неявные перемен-

ные не покидают пределов дочернего пресета.

Флаги:

10.2.1.1) 1 — не возвращаться к выполнению текущего пресета.

То есть безусловная передача управления на указанный пресет. Этот флаг действует только на текущий уровень обработки пресета. То есть глобальная последовательность выполнения не прерывается.

10.2.1.2) 2 — обработать секцию Configuration.

То есть извлечь и применять указанные там переменные.

10.2.1.3) 4 — исполнить секцию PreActions

самого пресета.

10.2.1.4) 8 — исполнить секцию PostActions

самого пресета.

10.2.1.5) 16 — разделить переменные.

То есть явные и неявные переменные, заданные или измененные в дочернем пресете, могут использоваться в вызывающем пресете и наоборот.

...

10.2.2) (+) [#include|<up-файл>]

Вставляет все секции указанного пресета, кроме Configuration, PreActions и PostActions, в текущее место.

Обратите внимание на то, что данное действие реально просто вставляет содержимое одного пресета в другой, т. е. вставляемый пресет не рассматривается как самостоятельный. Это означает, например, что действие #stop внутри вставленного пресета повлияет на результирующий составной пресет. Если требуется другое поведение, применяется конструкция #apply с соответствующими опциями.

10.2.3) (+) [#wait|<миллисекунды>]

Остановить применение пресета на заданное время.

10.2.4) (+) [#stop]

Прервать применение пресета. Прерывается только текущий пресет, то есть глобальная последовательность не прерывается. Имеет смысл в условных конструкциях.

10.2.5) (–) [#abort]

Прервать применение пресета. Влияет на всю последовательность вызовов, то есть безусловно завершает текущую обработку программы. Однако другие пресеты, переданные в параметре /a будут применены, как обычно.

10.2.6) (–) [#log|<текст>]

Внести запись в журнал на пользовательском уровне.

10.2.7) (–) [#store|<up-файл>]

```
[#store|<up-файл>]
<блок>
```

Запись следующей секции (или нескольких секций в операторных

скобках) в указанный пресет. Служит для программного формирования нового пресета.

...

10.2.8) (–) [#elevate|<уровень>]

Запросить особые привилегии. Можно как повышать, так и понижать текущий уровень привилегий.

Применимы следующие уровни:

10.2.8.1) current (по умолчанию, если опущено)

Используется токен безопасности, переданный родительским процессом. То есть уровень доступа анализатора возвращаться к значению, имевшемуся в момент его запуска.

10.2.8.2) user

Если анализатор обладает правами администратора, то понизить уровень доступа до обычного пользователя.

10.2.8.3) highest

Получить максимально возможный уровень доступа.

10.2.8.4) admin

Потребовать права администратора (только в этом случае сработает UAC).

Данная конструкция может употребляться сколько угодно раз в одном пресете. Даже после ограничения уровня доступа до "User", можно получить обратно имевшиеся в момент запуска анализатора права.

10.3) (–) Алгоритмические операторы

10.3.1) (–) Условная конструкция

10.3.1.1) в общем случае

```
[#if]
<блок условия>
[#then]
<блок если да>
[#else]
<блок если нет>
```

Если не требуется ничего выполнять при несоблюдении условий, то конструкция упрощается:

10.3.1.2) без else-блока

```
[#if]
<блок условия>
[#then]
<блок если да>
```

10.3.1.3) замена блоков процедурами

```
[#if|<номер_процедуры>]
```

и/или

```
[#then|<номер_процедуры>]
```

и/или

```
[#else|<номер_процедуры>]
```

Каждая из частей конструкции (независимо) может быть вынесена в

отдельную процедуру. В этом случае данный набор секций получит целочисленный номер-идентификатор. Для ссылки на него после ключевого слова конструкции нужно поставить вертикальную черту, а за ней номер процедуры:

Например:

```
[#if|1]
[#then|-1]
[#else|-2]
```

Естественно, использовать процедуры или нет, решает составитель пресета. Нет никакой разницы, как заданы блоки.

10.3.1.4) однострочная запись

Допустимой формой записи является однострочная, при которой конструкции #then и #else не используются, а номера процедур указываются в заголовке #if в качестве параметров:

```
[#if|1|-1|-2]
```

Данная форма менее наглядна, поэтому не рекомендуется к применению.

Такая форма применима, очевидно, только при задании всех частей конструкции в виде ссылок на процедуры (логическое значение условия может быть задано непосредственно).

10.3.1.5) передача логического значения непосредственно

Блок условия может вырождаться в непосредственное логическое значение. Такие значения могут быть получены, например, после закрытия диалога-запроса (см. 14). Пользователь и сам может создавать переменные (см. 17). Значение переменной ложно, если оно пусто или равно «0», и истинно в любом другом случае.

Например:

```
[dm|Вопрос|Вы согласны?|YN|Y]
[#if|_ButtonYes|-1]
```

10.3.2) (–) Цикл со счетчиком

Здесь не ставится целью предоставить некую переменную, изменяющую своё значение с каждой итерацией. Данная конструкция лишь обеспечивает выполнение действий несколько раз.

10.3.2.1) в общем случае

```
[#for|<ЧислоПовторов>]
<блок>
```

10.3.2.2) замена блока процедурой

```
[#for|<ЧислоПовторов>|<НомерПроцедуры>]
```

10.3.3) (–) Цикл с предусловием

Данная конструкция позволяет полностью контролировать порядок и количество итераций.

10.3.3.1) в общем случае

```
[#while]
<блок_условия>
[#do]
<блок>
```

10.3.3.2) замена блоков процедурами

```
[#while|<НомерПроцедурыУсловия>]
```

и/или

```
[#do|<номер_процедуры>]
```

Каждая из частей конструкции (независимо) может быть вынесена в отдельную процедуру. В этом случае данный набор секций получит целочисленный номер-идентификатор. Для ссылки на него после ключевого слова конструкции нужно поставить вертикальную черту, а за ней номер процедуры.

10.3.4) (–) Цикл с постусловием

Для удобства ключевые слова циклов не пересекаются.

10.3.4.1) в общем случае

```
[#repeat]
<блок>
[#until]
<блок_условия>
```

10.3.4.2) замена блоков процедурами

```
[#repeat|<НомерПроцедуры>]
```

и/или

```
[#until|<НомерПроцедурыУсловия>]
```

Каждая из частей конструкции (независимо) может быть вынесена в отдельную процедуру. В этом случае данный набор секций получит целочисленный номер-идентификатор. Для ссылки на него после ключевого слова конструкции нужно поставить вертикальную черту, а за ней номер процедуры.

Если и только если использовать процедуры, появляется возможность составлять логические выражения для условий.

10.4) (–) Конструкия use/unuse

Нередко приходиться применять несколько манипуляций в одном и том же файле. В первую очередь имеется в виду работа с INI-, XML-файлами.

Свойство DefaultFile помогает немного сократить объем кода при объявлении секций. Но что если в одном пресете нужно выполнить множество манипуляций с множеством файлов? Кроме того, что в заголовке каждой секции придется повторять имя обрабатываемого файла, существует более серьезная проблема: для выполнения каждой отдельной манипуляции происходит открытие файла, чтение данных, построение программной модели, внесение изменений и сохранение в файл.

Было бы хорошо в подобном случае открыть файл однажды, произвести все манипуляции над программной моделью файла и только потом сохранить.

Конструкция use позволяет открыть указанный файл и построить его программную модель. Все соответствующие секции внутри конструкции манипулируют программной моделью, но не взаимодействует с самими файлами. Таким образом в определённых сценариях достигается огромное увеличение скорости применения пресеты.

Синтаксис конструкции на примере INI-файла:

```
[#use|i|usercmd.ini]
...
[#unuse|usercmd.ini]
```

10.4.1) (–) [#use|<тип>|<файл>]

Открывающая секция.

Тип определяет, какую программную модель следует построить:

10.4.1.1) i

Модель для INI-файла.

10.4.1.2) I

Модель для INI-файла с поддержкой RedirectSection.

10.4.1.3) ic

Модель для INI-файла, без игнорирования комментариев.

10.4.1.4) Ic

Модель для INI-файла, без игнорирования комментариев и с поддержкой RedirectSection.

10.4.1.5) x

Модель для XML-файла.

...

10.4.2) (–) [#unuse|<файл>]

Закрывающая секция.

Очевидно, что нельзя открыть один файл в нескольких режимах (например INI и текст). Поэтому в закрывающей секции и не указывается тип программной модели.

Между открывающей и закрывающей секциями обычно располагаются секции, модифицирующие открытый файл. Хотя вполне допустимо использовать другие секции внутри конструкции, даже того же типа, что и подразумеваемые в данной конструкции, но с другими файлами. Автор пресета всегда явно указывает, что используется уже открытый файл. Однако нельзя обратиться к файлу по явному пути, если он уже открыт. В этом случае секция игнорируются, а в журнал заносится сообщение на уровне ?.

10.4.3) Если используется только один уровень, то:

Файл становится DefaultFile, а значит может (и это рекомендуется) опускаться во вложенных секциях.

Не обязательно указывать имя файла в закрывающей секции (в этом случае закрывается последний открытый файл).

10.4.4) Можно вкладывать конструкции друг в друга, даже пересекать их.

Если конструкции use вкладываются друг в друга, то открытые файлы получают псевдоимена вида ":номер". Файлы нумеруются в порядке открытия, а не в порядке вложенности.

Псевдоимя ":0" указывает на файл DefaultFile и может быть опущено (хотя в многоуровневой конструкции это не рекомендуется, так как снижает наглядность). Остальные файлы получают имена ":1", ":2" и так далее.

Во внутреннем списке программные модели индексируются по полному пути, поэтому и при открытии, и при закрытии можно указывать путь лю-

бым способом — он будет разрешён.

Примеры

Пример №1

Применение конструкции use для массовой модификации ключей разных секций в INI-файле без переоткрытия.

```
[#use|i|usercmd.ini]
 [im||em_BackupSelected]
 menu=Создать резервные копии выделенных файлов/папок
 [im||em_BAR_Addon]
 menu=Панель дополнительных кнопок
 [im||em_BAR_Editors]
 menu=Редакторы
[#unuse|usercmd.ini]
```

11) (-) Секции условий

При использовании алгоритмических конструкций требуется задавать различные условия: существование файла, наличие ключа в INI-файле, равенство конкретной записи в реестре конкретному значению и т.д. Все эти проверки возможно описать с помощью условных секций.

Синтаксис таких секций, очевидно, должен быть очень похож на соответствующие обычные секции-действия. Хотя не обязательно специально обозначать такие секции каким-то отличительным символом, так как области их применения не пересекаются.

Первая буква имени секции совпадает с первой буквой секций-действий. Затем следуют различные символы, уточняющие смысл запроса. Далее через вертикальные черты указываются нужные параметры.

INI-файлы.

Проверка наличия ключа:

[i?|<файл>|<секция>]

<имя_ключа>

Проверка значения ключа:

[i?|<файл>|<секция>]

<имя_ключа>=<значение>

Проверка существования секции:

[i?|<файл>|<секция>]

То есть пустота секции условия означает, что проверяется сама секция.

Отсутствие секции, ключа или значения проверяется аналогично, но с заменой "?" на "!".

Как и в секциях-действиях, вместо имени секции может стоять шаблон "*", "?" или даже регулярное выражение, начинающиеся с "~".

Реестр.

Проверка наличия ключа:

[r?|<куст\путь\раздел>]

<имя_ключа>

Проверка значения ключа:
[r?|<куст\путь\раздел>]
<имя_ключа>=<значение>

Проверка существования раздела:
[r?|<куст\путь\раздел>]

XML-файлы.
Проверка наличия элемента:
[x?|<файл>|<путь\базовый_элемент>]
<путь\элемент>

Проверка наличия атрибута:
[x?|<файл>|<путь\базовый_элемент>]
<путь\элемент>@<атрибут>

Проверка содержимого элемента:
[x?|<файл>|<путь\базовый_элемент>]
<путь\элемент>=<содержимое>

Проверка значения атрибута:
[x?|<файл>|<путь\базовый_элемент>]
<путь\элемент>@<атрибут>=<содержимое>

Файлы.
Проверка существования файла:
[fe?|<путь\имя_файла>]

Проверка существования папки:
[fe?|<путь\имя_папки>\]
То есть наличие слеша в конце указывает, что интересует папка.

Проверка пустоты файла:
[fz?|<путь\имя_файла>]

Проверка пустоты папки:
[fz?|<путь\имя_папки>]

Проверка, содержит ли файл заданную строку:
[ff?|<путь\имя_файла>|<строка>]

Проверка, равны ли файлы по содержимому:
[fm?|<путь\имя_файла1>|<путь\имя_файла2>]

Проверка, меньше ли размер файла указанной величины:
[fsl?|<путь\имя_файла>|<размер_в_байтах>]

Проверка, больше ли размер файла указанной величины:
[fsg?|<путь\имя_файла>|<размер_в_байтах>]

Проверка, равен ли размер файла указанной величине:
[fse?|<путь\имя_файла>|<размер_в_байтах>]

Проверки размеров папок осуществляются теми же конструкциями, но в конце имени ставится слеш. Размер папки подсчитывается рекурсивно, учитывая скрытые файлы.

Проверка, равно (больше, меньше) ли количество файлов в папке (или количество строк в файле) указанной величине:
[fce(g,l)?|<путь\имя_файла>(\)|<количество>]

Проверка, равно (больше, меньше) ли время последнего изменения файлов:
[fte(g,l)?|<путь\имя_файла>(\)|<путь\имя_файла>(\)]

Проверка, является ли файл символьной ссылкой:
[fl?|<путь\имя_файла>]

Процессы.

11.1) Логические операции

В выражениях поддерживаются круглые скобки для изменения приоритета операций, и в порядке убывания приоритета операции:

11.1.1) !

не

11.1.2) *

и

11.1.3) ^

либо

11.1.4) +

или

Например:

```
[#if|25*(3+!14)]...
```

Примеры

Пример №1

12) (±) Секции Actions, PreActions и PostActions

Секции Pre/PostActions могут встретиться в пресете/в конфиге категории/в конфиге программы. Они служат для определения действий, которые должны быть произведены до/после выполнения действий, заданных пресетом. Действия, объявленные в конфиге категории, выполняются при применении каждого пресета данной категории. Действия, объявленные в конфиге программ, выполняются всегда (если, конечно, это не отменено опциями 2.4–2.6).

12.1) (+) Порядок выполнения секций

12.1.1) (+) PreActions в конфиге программы

Может быть отменено, если использованы опции 2.5 или 2.6. Может быть отменено, если указан соответствующий флаг в опции 2.4.

12.1.2) (+) PreActions в конфиге категории

Может быть отменено, если использована опция 2.6. Может быть отменено, если указан соответствующий флаг в опции 2.4.

12.1.3) (+) PreActions в пресете

Может быть отменено, если указан соответствующий флаг в опции 2.4.

12.1.4) (+) содержимое пресета

На выполнение секций влияет множество факторов. См. главу 10 и пункт 2.3.

12.1.5) (+) PostActions в пресете

Может быть отменено, если указан соответствующий флаг в опции 2.4.

12.1.6) (+) PostActions в конфиге категории

Может быть отменено, если использована опция 2.6. Может быть отменено, если указан соответствующий флаг в опции 2.4.

12.1.7) (+) PostActions в конфиге программы

Может быть отменено, если использованы опции 2.5 или 2.6. Может быть отменено, если указан соответствующий флаг в опции 2.4.

12.2) Содержимое секций PreActions и PostActions

Очевидно, что перечень *непосредственно* доступных действий и операций здесь значительно меньше того, что предоставляется при использовании обычных секции-действий в пресете. Однако синтаксис ключей по возможности очень похож на синтаксис обычных секции-действий.

Хотя типичным применением секций Pre/PostActions является вополнение каких-либо действий с процессами (глава 9), однако, учитывая возможность выполнения процедур (глава 10), данные секции позволяют выполнять любые операции, доступные в основной части пресета.

12.2.1) (–) Универсальный синтаксис

Если и только если секция не содержит ключей, следующая секция рассматривается как её содержимое. Если следующая секция открывает блок секций, то можно описать любое количество любых действий в обыч-

ном синтаксисе. Например:

```
[PostActions]
[pse|calc.exe]
```

12.2.2) (+) Упрощенный синтаксис

На практике действия в этих секциях будут простыми и прямолинейными. Поэтому предусмотрен упрощенный синтаксис, в котором действия записываются в качестве ключей секции Pre/PostActions. При этом запись ключа совпадает с аналогичным заголовком секции, то есть знак «=» не играет специальной роли.

```
[PostActions]
pse|calc.exe
dmi|Калькулятор запущен
```

Секции Actions, PreActions, PostActions позволяют определить наборы различных действий. Секция Actions выполняется в порядке расположения в пресете. Можно определить и несколько таких секций. PreActions и PostActions выполняются соответственно до и после выполнения остальной части пресета. Эти секции могут присутствовать только в единственном экземпляре.

В остальном синтаксис этих трёх секций совпадает.

Каждая строка в таких секциях определяет одно действие. В простейшем случае синтаксис строки совпадает с синтаксисом заголовка требуемого действия за исключением того, что первый символ "|" заменяется на "=". Очевидно, что здесь неудобно использовать действия, основной смысл которых содержится в ключах секции (вроде [i... и т.п.), хотя это и возможно. Для этого нужно указывать каждый ключ через знак "|", а имя ключа отделять от значения знаком "=".

Например:

[Actions]

pss=notepad.exe

im=some.ini|section|param1=value|param2=

Для файловых операций синтаксис следует упростить и не указывать исходную и целевую папки (как в заголовке обычных f-секций), а использовать явные пути.

[PreActions]

fc=d:\1.txt|d:\1.txt.bak

Универсальный синтаксис:

[PostActions]

[{]

[pse|calc.exe]

[dmi|Калькулятор запущен]

[}]

То есть, если и только если секция не содержит ключей, следующая секция рассматривается как её содержимое. Если следующая секция открывает блок секций, то можно описать любое количество любых действий в обычном синтаксисе.

На практике же действия в этих секциях будут простыми и прямолинейными. Поэтому предусмотрен

упрощенный синтаксис:

[PostActions]

pse=calc.exe

dmi=Калькулятор запущен

Примеры

Пример №1

Комбинирование действий в секциях Pre/PostActions:

```
[PostActions]
[{]
 [im|some.ini|SomeSection]
  param1=value1
  param2=value2
 [pse|calc.exe]
 [dmi|Калькулятор запущен]
[}]
```

13) (–) Каркасы (набросок)

Если пользователь решит создать собственный пресет, он может сделать его по аналогии с имеющимися. Однако ничто не помешает ему добавить дополнительные настройки или действия. То есть в общем случае никак не оговаривается, ЧТО делают пресеты данной категории.

Очевидно, нужен некий механизм, который если и не принудит создателя пресета делать адекватный пресет, то хотя бы порекомендует, каким пресет должен быть, какие настройки затрагивать, какие действия совершать. За одно этот механизм позволит выявлять аномалии в пресете (когда данный пресет отличается от остальных, не соответствует рекомендации) и предупреждать об этом конечного пользователя в графическом интерфейсе. Более того, в дальнейшем наличие такого каркаса облегчит создание пресета с нуля.

Итак, очевидно, что информация каркаса должна храниться в конфиге категории:

[Template]

<заголовок_секции_без_скобок>=<комментарий_в_свободной_форме>

и

[Template|<заголовок_секции_без_скобок>]

<ключ>=<комментарий_в_свободной_форме>

Если рекомендованные каркасом ключ или секция отсутствует в пресете, название и комментарий должны отображаться красным шрифтом в графическом интерфейсе, а при проверке адекватности пресета именно такие ключи и секции будут проверяться.

Если требуется указать, какие ключи НЕ должны присутствовать в секции пресета, первым символом комментария нужно поставить "!".

Если некоторый ключ не является обязательным, то есть может пригодиться только при определенных обстоятельствах, то его комментарий должен начинаться с "?". Такие ключ и комментарий будут отображаться зеленым шрифтом в графическом интерфейсе, и не будут учитываться при проверке адекватности пресета.

Таким образом можно будет указать не только структуру пресета, но и описать, какие ключи следует определять в каждой секции. Если некоторое действие является общим для всех пресетов категории, то оно определяется в конфиге категории. Не требуется добавлять в каркас какую-либо информацию о подразумеваемых настройках и общих действиях.

Например, таким может быть каркас для цветовых схем TC:

[Template]

im|wincmd.ini|Colors=одноименная секция полностью, кроме ColorFilter1

im|wincmd.ini|Searches=шаблоны для поддержки цветов

[Template|im|wincmd.ini|Colors]

ColorFilter1=!должен отсутствовать, а в целевом wincmd.ini должен содержать подключение плагина-авторана

[Template|iA|wincmd.ini|Colors]

ColorFilter1=?можно добавить некий заведомо ни чему не соответствующий шаблон для тех случаев, когда плагин-авторан не используется.

Очевидно, не обязательно объявлять в каркасе секцию, если далее будет объявлен конкретный ключ в этой секции. Однако такое объявление не просто бы добавило комментарий к секции в целом, но и показало, что в этой секции могут встретится ключи, которые не будут описаны отдельно, и это не будет считаться аномалией. Если же секция не объявляется, а объявляются лишь некоторые ключи этой секции, значит никаких других ключей в этой секции быть не должно. Лишние ключи приводят к аномалии и выделяются оранжевым цветом.

Примеры

Пример №1

14) (–) Интерактивность (набросок)

Хотя программа задумывалась, как универсальный конфигуратор, выполняющий всевозможные действия с системой в автоматическом режиме, без каких-либо запросов к пользователю, впоследствии стало очевидно, что такие запросы иногда были бы очень полезны. Например, если какой-то пресет приводит к серьезным изменениям в системе и может потребоваться спросить пользователя, уверен ли он, и отменять применение пресета, если нет. Или же просто уместно дать пользователю краткое пояснение, что произойдет или что произошло.

Более того. Можно попросить пользователя ввести какой-то параметр самому. И если уж обеспечивать полную универсальность, то таких диалогов нужно предусмотреть массу.

Конечно, подобные запросы обычно выполняются самой программой, если вообще можно привести аналог. Однако в случае с данной программой это было бы крайне не гибко, потому что составитель пресета не смог бы указать момент появления запроса пользователю. А если учесть, что при использовании алгоритмических конструкций некоторые секции могут не выполнятся, становится очевидно, что показом запросов к пользователю и использованием результатов таких запросов нужно управлять прямо в пресете, с помощью секций особого вида наряду с остальными секциями.

Интерактивность напрямую связана с вопросом локализации. Составитель пресета возможно захочет создать мультиязычный пресет, а значит и диалоги с пользователем должны поддерживать локализованные параметры.

Секция диалога должна быть единственной, но текст, определяемый в параметрах, должен зависеть от текущего языка.

Очевидно, достичь этого можно только с помощью использования неких констант, подобно переменным окружения. Эти константы должны определяться внутри пресета (или в конфиге категории), и их должно быть возможно переопределить с указанием номера языка.

Использование констант не ограничивается только локализацией текста диалогов. Такие константы можно использовать где-угодно, например, в пути к какому-то файлу. Более того. Можно предусмотреть конструкцию для изменения значения "константы" в процессе выполнения пресета! Таким образом удобнее использовать обобщающий термин — псевдоним (см. 17).

14.1) Конструкция

Секции диалогов предваряются буквой «**d**». Затем следует буква, определяющая тип диалога.

Так как существуют большое число типов диалогов с различными наборами предварительно задаваемых параметров и различными видами результатов, предусмотрено несколько вариантов конструкции открытия диалогов-запросов.

14.1.1) Общий синтаксис

В самом общем случае, диалоговое окно может содержать какие-угодно

элементы (в разумных пределах), поэтому требуется универсальный расширяемый синтаксис для описания диалога в пресете.

Заголовок состоит только из типа диалога, а все специфические настройки осуществляются через ключи секции:

```
[d<тип><подтип>*]
<Свойство1>=<Значение1>
<Свойство2>=<Значение2>
...
```

Свойства имеют стандартные имена, перечисленные далее. Ряд свойств встречается во всех типах диалоговых окон, другие — специфичны для конкретных типов.

14.1.2) Упрощенный синтаксис

Очевидно, во всех диалоговых окнах присутствует заголовок и почти во всех диалоговых окнах выводится тот или иной текст. Более того, простейшие диалоговые окна больше ничего и не содержат (кроме различных кнопок).

Таким образом, будет удобно использовать упрощённый синтаксис, при котором самые основные параметры диалогового окна выносятся в заголовок секции, а указывать ключи в секции, возможно, не понадобится вообще:

```
[d<тип><подтип>*|<заголовок окна>|<текст>(|<основные параметры>)*]
```

14.1.3) Смешанный синтаксис

Очевидно, допустимо задавать свойства и в заголовке, и в ключах секции одновременно.

В случае конфликта между значением в заголовке и значением в ключе секции, последнее имеет приоритет.

Например:

```
[dm|заголовок1||Ok]
_Title=заголовок2
_Text=текст
```

Такая конструкция заставит отобразить обычное сообщение с кнопкой «ОК», с заголовком окна «заголовок2» и текстом «текст».

14.2) Свойства диалогов

Здесь перечислены общие свойства, применимые для всех диалогов. Если имеются исключения, об этом будет сказано при описании конкретного типа диалога.

Специфические свойства различных типов диалогов описываются отдельно в разделе, посвященном данному типу диалога.

Все свойства имеют имена, начинающиеся с символа «_» (подчеркивание), для того чтобы отличать стандартные имена свойств от пользовательских имён переменных (псевдонимов).

Некоторые свойства следует задавать для настройки диалогового окна. Другие являются выходными, то есть заполняются самим диалоговым окном при его закрытии. Третьи тоже являются выходными, но могут быть заданы в секции, тем самым определяя значение по умолчанию. Четвёртые являются косвенными, т. е. не влияют ни на вид диалогового окна, ни содержат ка-

кие-либо результаты, но могут использоваться для низкоуровневых манипуляций с диалоговым окном.

14.2.1) _Name

Внутреннее имя диалогового окна. Косвенное свойство.

Позволяет получить доступ к диалоговому окну на низком уровне для выполнения каких-либо манипуляций (в перспективе).

По умолчанию: пусто (низкоуровневое использование невозможно)

14.2.2) _Title

Текст заголовка окна. Входное свойство. Не изменяется.

По умолчанию: пусто

14.2.3) _Text

Текст внутри диалогового окна. Входное свойство. Обычно не изменяется (изменяется только в диалоге ввода текста).

По умолчанию: пусто

14.2.4) _Result

Содержит имя нажатой кнопки. Выходное свойство. Может быть задано ключом в секции, тогда указанная кнопка будет кнопкой по умолчанию.

По умолчанию: не задано (ни одна кнопка не выбрана заранее, а значит не будут сразу срабатывать клавиши «Enter» и «Esc»)

14.2.5) _Timeout

Время до автоматического закрытия диалога в секундах.

Если равно 0, то диалоговое окно не закрывается автоматически.

Если используется автоматическое закрытие диалога, следует позаботится об установке свойства _Result, иначе по истечении времени, если пользователь не нажмёт какую-либо кнопку (или не перейдёт к ней клавишей «Tab»), оно останется пустым.

По умолчанию: 0

...

14.3) Типы диалогов

Далее будет описан каждый тип диалоговых окон. Каждое свойство будет описано, как задаваемое в ключе секции. В заголовке секции будет указаны имена свойств, чтобы показать порядок свойств. Как сказано ранее, задавать свойства можно любым способом.

14.3.1) [dm<иконка>|_Title|_Text|_Buttons|_Result]

Универсальное сообщение-запрос: от простого сообщения с кнопкой «Ок», до вопроса с несколькими кнопками — вариантами ответа.

14.3.1.1) Свойство _Buttons

Определяет набор кнопок диалога. Входное свойство.

Обычно используются стандартные кнопки (встречающиеся в подавляющем большинстве программ). Далее указана буква, применяемая в заголовке секции, затем описание кнопки, а в скобках указано имя переменной (логического типа), которая создаётся для этой кнопки и может быть использован для определения нажатия кнопки.

O — Ok (_ButtonOk)
C — Cancel (_ButtonCancel)
Y — Yes (_ButtonYes)
N — No (_ButtonNo)
R — Retry (_ButtonRetry)
I — Ignore (_ButtonIgnore)
A — Abort (_ButtonAbort)
H — Help (_ButtonHelp)
Пользовательские кнопки

Имена кнопок записываются без пробелов. Регистр символов учитывается. Например:

```
[dm|Вопрос|Вы согласны?|YN|Y]
```

или то же самое:

```
[dm|Вопрос|Вы согласны?]
_Buttons=YN
_Result=Y
```

Стандартные имена кнопок локализуемы (если только не задано PreferSystemDialogs=1 — в этом случае локализацией диалогов занимается операционная система).

Если требуется использовать собственную кнопку, нужно использовать цифру (0–9) вместо буквы. Затем в ключе секции нужно задать текст для этой кнопки.

Пользовательские кнопки получают внутренние имена вида «_Button<номер>». Очевидно, что количество пользовательских кнопок ограничено десятью.

Если текст не будет задан, то им станет номер данной кнопки. Например:

```
[dm|Выбор|номера:|123C|C]
_Button1=один
_Button3=три
```

По умолчанию: O (т. е. одна кнопка «Ok»)

14.3.1.2) Свойство <иконка> (_Icon)

Определяет иконку, отображаемую в окне диалога. Входное свойство.

Данное свойство является особым. Это так называемый подтип диалога. Иконка в некотором смысле определяет важность сообщения, поэтому стандартные варианты нужно (это единственный вариант доступа к встроенным иконкам) указывать прямо в типе секции.

Q — Question — вопросительный знак в синем кружке.
I — Information/Asterisk — восклицательный знак в синем кружке.
W — Warning/Exclamation — восклицательный знак в жёлтом треугольнике.
E — Error/Stop/Hand — крест в красном кружке.

Регистр символа учитывается.

Если подтип-иконка не указан, то по умолчанию диалог отобразится без иконки. Но можно задать пользовательскую иконку посредством ключа секции — свойство _Icon. Здесь следует задать путь к изображению. Размер изображения в окне диалога обычно равен 32x32, хотя программа не будет как-то масштабировать пользовательское изображение. Поддерживаются типы файлов: BMP, PNG и ICO. В формате ICO могут быть несколько иконок. По возможности будет взята иконка 32x32 с максимальным разрешением цветов. Если такой размер отсутствует, будет взята следующая, более крупная иконка. Если таковых нет, будет взята более мелкая иконка. В последнюю очередь выбор опускается до худшего разрешения цветов и на том уровне алгоритм поиска повторяется.

Путь разрешается относительно данного пресета.

14.3.2) [df|_Title|_Text|_Buttons|_Result]

Универсальное сообщение-запрос: от простого сообщения с кнопкой «Ок», до вопроса с несколькими кнопками — вариантами ответа.

Прочие переменные сильно зависят от типа диалога. Эти специфические переменные позволяют получить всю необходимую информацию из диалога после его закрытия. Например, для диалога выбора файла (как открытие/сохранение):

_FullName — полное абсолютное имя файла

_Name — имя файла

_Path — полное абсолютное имя папки

_Directory — имя папки

и т.д.

Специфическим переменным можно присвоить значения перед запуском диалога с помощью ключей секции этого диалога.

Использовать переменные можно как и псевдонимы, заключая в тупые кавычки. Во избежание конфликтов с пользовательскими псевдонимами все переменные диалогов предваряются подчерком.

Для некоторых диалогов уместно автоматически закрывать по истечение некоторого времени. Такой таймаут указывается в секундах через переменную _Timeout в ключах секции. Если переменная не задана или равна 0, то диалог не закрывается автоматически.

Предусмотрены для начала следующие типы диалогов.

[dm<иконка>|<текст заголовка>|<текст сообщения>]

Результат:

_Ok — всегда истина

_Result — всегда "Ok"

===пример==========

[Aliases]

Message=TWinKey functions will be enabled.

Message.1049=Будет автоматически включен функционал TWinKey.

[dmx|kIT Programs PowerPack|`Message`]

_Timeout=10

==========

[dd<иконка>|<текст заголовка>|<текст сообщения>]

Универсальный диалог, в котором можно отобразить любые нужные кнопки.

Список кнопок задается в заголовке секции последним параметром через обратный слеш.

Результат:

_<НажатаяКнопка> — истина

_<ОстальныеКнопки> — ложь

_Result — всегда "<НажатаяКнопка>". Можно присвоить значение для указания кнопки по умолчанию.

===пример==========

[ddq|kIT Programs PowerPack|Вы уверены?|Ok/Cancel]

_Result=Ok

[#if|_Cancel]

[#stop]

==========

[df|<текст заголовка>]

Диалог выбора файла.

===пример==========

[df|Выберите INI-файл]

Name=wincmd.ini

Directory=%COMMANDER_PATH%

[im|`FullName`|Configuration]

Key=Value

==========

Примеры

Пример №1

15) (–) Авторезервирование (набросок)

Если применяется действие Взаимная перезапись (x, обмен), то не всегда будет удобно сохранять старое значение в самом up-файле. По умолчанию так и будет, но следует предусмотреть ключ, с помощью которого можно будет указать файл, куда будут сохранятся старые значения:

[Configuration]

BackupFile=имя_файла

Реальный путь определяется по общему принципу, со значением DefaultDirectory.

Указанный файл будет создан, если не существовал. Но если существовал, то в нём просто будут обновлены/добавлены соответствующие ключи. Таким образом, один резервный файл можно использовать для всех пресетов категории. А значит данный параметр может размещаться и в конфиге категории.

Следует помнить, что, если используется один резервный файл для нескольких пресетов и все эти пресеты используются попеременно, резервирование будет ненадежным. В момент применения первого пресета изначальные значения сохранятся в резервном файле. При применении затем второго пресета сохранятся значения первого, а оригинальные будут потеряны!

Если же использовать отдельные резервные файлы для каждого пресета, то резервирование всё равно не будет полностью адекватным. Второй примененый пресет зарезервирует значение, примененое первым пресетом. То есть появляется зависимость результата от истории применения пресетов. С одной стороны такую функциональность следует оставить, так как, возможно, она позволит создавать сложные, динамически модифицирующиеся пресеты. Но так же важно предоставить стабильный механизм резервирования.

Итак, исключительно в конфиге категории можно определить следующий параметр.

[Configuration]

OriginalFile=имя_файла

Этот файл подобно BackupFile будет служить для сохранения имевшихся в целевых файлах (или реестре) значений, однако, если значение уже занесено в этот резерв-оригинал, то оно не замещается новым "старым" значением. Таким образом, какой бы пресет ни был применен первым, в резерв-оригинал попадут изначальные значения.

При обработке операций обмена в пресете, в категории которого определен резерв-оригинал, нужно сравнить имеющиеся значения параметров в целевом файле (всех вместе для данной секции) со значениями в пресете. Если хотя бы одно значение отличается, значит данный пресет не применялся, поэтому выполняется присвоение значений из пресета. В этот же момент имевшиеся значения записываются в резерв-оригинал (но без замены значений).

В противном случае, если значения в целевом файле совпадают со значениями в пресете, значит данный пресет был применен ранее, и нужно выполнить обратный перенос значений. Эти значения будут взяты из резерва-ори-

гинала, если они там имеются для конкретного параметра, и вообще не изменяются в противном случае.

Если по каким-то причинам в категории объявлен и BackupFile и OriginalFile, то резерв-оригинал имеет преимущество (то есть BackupFile в конфиге категории будет проигнорирован). Однако, BackupFile в самом пресете имеет преимущество и над своим вышестоящим аналогом, и над резервом-оригиналом.

Примеры

Пример №1

16) (-) Секции отката (набросок)

Область применение программы можно значительно расширить, если реализовать возможность не только применять изменения, а сохранять текущее состояние различных настроек. Используя файловые операции (секции [f...), можно легко сохранить нужные файлы целиком, а потом их так же скопировать обратно. Однако не редко будет нужно сохранить только некоторые параметры из INI-файлов, XML-файлов или реестра. Очевидно, что сохранять эти параметры нужно на языке UP, чтобы потом наиболее естественным образом вернуть сохраненные настройки. И если уж автоматизировать создание пресета-бэкапа, то нужно поддерживать весь спектр объектов, которые может потребоваться сохранить. Таким образом требуется дополнительное расширение синтаксиса для секций [i, [r, [x и даже [f.

16.1) Общие настройки

16.1.1) RestoreFile в пресете

16.1.2) RestoreFile в конфиге категории

Up-файл отката указывается глобально для всего пресета или категории в ключе RestoreFile.

Если ключ RestoreFile не задан, все секции сохранения лишаются смысла и игнорируются. Задавать какое-то имя по умолчанию неправильно, так как, если сохранять в файл рядом с пресетом, то его имя может совпасть с другим имеющимся пресетом. Кроме того, настройки сохранятся, чтобы их можно было восстановить, а значит автор пресета должен четко определить имя up-файла отката.

При отсутствии ключа RestoreFile в журнал заносится запись об этом на уровне 3.

16.1.3) RestoreDirectory в пресете

16.1.4) RestoreDirectory в конфиге категории

Для секций [f требуется отдельно определить также ключ RestoreDirectory. В эту папку будут помещаться резервные копии файлов и папок. Отсутствие ключа RestoreDirectory не является ошибкой. По умолчанию используется папка

```
<имя_пресета>.Restore
```

в папке категории.

Один пресет может породить в результате работы только один файл отката. Если требуется одним действием (запустив один пресет) создать несколько файлов отката с различными настройками, то соответствующие секции сохранения нужно определить в различных пресетах, в каждом из которых указать нужные RestoreFile и RestoreDirectory.

Объединить воедино все эти пресеты можно с помощью конструкций [#apply...], во флагах которой нужно включить обработку конфига секции. Конструкция [#include...] для этой цели, очевидно, не подходит. Однако если в одной ситуации нужно создать несколько файлов отката, а в другой один, но с теми же настройками, то замена [#apply...] на [#include...] автоматически даст такую возможность.

Так как в секциях бэкапов не требуется описывать ничего, кроме сохраняемых ключей и файлов, в которых они содержатся, первая часть заголовка секции будет только из одной из букв i, r, x или f состоять после символа "~" . Если указана еще одна буква (буква действия), то именно она будет вставлена в up-файл отката, но ни на что другое она не окажет влияние. По умолчанию же используются наиболее подходящие буквы: im, rm, xm.

В остальном синтаксис очень похож на обычные секции, хотя различия есть.

INI-файлы.

Заголовок также содержит имя файла и имя секции. А логика их обработки такова.

* Если секция состоит только из заголовка, то сохраняется вся секция.

* Если указан хотя бы один ключ, то сохранение происходит выборочно:

- если указано только имя ключа (без знака равно), то ключ сохраняется безусловно,

- в противном случае ключ сохраняется, только если имеет значение, ОТЛИЧНОЕ ОТ заданного (пусть и пустого).

===пример==========

[~i|wincmd.ini|Configuration]

UseRightButton

LongInStatus=0

==========

Такая секция приведет к созданию в up-файле отката следущей секции:

==========

[im|wincmd.ini|Configuration]

UseRightButton=<знач>

LongInStatus=<знач>

==========

Причем LongInStatus будет сохранен, только если он был не равен нулю.

====================

Реестр.

Заголовок содержит путь к ключу реестра. Логика обработки секции такова.

* Если секция состоит только из заголовка, то сохраняется весь ключ реестра.

* Если указан хотя бы один ключ, то сохранение происходит выборочно:

- если указано только имя ключа (без знака равно), то параметр сохраняется безусловно,

- в противном случае параметр сохраняется, только если имеет значение, ОТЛИЧНОЕ ОТ заданного (пусть и пустого).

Синтаксис значения аналогичен обычным r-секциям, то есть может включать тип данных.

===пример==========

[~r|HKCU\Some\Key]

Item1

Item2=dword:00000001

==========

Такая секция приведет к созданию в up-файле отката следущей секции:

==========

[rm|HKCU\Some\Key]

Item1=<знач>

Item2=<знач>

==========

Причем Item2 сохранится, только если его значение не равно единице.

==========

С файлами всё сложнее. Файлы требуется сохранить в определенном месте, а в up-файл отката занести соответствующую секцию [f которая скопирует файлы обратно. В папке, заданной ключем RestoreDirectory, потребуется определить некую иерархию файлов и папок, чтобы избежать конфликтов имен между различными секциями [~f. Эта иерархия не обязана быть как-то связана с реальной иерархией сохраняемых файлов.

===пример==========

[Configuration]

RestoreFile=undo.up

RestoreDirectory=undo

[~f|c:\dir1]

File1.txt

File2.key

[~f|c:\dir2]

File2.txt

[~f|c:\dir3]

File3.txt

==========

Такие секции, кроме того что скопируют эти файлы в подпапки 0, 1 и 2 в папке RestoreDirectory, но и создадут в up-файле отката следующую секцию.

==========

[f|undo\0|c:\dir1]

File1.txt

File2.key

[f|undo\1|c:\dir2]

File2.txt

[f|undo\2|c:\dir3]

File3.txt

====================

Однако, в секции [~f, как и в [f, можно не указывать исходную папку в заголовке секции, а задавать абсолютные пути хоть в каждом пункте секции.

Наверняка потребуется добавить в пресет отката заранее известные конструкции. В этом случае нельзя применять f-секции для дозаписи в файл отката, так как он будет открыт на запись в течение обработки пресета. Да и выглядело бы это не совсем логично.

Для непосредственной записи в файл отката предусмотрена специальная конструкция:

[#store]

Блок

То есть любая секция, следующая за [#store], будет записана в пресет отката. Если блок заключить в операторные скобки [[]…[]], можно записать сразу несколько секций (сами скобки записаны не будут).

Похожая конструкция может применяться для программного формирования нового пресета:

[#store|up-файл]

Блок

Однако эта конструкция позволяет записать только заранее известные секции и по сути никак не связана с функцией сохранения настроек.

Примеры

Пример №1

17) (-) Псевдонимы и переменные (набросок)

Многие типы секций в результате своего выполнения порождают или определяют некоторые объекты. Такие объекты вполне могут применяться в других секциях. Очевидно, как и в случае с диалогами-запросами, такие объекты можно передавать через неявные переменные (псевдонимы). Использовать полученные значения можно, заключая их в тупые кавычки.

Отличный пример — создание процесса. Секцией, например, [pee... создается процесс, кроме того неявно создается переменная _PID. Она может использоваться, например, для ожидания завершения процесса или для самостоятельного его закрытия.

===пример==========

;запускаем прогу

[pee|c:\myprog.exe]

;сохраняем _PID

[a|mypid|`_PID`]

;запускаем калькулятор системно

[pse|calc.exe]

;ждем закрытия калькулятора. Если значение не указано, неявно берется текущее значение _PID

[piw|]

;мягко закрываем прогу

[piq|`mypid`]

==========

Изначально в пресете уже присутствуют несколько неявных переменных, относящихся к свойствам пресета в целом. Это все параметры из секций [Configuration], предваренные соответствующим префиксом: "_Preset_" для пресета, "_Category_" для категории и "_Main_" для самой программы.

Даже такие секции, как например секции для работы с INI-файлами, после выполнения задают значения определенных переменных.

_Ini_File — последний оперируемый INI-файл (с абсолютным путем). Имя файла может и не быть заранее известно, например, если используется секции [I с поддержкой RedirectSection. _Ini_File будет хранить имя фактического файла.

Аналогично _Ini_Section, _Ini_Key, _Ini_Value.

Уже после обработки заголовка секции файловых операции появляются переменные _Target_Directory и _Source_Directory, которые могут использоваться в этой же секции. Они хранят разрешенный абсолютный путь к соответствующим папкам. Кроме того, переменные _Target/_Source_File содержат полностью квалифицированные имена только-что обработанных файлов. Если операция работает с одним файлом (например, удаление), то изменится значение только одной переменной. Эти переменные определяются сразу после выполнения операции, то есть могут использоваться уже в следующей

строке секции.

Итак, псевдонимы можно задавать различными способами. Наиболее востребованным из них, скорее всего, будет секция [Aliases]. Ключами этой секции будут имена псевдонимов, возможно, с уточнением номера языка. А их значением будет, собственно, текст.

Например:

[Aliases]

Hello=Hello world!

Hello.1049=Привет, мир!

...

Очевидно, если в конфиге программы будет выбран язык 1049, то и псевдоним Hello будет содержать русский вариант текста.

Для вставки псевдонима в нужное место его имя нужно заключить в тупые кавычки.

Например:

[im|some.ini|section]

Key1=`myalias`

Key2=`myalias`

Примеры

Пример №1

18) (-) Журнал (в процессе осмысления)

Максимальный уровень важности события для записи в журнал.

1 — ошибка в самой программе (уровень 0),

2 — неверные параметры запуска (уровень 1),

4 — ошибки в заголовке секции (не существующие файлы, например) (уровень 2),

8 — не удалось внести изменение (уровень 3),

16 — не удалось выполнить пре-, пост-действие (уровень 4)

…

19) (±) Генерация предпросмотра

Возможность задать изображение-скриншот для пресета, конечно, предоставляет полную свободу автору пресета. Однако во многих ситуациях делать настоящий скриншот проблематично или бессмысленно. Или же однотипных пресетов очень много, и это просто неблагодарный труд.

Для таких случаев предусмотрен механизм автоматической генерации изображения на основании информации из конкретного пресета. Так в конфиге категории (или же в отдельном файле) можно определить общий шаблон «изображения», применяя некоторые служебные конструкций, которые впоследствии заменятся на конкретные значения из пресета.

Рассматриваемые значения — это совсем не обязательно текст в значении параметра в ini-файле. Это может быть что-угодно. Например, значение цвета может быть задано в различных программах различным способом: просто целое число, шестнадцатиричное число вида RRGGBB, инструкция вида rgb(1,2,3) и т.д.

Поэтому кроме прямой подстановки значения из пресета, возможно выполнить некоторые преобразования значения перед вставкой.

Для целей настраиваемого вывода изображения подходят многие форматы: HTML, SVG, RTF. В некоторых случаях можно задействовать CSS. В разных целевых форматах применимы разные форматы записи, например, цвета. Программе достаточно знать один из допустимых форматов записи конкретных типов значений в целевой файл.

Реализация как такового отображения различных форматов лежит на GUI-части программы и к языку UP по сути отношения не имеет.

При составлении пресетов следует придерживаться ряда правил. Заголовки секции должны записываться одинаково во всех пресетах, иначе адресовать параметры в шаблоне не удастся. Хорошей практикой будет присваивать псевдонимы секциям, чьи ключи потребуются в шаблоне. Псевдонимы также должны совпадать в разных пресетах.

Кроме того, формат записи значений (например, цвета) в каждом пресете должен совпадать. Альтернативный вариант — использовать специальный формат входного параметра «@» (Auto), при котором программа пытается определить формат по фактическому значению. Естественно, существует масса неоднозначностей в записи, например, цвета, поэтому полагаться на псевдоформат «@» всегда не стоит.

В конфиге категории в секции [Configuration] следует задать ключ PreviewTemplate. Относительный путь разрешается относительно папки категории.

Если в пресете задан ключ Preview, то он имеет приоритет (если указанный файл существует, иначе ключ игнорируется).

19.1) (±) Определение шаблона предпросмотра

Далее перечислены возможные способы указания шаблона предпросмотра в порядки убывания приоритета.

19.1.1) (+) Ключ [Configuration] Preview=<имя_файла> в пресете

Наряду с обычными изображениями в этом ключе можно указать файл,

например, RTF. Перед использованием в нём будет произведена замена служебных конструкций.

См. 4.1.3.

19.1.2) (+) Ключ [Configuration] PreviewTemplate=<имя_файла> в конфиге категории

Здесь следует указывать только обрабатываемые файлы (вроде RTF, HTML, SVG и т.п.). Данный файл применяется, только если в конкретном пресете ключ Preview опущен.

См. 1.2.1.12.

19.1.3) (–) Шаблон, описанный внутри конфига категории (в процессе осмысления)

В конфиге категории можно также указать значения по умолчанию для параметров, используемых в шаблоне (не обязательно всех). Это полезно, если содержимое пресетов оговорено не строго. Значения по умолчанию также позволяют просмотреть шаблон безотносительно конкретного пресета.

Значения по умолчанию определяются в секции [AutoPreviewDefaults]. Строки этой секции имеют формат...

19.2) (±) Синтаксис служебных конструкций

Логика обработки файлов может варьировать в зависимости от типа целевого файла. Но в большинстве случаев (как для RTF, так и для XML, HTML, SVG и CSS) подходит следующий синтаксис.

В самом общем виде (все символы обозначают себя):

```
&[[имя_up-файла]имя_секции]параметр=тип:формат1|формат2|...|форматN;
```

Обязательными здесь являются только поля *имя_секции* и *параметр*, т. е. минимальный вид конструкции таков:

```
&[имя_секции]параметр;
```

В этом случае значения параметра будет использовано в виде текста, без изменений.

19.2.1) (–) [имя_up-файла]

Имя up-файла (и автоматически любого ini-файла) задаёт файл, в котором находится извлекаемое значение. Такой вариант следует применять, только если некий параметр содержится в пресете (или части пресета), не являющемся текущим, выбранным, для которого и строится предпросмотр.

```
… &[[common.up]SomeSection]Name; …
```

Указанный файл будет открываться и обрабатываться каждый раз при встрече такого варианта конструкции, что негативно скажется на производительности.

19.2.2) (±) имя_секции]

Имя секции может быть записано в двух вариантах:

19.2.2.1) (+) полный заголовком соответствующей секции

В этом случае требуется полное соответствие с заголовком с учётом регистра.

19.2.2.2) (–) псевдоним секции

Сначала строка рассматривается как имя псевдонима, а потом как заголовок секции. В редких случаях может возникнуть конфликт имён, устранение которого лежит на плечах составителя набора пресетов (всегда можно выбрать другое сочетание символов для псевдонима).

Примеры:

```
… &[im|%COMMANDER_INI%|Colors]BackColor=C:$HBGR|HTML; …
… &[colors]BackColor=C:$HBGR|#HRGB; …
```

В первом случае указывается полный заголовок нужной секции, а во втором используется псевдоним.

19.2.3) (+) параметр

Имя ключа в заданной секции.

19.2.4) (+) =тип:

Тип данных исходного значения. От указанного типа зависит доступный список форматов для конвертирования текстового значения (ведь в up-файле все данные рассматриваются как текстовые) параметра из исходного (в пресете) в целевой (в файле предпросмотра).

Если тип данных не указан (и опущен символ «=»), то значение указанного параметра вставляется как есть, без изменений. В остальных случаях следует явно указывать тип данных для интерпретации значения параметра.

19.2.5) (+) форматX

Смысл перечисленных форматов зависит от типа данных.

Обычно форматов либо не предусмотрено совсем (двоеточие после типа всё равно ставится).

Либо может ожидаться фиксированное количество, например, 2. Тогда они обычно воспринимаются как исходный и целевой форматы. Кстати, целевой формат можно опустить, так как для каждого типа файла предпросмотра (не типа данных!) предусмотрен свой целевой формат по умолчанию.

Либодо пускается неограниченное количество форматов. Тогда они обычно определяют манипуляции над данными.

Для каждого типа будет уточнено, что означают форматы и какие их варианты предусмотрены.

19.3) (+) Тип данных «C»: Цвет

Существует немало способов представления цвета, поэтому предусмотрены большинство из практически встречаемых форматов.

Общий вид строки форматов:

```
C:исходный_формат|целевой_формат
```

19.3.1) (+) Исходные форматы

Если перед записью формата далее указано «[?]», то перед фактической записью значения учитывается данный символ. Чаще всего это «#» или «$».

Регистр в исходном значении не учитывается.

В конструкции, начинающейся с «H», исходное значение может содержать как полную шестнадцатеричную запись, например: #FF88CC, так и сокращённую запись, например: #F8C. Анализатор обнаруживает сокращённый вариант автоматически.

19.3.1.1) (+) [?]HRGB

как принято в HTML, например «#HRGB» правильно распознает:

```
#3280FF
```

19.3.1.2) (+) [?]HARGB

например:

```
#FF808040
```

19.3.1.3) (+) [?]HBGR

как принято в Total Commander, например:

```
CursorColor=$A0FF80
```

19.3.1.4) (+) [?]HABGR

реже встречается в Total Commander, при чём первая составляющая не обязательно означает Alpha-канал. Например:

```
InactiveFocus=$3A0FF80
```

В этом случае, чтобы правильно извлечь цветовые составляющие, нужно применить исходный формат $HABGR.

19.3.1.5) (+) IRGB

основной формат записи цвета в Total Commander, например:

```
CursorColor=10551168
```

Данное число рассчитывается по формуле: R+G*256+B*65536.

19.3.1.6) (+) IBGR

Например, тот же цвет:

```
8454048
```

Данное число рассчитывается по формуле: B+G*256+R*65536.

19.3.1.7) (+) IRGBA

основной формат записи цвета в Total Commander, например:

```
CursorColor=10551168
```

Данное число рассчитывается по формуле: R+G*256+B*65536+A*16777216.

19.3.1.8) (+) IBGRA

Например, тот же цвет:

```
8454048
```

Данное число рассчитывается по формуле: B+G*256+R*65536+A*16777216.

19.3.1.9) (+) rgb

как возможно в CSS 2. Составляющие перечисляются в десятичной записи через запятую. Допускается задание значения в процентах. Пробелы в записи игнорируются, например:

```
rgb(128,160, 255)
rgb(50%, 60%,100%)
```

19.3.1.10) (+) rgba

расширение rgb в CSS 3. Alpha-канал записывается дробным десятичным числом 0..1. Целая часть отделяется точкой, например:

```
rgb(128,160,255,0.5)
```

19.3.1.11) (+) hsl

как возможно в CSS 3. Первый параметр определяет оттенок в градусах (в диапазоне 0..360). Второй — насыщенность, задаваемая в процентах. Третий — примесь белого, в процентах. Пробелы в записи игнорируются, например:

```
hsl(270, 100%, 50%)
```

19.3.1.12) (+) hsla

аналогично hsl, но с четвёртым параметром: Alpha-канал в виде дробного числа 0..1. Пробелы в записи игнорируются, например:

```
hsla(120, 100%, 50%, 0.75)
```

19.3.2) (+) Целевые форматы

Программа не является конвертером между формами записи цветов, поэтому достаточно одной формы записи для каждого применяемого типа изображения.

19.3.2.1) (+) RTF

Вместо конструкции вставляется строка вида «\redR\greenG\blueB;», где R, G и B записаны в десятичном виде 0..255, например:

```
\red255\green160\blue32;
```

Такая строка применяется для определения палитры в RTF-документах в секции {\colortbl ; ... }.

19.3.2.2) (+) HTML

Вместо конструкции вставляется строка вида «#RRGGBB», где R, G и B записаны в шестнадцатеричном виде 00..FF, например:

```
#FFA020
```

19.3.3) (+) Специальный исходный формат «@»

(стилизованное «Auto»). Данный псевдоформат служит для указания программе автоматически распознать формат фактического значения. Так как существуют неоднозначности в записи цвета, данный режим пригоден только для ограниченного подмножества форматов. В случае неоднозначности в данное подмножество добавляется наиболее распространённый формат.

19.3.3.1) (+) Начинается с цифры

Применяется формат IRGB.

19.3.3.2) (+) Начинается с «#»

Применяется формат #HRGB.

19.3.3.3) (+) Начинается с «$»

Применяется формат $HBGR.

19.3.3.4) (+) Начинается с «r»

Применяется формат rgb.

19.3.3.5) (+) Начинается с «h»

Применяется формат hsl.

19.4) Тип данных «T»: Текст

Как уже упоминалось, если не указан никакой тип данных (и опущен символ «=»), то в целевое содержимое вставляется текст из пресета без изменений. Однако тип данных «T» позволяет произвести некоторые манипуляции со строкой перед вставкой в целевое содержимое.

Общий вид строки форматов:

```
T:формат1|формат2|...|форматN
```

Каждый формат определяет некоторую манипуляцию над исходной строкой, полученной из пресета. Каждая следующая манипуляция производится над результатом предыдущей. У каждой манипуляции может быть собственный подсинтаксис записи соответствующих параметров, о чём будет упоминаться далее.

19.4.1) Манипуляция «S»: Подстрока

Формат:

```
S<позиция_начала>,<позиция_конца>
```

Отсчёт позиций ведётся с 1.

Значения (как по отдельности, так и одновременно) могут быть опущены, тогда они подразумевают соответствующее предельное значение: с первого символа и до конца строки.

Значения могут быть отрицательными, тогда позиция отсчитывается от конца строки.

Пример:

```
...&[im||Colors]ColorFilter1=T:S2,;...
```

устранит обычно присутствующий в начале значения таких ключей символ «>».

20) (-) Вычисляемые поля (в процессе осмысления)

Если любая секция начинается с символа "**$**", то все её не фиксированные поля могут быть вычислены перед анализом этой секции.

Синтаксис вычисляемых конструкций позаимствован из языка Лисп. То есть, если требуется вычислить какие-либо поля, нужно взять выражение в круглые скобки и применить различные функции. Если выражение не взято в скобки, оно воспринимается атомом, то есть используется как есть.

Такие секции позволят выполнить любые мыслимые действия. Наиболее реалистичный вариант использования — манипуляции объектами, имена которых зависят от некоторых факторов.

Например (использованы гипотетические команды):

```
[$im|wincmd.ini|(concat (screen_width) "x" (screen_height) " (8x16)")]
maximized=(round (random 0 2))
```

21) (–) Окна (в процессе осмысления)

Секции для работы с окнами и различными дескрипторами предваряются буквой «**w**».

22) (-) Текстовые файлы (набросок)

Для работы с текстовыми файлами предусмотрен тип секций [t...], для бинарных - [b...].

Эти секции позволяют производить все мыслимые операции над содержимым файла. Начиная от дозаписи строки в конец текстового файла и заканчивая патчингом исполняемых файлов.

Текстовые файлы могут быть записаны в различных кодировках, а также использовать различные варианты перевода строки. Поэтому программа должна не только автоматически определять по возможности некоторые кодировки, но и позволять указывать любую существующую кодировку явно.

Общий синтаксис:

[t<тип элемента><действие>|<файл>|<кодировка>|<перевод строки>]

<адрес>=<содержимое>

Перевод строки задается константами (win, nix, mac) или набором шестнадцатиричных чисел, определяющих коды символов - разделителей строк. Например, вместо "win" можно написать "0d0a". Если поле опущено, принимается вариант "win".

Кодировка может быть опущена:

[t??|<файл>]

В таком случае производится вялая попытка определить кодировку: по первым двум байтам определяется Unicode ли это и какого типа. Если не Unicode, предполагается CP1251.

Адрес - это выражение, позволяющее точно определить элементы к которым применяется действие. В адресе можно применять номера, специальные константы и шаблоны. Более того имеются средства сложного контекстозависимого комбинирования.

В простейшем случае адресом будет номер.

Смысл номера зависит от типа действия. Например, в случае вставки строки это номер строки, после которой будет вставлена указанная строка.

[tli|file.txt]

2=новая строка

5=еще строка

Предусмотрены также константы:

$ - указывает на последний элемент (например, строку)

^ - указывает на элемент, предшествующий первому (виртуально, равносильно "0")

Шаблоны позволяют оперировать неизвестными заранее позициями. Шаблон заключается в кавычки и записывается в синтаксисе базовых регулярных выражений.

Для каждого типа секции определен смысл номеров и вообще адресов. То есть, например, для "[tli" очевидно, что адреса будут относится и указывать на строку, а не на символ или слово. Смысл адреса по умолчанию применяется к указанному адресу (возможно составному) в заключении его обработки, чтобы можно было осуществить действие, предписанное в заголовке секции.

Если же потребуется, например, адресовать неизвестную строку на основании наличия в ней некоторого слова, то придётся явно указать смысл адреса с помощью символа-префикса.

L - строка

W - слово

C - символ

S - подстрока.

Например, удалить строки, в которых встречается некое слово или содержится некий символ :

[tld|text.txt]

W"word"

C"@"

Комбинирование

Явные адреса (номера и константы) и неявные адреса (шаблоны) можно дополнительно уточнять с помощью адреса любого типа, указывая их последовательно через двоеточие ":".

Уточнение явного адреса шаблоном понимается как условие: такой-то номер, если он соответствует шаблону.

1:"^@echo off$" - первая строка, если в ней ровно "@echo off".

Уточнение шаблона номером/константой означает порядковый номер элемента, соответствующего шаблону.

"word":$ - последнее вхождение слова word.

Если это имеет смысл, можно комбинировать и однотипные адреса.

Общее правило Уточнение - выражение справа от двоеточия.

Уточнение-шаблон добавляет условие. То есть среди элементов, выбранных с помощью неуточненного адреса, будут адресованы только те элементы, которые соответствуют шаблону-уточнению.

Уточнение-номер адресует один из элементов, выбранных с помощью неуточненного адреса, по порядковому номеру.

Важно понимать, что явное указание смысла адреса как бы отвязывает адрес от текущего места обработки. То есть поиск и сопоставление, например, слова будет осуществляться в границах всего файла. Поэтому, если такой адрес уточняется, например, номером, то этот порядковый номер будет отсчитываться от начала файла.

Например, удалить ОДНУ строку, где некое слово встретится десятый раз:

[tld|text.txt]

W"word":10

Выражение в целом удовлетворяется, если последовательно удовлетворились все составляющие уточнения.

Например:

"шаблон":$ - адресует последний элемент среди соответствующих шаблону.

Можно также просто выполнить операцию И над адресами, а не уточнять некоторый адрес.

Например:

$,5 - только если последняя строка будет пятой

Имеются следущие типы обработки текстовых файлов, другими словами типы элементов.

l - целая строка.

w - целое слово. При этом разделителями считаются пробел, табуляция, переводы строки.

c - один символ. Вполне можно применять шаблон, но он должен соответствовать одному символу.

s - подстрока. Можно считать это расширением типа "c", так как обрабатывается ровно тот фрагмент текста, который удовлетворил шаблону. Здесь нельзя задавать только номер элемента без шаблона. Можно обратиться не только ко всем элементам, но и к одному по номеру (константе). Такой номер следует указывать после шаблона через двоеточие ":". Главное отличие от адресации по "l" в том, что здесь ничего не значат переводы строки, то есть текст рассматривается монолитом. Это применимо, например, для пакетной смены типа переводов строк.

Имеются следующие виды действий над элементами.

d - удаление. Адресуемый фрагмент будет вырезан из текста, а остальной текст будет сдвинут к началу.

i - вставка. Остальной текст будет сдвинут к концу. И удаление, и вставка осуществляется отдельно для каждой строчки секции, а значит номера следует соответственно пересчитывать заранее.

r - замена. Фрагмент заменяется на указанное.

[tld|file.txt|utf16be]

"^//"
Удалит все строки, начинающиеся с комментария.
В данном случае символ "=" можно опустить.
[twr|file.txt|koi8r]
"[Mm]icro"=микро
Заменить все слова "Micro" и "micro" на "микро".
Примеры комбинирования.
[tsr|script]
1,"/sh"=/bash
Заменить подстроку /sh на /bash в первой строке файла..

23) (–) Бинарные файлы (набросок)

Для работы с бинарными файлами предусмотрен тип секций [b…].