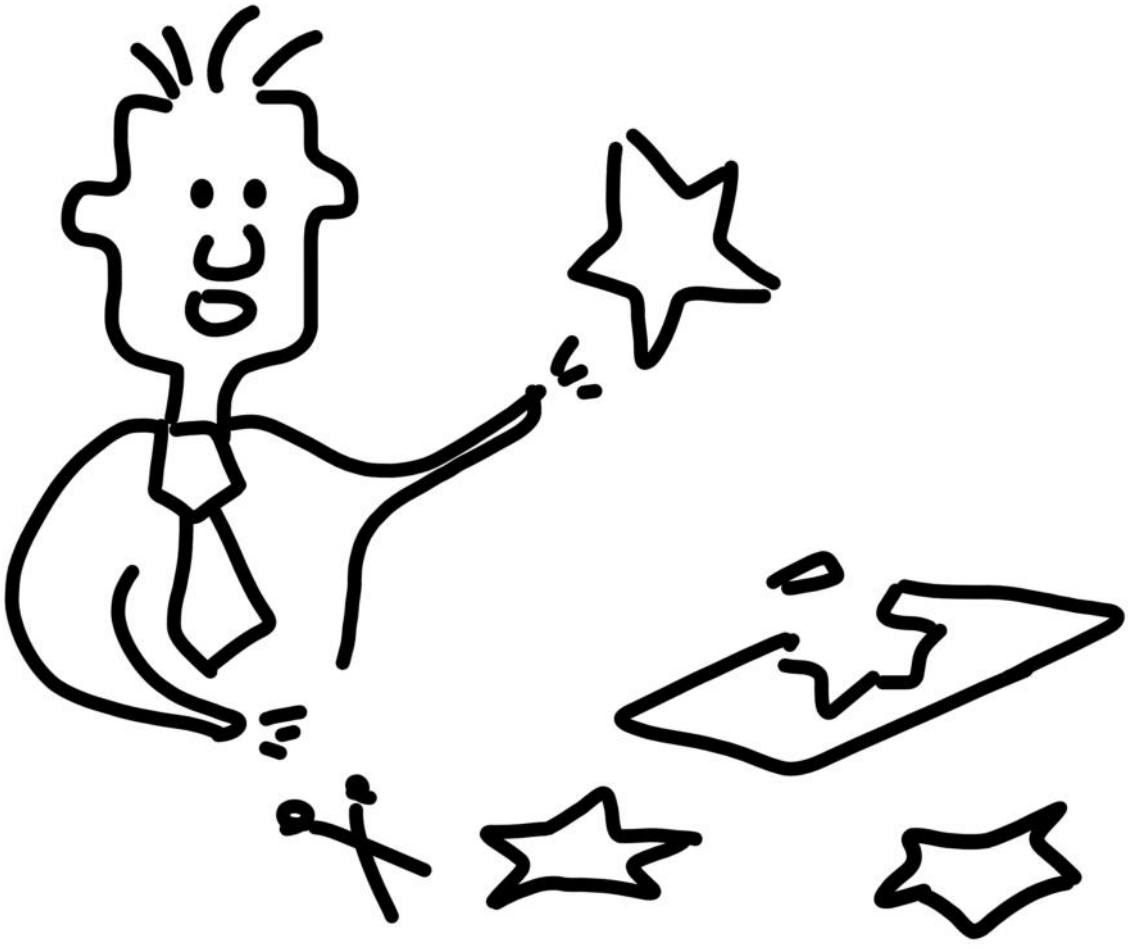
PRAGMA

Руководство администратора

Октябрь 2011



Пятый Уровень

Создание шаблонов транзакций

В руководстве описывается создание шаблонов EPM-транзакций для решения «Пятый Уровень».

1. EPM-транзакции

Работая в бизнес-приложениях (программах), пользователи часто ***циклически*** выполняют определенные типовые операции. Например, формируют платежный документ. Если у такой операции имеются формальные признаки начала и окончания операции, то операцию можно определить как **транзакцию**. Между началом и окончанием транзакции пользователь может производить в программе другие действия. По формальным признакам (событиям), которые фиксирует EPM-агент по ходу работы программы, можно определить, что пользователь произвел какое-либо действие. Таким образом, событие начала, событие окончания, а также сценарий возможных действий пользователя между этими событиями, формально описывает транзакцию.

Начавшись, транзакция может закончиться успешно (пройти по одному из путей, предусмотренных в сценарии). Такие транзакции будем называть ***успешными***.

Если в сценарии предусмотрены события или комбинация событий, при наступлении которых фиксируются ошибки (критически значимые отклонения от нормального течения транзакции), то транзакция принудительно завершается, и будет иметь признак ***транзакции с ошибкой***.

Возможно также, что при наступлении некоторых событий фиксируется ***нетипичное поведение*** (Unusual Case) в ходе транзакции. В качестве примера можно привести выдачу программой сообщения о неверных действиях пользователя или о не фатальной ошибке. При этом выполнение транзакции может быть продолжено. Количество случаев нетипичного поведения фиксируется в принадлежности к конкретной транзакции, в ходе выполнения которой они произошли.

Если для транзакции вводится значение таймаута (время, за которое типичная транзакция должна окончиться), и начатая транзакция не завершается в отведенное время, то она завершается принудительно и будет называться ***транзакцией, завершённой по таймауту***.

Наконец, при некоторых условиях начавшаяся транзакция может быть ***снята***. Таким условием может быть завершение работы программы или невыполнение некоторых предварительных условий (если таковые определены), которые должны быть выполнены перед началом транзакции и продолжать выполняться в ходе транзакции.

В процессе выполнения транзакции производится замер времени ее работы (работает таймер). Стоит отметить, что имеется возможность включать и выключать таймер по любым событиям в ходе транзакции, не обязательно по началу и окончанию транзакции.

2. Структура шаблонов транзакций EPM

Шаблон EPM-транзакции – xml-файл, размещённый в подкаталоге Templates каталога установки программы EPMAgent.exe.

Начинается строкой `<?xml version="1.0" encoding="utf-8"?>` (кодировка может быть и другая).

Корневой узел (node) должен иметь имя **document** с атрибутами `type="EpmDoc" version="1.0"` и атрибут с именем `xmlns`. Значение атрибута может быть любым текстом, сдержащим имя узла описания транзакции:

```
<?xml version="1.0" encoding="utf-8"?>
<document type="EpmDoc" version="1.0" xmlns="xTemplate">
  <xTemplate type="GUI_TA_TEMPLATE" Version="1.0">
    <Transactions>
      <Transaction …
      …
      …

      </Transaction>
    </Transactions>
  </xTemplate>
</document>
```

Пример 1. Структура корневых элементов xml-документа с шаблонами транзакций.

В примере, значение атрибута `xmlns="xTemplate"`, соответственно в корневом теге document содержится узел с этим именем **xTemplate**. Узел должен иметь атрибут type. В спецификации версии 1.0 поддерживается только тип `type="GUI_TA_TEMPLATE"`. Этот тип означает транзакции на основе событий графического интерфейса пользователя (GUI).

Узел xTemplate должен содержать, по крайней мере, один узел с именем **Transactions**, в который входят узлы (по крайней мере один) с именем **Transaction**. Последние и содержат описание отдельных GUI транзакций.

Каждый узел **Transaction** должен содержать набор обязательных и опциональных атрибутов, например:

```
<Transaction GUID="{6B27EBEE-D4E9-46bd-9A79-E443361AED8C}"
             Vendor="ProLAN"
             Title="Поиск агентов управления"
             alias="RCAgentsSearch"
             Process="NPMOper.exe"
             Bits="32"
             subtype="system"
             Timeout="60000"
             Group="Task"
             Threshold="10000"
             ApplicationGuid="{D27C32BF-F640-47bc-BEB8-F9E3465F439F}"
             Application="SLA-ON Operations"
             ApdexGuid="{3594D9FF-7D29-4543-97E2-A398368C3409}"
             ApdexGuidName="Office Location"
             Protocol="UDP">
```

```
            …
            …
    </Transaction>
```

Пример 2. Узел Transaction с атрибутами.

В Таблице 1 приведён полный список атрибутов тега **Transaction**.

Имя атрибута	**Обязательный /опциональный**	**Описание**	**Значения атрибута**
GUID	Обязательный	Уникальный идентификатор транзакции. Не должно быть двух транзакций с одинаковым GUID.	Значение GUID транзакции можно генерировать специальными утилитами, например guidgen.exe.
Vendor	Обязательный	Строка, длиной до 255 символов, содержащая название компании – разработчика программы, для которой создается транзакция.	
Title	Обязательный	Строка, длиной до 255 символов, содержащая название транзакции, так как ее считает необходимым назвать разработчик шаблона.	
Alias	Опциональный	Псевдоним транзакции, строка длиной до 255 символов.	Если атрибут не задан, то псевдоним транзакции отсутствует.
Process	Обязательный	Строка, длиной до 260 символов, содержащая имя образа модуля процесса (имя процесса) программы, для которой создается транзакция. Имя процесса не включает дисковый путь к модулю.	
Bits	Опциональный	Содержит значение разрядности модели приложения. Поддерживаются 32 и 64 разрядные приложения.	Если атрибут не задан, то приложение идентифицируется только по имени процесса, все зависимости от разрядности.
Subtype	Опциональный	Определяет один из возможных подтипов транзакции: • Тип неизвестен;	Unknown User System

		• Пользовательская транзакция – действия в транзакции выполняются пользователем, и реакция системы не учитывается; • Системная транзакция – учитывается только время реакции системы; • Смешанная транзакция – учитывается и время реакции системы, и время действий пользователя.	Misc Если атрибут не задан, то принимается значение **Misc**
Timeout	Опциональный	Значение максимального времени, в миллисекундах, в течение которого транзакция должна завершиться. Если транзакция продолжается дольше заданного времени, то фиксируется таймаут и транзакция снимается.	Время в миллисекундах. Если атрибут не задан, либо задано значение 0,то отсчет таймаута на завершение транзакции не производится (транзакция никогда не будет снята по таймауту).
Threshold	Обязательный	Определяет порог времени выполнения транзакции для расчета оценки по методике Apdex.	Время в миллисекундах. Если задано значение атрибута Timeout, то значение Timeout должно быть больше в 4 и более раза значения Threshold (по методике расчета Apdex).
Group	Опциональный	Определяет один из возможных подтипов замеров времени внутри транзакции: • Не задано; • Задача (Task) – В транзакции замеряется не общее время выполнения транзакции, а время между некоторыми событиями внутри транзакции, например, время реакция системы на запрос и получение ответа от сервера; • Цепочка задач (Task Chain) - В транзакции замеряется несколько отрезков времени между различными событиями. Например, в транзакции выполняется несколько запросов	None Task Task Chain Process Если атрибут не задан, то принимается значение **None**

		к серверу; • Процесс (Process) – В транзакции замеряется, как правило, общее время ее выполнения.	
Application	Обязательный	Строка, длиной до 255 символов, содержащая название приложения либо сервиса, к которому относится приложение. Разные приложения могут быть объединены в один сервис, например сервис электронной почты.	
ApplicationGuid	Обязательный	Уникальный идентификатор, соответствующий значению атрибута Application.	Значение GUID можно генерировать специальными утилитами, например guidgen.exe.
ApdexGuidName	Обязательный	Строка, длиной до 255 символов, содержащая название комбинации факторов в которых работает приложение. Например, клиент-серверное приложение может стоять, в одном случае, в одной локальной сети с сервером, а в другом случае доступ к серверу осуществляется через интернет. Очевидно, что это разные конфигурации системы и соответственно на выполнение одинаковых транзакций будут влиять разные группы факторов.	Название группы факторов по возможности должно характеризовать набор факторов, влияющий на условия выполнения транзакции. Например: Office Location, Internet Location.
ApdexGuid	Обязательный	Уникальный идентификатор, соответствующий значению атрибута ApdexGuidName. Отдельно описанные транзакции группируются вместе, если у них совпадает значения атрибутов ApplicationGuid и ApdexGuid, т.е. транзакции относятся к одному приложению (либо сервису) и работают в одинаковой группе факторов. Группировка производится на уровне расчета Apdex приложений	Значение GUID можно генерировать специальными утилитами, например guidgen.exe.

		(сервисов).	
Protocol	Опциональный	Строка, длиной до 255 символов, содержащая название протокола, то которому выполняется транзакция, например UDP, TCP, HTTP, SMB и т.п.	Если значение не задано, то атрибут содержит пустую строку.

Таблица 1. Атрибуты тега Transaction

3. GUI-события

Технология Пятого Уровня основана на отслеживании наступления определенных событий при работе бизнес-приложений. Эти события связаны с графическим интерфейсом Windows приложений, поэтому называются **GUI-событиями**. Различных типов GUI-событий десять:

1. **Создание окна**. Для того чтобы понять, что создается именно нужное окно, в качестве условия можно задать произвольный набор ограничений на такие свойства как:
   Имя класса или атом окна, текст или шаблон текста, стиль, идентификатор, предок и родитель окна.
   Все другие типы GUI-событий ссылаются на соответствующее событие создания окна.
2. **Закрытие (уничтожение) окна**.
3. **Изменение текста окна**.
4. **Разрешение/запрещение окна** (Enabled/Disabled).
5. **Показ/скрытие окна.**
6. **Инициализация окна диалога.**
7. **Активизация дочернего MDI окна.**
8. **Изменение стиля окна.**
9. **Получение окном команды от меню, акселератора или элемента управления.**
10. **Получение окном нотификации от элемента управления.**

GUI-событие может находиться в одном из двух состояний: **выполнено** и **не выполнено**. При определенных условиях, в зависимости от логики, заложенной в сценарий, выполненное GUI-событие может вновь стать не выполненным. В этом случае, если становится невыполненным GUI-событие создание окна, то автоматически становятся невыполненными другие GUI-события, которые ссылаются на это событие. Транзакция (в упрощенном понимании) содержит определенную последовательность GUI-событий, обязательно содержащую в своем начале GUI-событие создания окна.

Далее в сценарии описываются другие GUI-события, которые ссылаются на событие создания окна. Представим, что для выполнения транзакции, вслед за созданием окна требуется выполнение определенной последовательности других GUI-событий, которые ссылаются на созданное окно. Например, нажатие кнопки (команда), изменение текста в заголовке окна, создание дочернего окна с определенным текстом и т.д. Теперь представим, что вслед за созданием окна (GUI-событие выполнено), происходят (также становятся выполненными) другие GUI-события, но не все которые требуются по описанию. Далее окно закрывается. Т.к. транзакция не выполнена полностью, то GUI-событие создания окна становится невыполненным. Вполне логично, что невыполненными становятся и все другие события, ссылающиеся на создание окна.

4. Блоки

Блок – это универсальный контейнер шаблона, который содержит внутри себя GUI-событие или другие блоки. Блок, как и GUI-событие, может находиться в **выполненном** и **невыполненном** состоянии. Сценарий транзакции (узел Transaction) содержит опциональный узел `PreCondition` и обязательный узел `Scenario`, которые являются **корневыми блоками** предусловия и основного цикла транзакции.

```
<Transaction …>
  <!-- Опциональный блок предусловия -->
  <PreCondition …>
    …
    …
  </PreCondition>

  <!-- Обязательный блок основного цикла транзакции -->
  <Scenario …>
    …
    …
  </Scenario>
</Transaction>
```

Многоточие (…), следующее за тегами блоков `PreCondition` и `Scenario` означают набор атрибутов блоков, который будет рассмотрен ниже.

Если для слежения за ходом выполнения транзакции, изначально должны быть выполнены некоторые условия, то в шаблоне создается блок **PreCondition** (предварительное условие). Такими условиями, например, может быть создание главного окна программы при ее запуске, создание дочерних окон программы и т.п. Довольно часто в конкретных шаблонах транзакций `PreCondition` отсутствует или содержит единственный блок GUI-события создания окна, реже содержит последовательную цепь GUI-событий.
Блок `Scenario` описывает циклически выполняющуюся в приложении транзакцию. Это обязательный блок. Блоки, содержащиеся внутри блоков `PreCondition` и `Scenario`, в структуре документа представляются узлами с именами тегов `Block0`, `Block1` и т.д.

```
<Transaction …>
  <PreCondition … >
    <Block0 …>
      <GuiEvent …>
        …
      </GuiEvent>
    </Block0>
    <Block1 …>
      <GuiEvent …>
        …
      </GuiEvent>
    </Block1>
  </PreCondition>

  <Scenario … >
    <Block0 …>
      <Block0 …>
        <GuiEvent …>
```

```
            …
          </GuiEvent>
        </Block0>
        <Block1 …>
          <GuiEvent …>
            …
          </GuiEvent>
        </Block1>
      </Block0>

      <Block1 …>
        …
      </Block1>
    </Scenario>
  </Transaction>
```

Пример 3. Блоки, входящие в корневые блоки `PreCondition` *и* `Scenario`.

Номер блока, следующий за ключевым словом, определяет последовательность вхождения в родительский блок. Номера блоков должны всегда начинаться с 0 и идти подряд. В примере 3 показано, что блоки, входящие в корневой блок, также могут иметь входящие в них дочерние блоки. Уровень вложенности не ограничен. Важно отметить, что в самой глубине вложения всегда имеются блоки, содержащие GUI-события `GuiEvent`.

5. Типы блоков

Тип блока определяется значением атрибута `type`, в теге объявления блока. Существуют три типа блоков: **GuiEvent**, **Chain** и **Alternative**. Самый простой тип – блок, содержащий GUI-событие (или просто **блок GUI-события**). В структуре документа этот тип блока выглядит следующим образом:

```
<BlockN type="GuiEvent" …>
  <GuiEvent …>
    …
  </GuiEvent>
</BlockN>
```

Пример 4. Блок GUI-события.

Блок представляет единое целое с GUI-событием. Блок будет выполненным, если выполнено GUI-событие.

Второй тип блока – **последовательный блок** (Chain). Блок этого типа содержит другие блоки, среди которых могут быть и блоки GUI-событий. Блок этого типа будет выполнен, если выполнены все блоки, входящие в него.

```
<BlockN type="Chain" …>
  <Block0 …>
    …
  </Block0>
  <Block1 …>
    <GuiEvent …>
      …
    </GuiEvent>
  </Block1>
  …
</BlockN>
```

Пример 5. Последовательный блок.

Следует отметить, что последовательный блок имеет две разновидности (подтипа), определяемые значением атрибута `order` в теге объявления блока. Если требуется, чтобы все блоки, входящие в последовательный блок выполнялись в строгом порядке, соответствующем номерам блоков, то атрибут содержит значение `order="sequenced"`. Если порядок выполнения блоков не имеет значения, то атрибут имеет значение `order="any"`.

Третий тип блока – **параллельный блок** (Alternative). Блок этого типа содержит другие блоки, среди которых могут быть и блоки GUI-событий. Блок этого типа будет выполнен, если будет выполнен любой входящий в него блок.

```
<BlockN type="Alternative" …>
  <Block0 …>
    …
  </Block0>
  <Block1 …>
    <GuiEvent …>
      …
    </GuiEvent>
```

```
    </Block1>
    …
</BlockN>
```

Пример 6. Параллельный блок.

Чтобы проиллюстрировать возможности использования различных типов блоков в сценариях приведем аналогию между выполнением сценария транзакции и человеком, проходящим лабиринт комнат, у которого имеется единственный вход (начало транзакции) и единственный выход (окончание транзакции). Внутри лабиринта есть множество комнат (GUI-события). Заходя в комнату, человек включает в ней свет и двери, выходящие из нее, открываются (событие выполнено). Человек может двигаться дальше, но может и вернуться.

Некоторые комнаты группируются в последовательно проходимые галереи (последовательные блоки `order="sequenced"`). Войти в некоторые комнаты можно свободно, но выйти можно только включив весь свет (несколько выключателей в одной комнате). Не имеет значения, в какой последовательности вы будете их включать, но нужно включить все (последовательные блоки `order="any"`).

В другие комнаты или галереи можно войти из одной и той же комнаты и в результате попасть в другую определенную комнату, независимо от того, в какую комнату вы пошли сначала (параллельные блоки). Некоторые комнаты или галереи могут быть заминированы. Если человек войдет в такую комнату или пройдет такую галерею до конца, то он взорвется (транзакция завершена с ошибкой).

У некоторых комнат или галерей есть вход, но нет выхода. Если вы туда зашли – вам придется возвращаться обратно и свет будет за вами гаснуть (нетипичное поведение в транзакции). Скажите спасибо, что не взорвались!

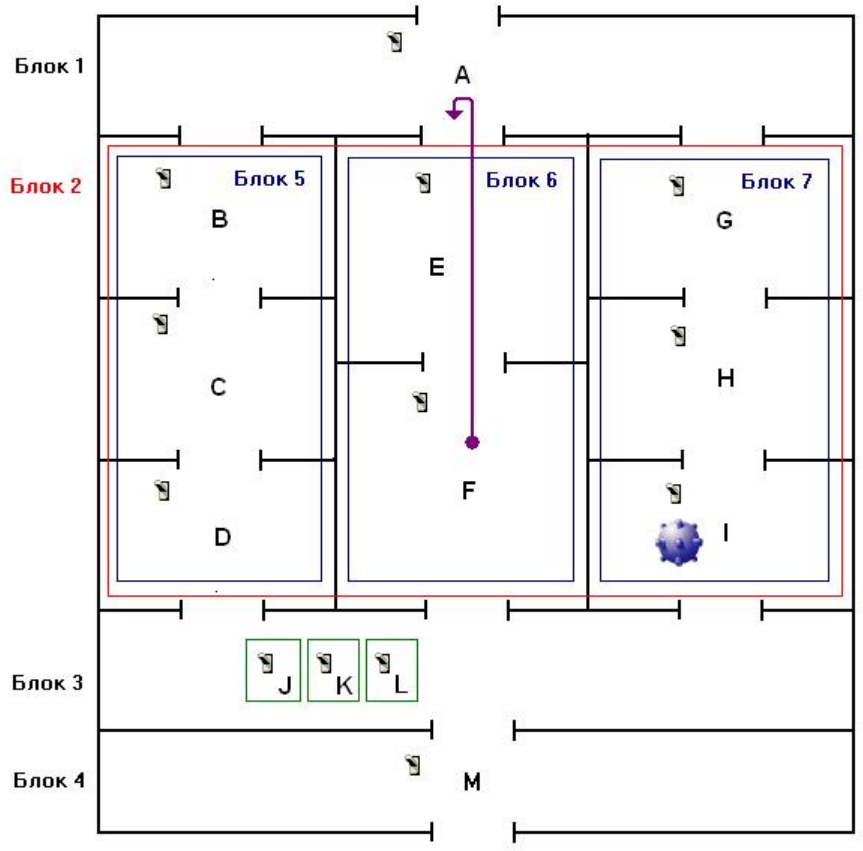


Рисунок 1. Сценарий транзакции - аналогия.

Транзакция, представленная на рисунке 1 от корня (вход в лабиринт) имеет 4 блока:

- *Блок 1.* Содержит единственную комнату A (GUI-событие). Это блок `type="GuiEvent"`.
- *Блок 2.* На рисунке выделен прямоугольником красного цвета. Это сложный блок. Он содержит внутри себя три параллельно выполняемых блока: Блок 5, Блок 6 и Блок 7, т.е. является блоком `type="Alternative"`. Блоки 5, 6 и 7 (галереи) в свою очередь являются последовательными блоками `type="Chain"` с порядком следования `order="sequenced"`. Каждый из блоков 5, 6 и 7 содержит внутри себя от 2-х до 3-х простых блоков, являющихся GUI-событиями.
- *Блок 3.* Является последовательным блоком `type="Chain"` с порядком следования `order="any"`. Он содержит три простых блока `type="GuiEvent"` – J, K и L, которые могут выполняться в любой последовательности. Но блок 3 будет выполнен (свет включится), только если все 3 события будут выполнены.
- *Блок 4.* Как и Блок 1, является простым блоком с `type="GuiEvent"`.

Заметим также, что сам сценарий является блоком `type="Chain"` с порядком следования `order="sequenced"`, т.к. содержит последовательно выполняемые блоки 1,2,3 и 4. К рисунку 1 мы будем обращаться при рассмотрении некоторых атрибутов GUI-событий и блоков.

6. Атрибуты GUI-событий

Если какой-либо блок сценария содержит GUI-событие, то блок содержит узел `GuiEvent` (см. пример 4). За открывающим тегом `GuiEvent` следуют атрибуты события. Важнейший атрибут - `type` . Как упоминалось выше, существует 10 различных типов GUI-событий.

Значение атрибута типа GUI-события	**Описание**
`WindowCreated`	Создание окна Window с определенными параметрами. Параметры создаваемого окна задаются внутри узла `GuiEvent`. Набор параметров будет рассмотрен ниже. Сценарий транзакции содержит, по крайней мере, один блок GUI-события с этим типом, т.к. другие типы событий ссылаются на события создания окон.
`WindowClosed`	Закрытие созданного окна. Внутри узла `GuiEvent` для данного типа события задается единственный параметр – ссылка на событие создания окна. Формат параметра будет рассмотрен ниже.
`WindowSetText`	Задание или изменение текста окна. В зависимости от класса окна Windows, текстом окна может являться текст в заголовке окна, текст в строке редактирования, текст в элементах управления или статических окнах и т.п. Внутри узла `GuiEvent` для данного типа события задаются параметры, которые будут рассмотрены ниже.
`WindowEnabled`	Разрешение или запрещение окна в интерфейсе программы. Параметры задаются внутри узла `GuiEvent` и будут рассмотрены ниже.
`ShowWindow`	Показ или скрытие окна. Параметры задаются внутри узла `GuiEvent` и будут рассмотрены ниже.
`InitDialog`	Инициализация диалога. Внутри узла `GuiEvent` для данного типа события задается единственный параметр – ссылка на событие создания окна диалога. Формат параметра будет рассмотрен ниже.
`MDIActivated`	Активизация дочернего окна в MDI интерфейсе. Параметры задаются внутри узла `GuiEvent` и будут рассмотрены ниже.
`WindowStyleChanged`	Изменение стиля окна. Параметры задаются внутри узла `GuiEvent` и будут рассмотрены ниже.
`Command`	Получение окном команды от меню, акселератора или элемента управления. Параметры задаются внутри узла `GuiEvent` и будут рассмотрены ниже.
`Notify`	Получение окном нотификации от элемента управления. Параметры задаются внутри узла `GuiEvent` и будут рассмотрены ниже.

Таблица 2. Типы GUI-событий

Другим важным параметром является псевдоним `alias` GUI-события (не путать с псевдонимом блока). Псевдоним может содержать произвольный текст и является обязательным для GUI-событий, на которые ссылаются другие GUI-события. Например, для событий создания окна (`type="WindowCreated"`) псевдоним чаще всего необходим, правда бывают исключения, когда на окно никто не ссылается (требуется только, чтобы оно было создано и всё). В любом случае, вы можете без каких-либо ограничений задавать псевдонимы любым типам GUI-событий, вне зависимости от того, ссылаются на них, либо нет.

```
<GuiEvent type="WindowCreated" alias="CreateAboutDlg">
  …
</GuiEvent>
  …
<GuiEvent type="Command" alias="OK buttom pressed">
  <Receiver>CreateAboutDlg</Receiver>
    …
</GuiEvent>
…
<GuiEvent type="WindowClosed" alias="AboutDlgClosed">
  <Window>CreateAboutDlg</Window>
</GuiEvent>
```

Пример 6. Типы GUI-событий и их псевдонимы.

7. Параметры GUI-событий типа WindowCreated

Тип GUI-событий создания окна обладает большим набором возможных параметров, позволяющих уточнять, создание какого именно окна ожидается, а также возможные действия при закрытии окна. Несмотря на то, что ни один из параметров не является обязательным, тем не менее, для создаваемого окна **должен быть задан, по крайней мере, один из параметров**: имя класса окна, либо атом окна, либо текст окна.

Параметр	**Описание**	**Значения**	**По умолчанию**
`ClassAtom`	Атом класса окна. 16-ти битное целое число, записанное в шестнадцатеричном виде. Приложения получают от системы атом класса, регистрируя собственные классы окон. Система имеет стандартные (предопределенные) атомы классов окон, например: Dialog Box - `0x8002`, Button - `0xC017`, Static - 0xC019 и т.д.	От `0x0001` до `0xFFFF`	Если параметр не задан или задано значение 0, то атом класса окна не определен.
`ClassName`	Имя класса окна. Строка, не чувствительная к регистру. Приложения, регистрируя собственные классы окон, передают системе имя класса, а в ответ получают атом класса. В системе имеются стандартные классы окон.	Строка символов, произвольной длины. Примеры стандартных классов: `#32770` – Dialog Box, `Button`, `Static`, `SysListView32`	Если параметр не задан или задана пустая строка, то имя класса окна не определено.
`Text`	Текст окна. Строка. В зависимости от класса окна это может быть текст заголовка окна, текст в строке редактирования, текст в элементах управления, статических окнах и т.п.	В строке могут присутствовать символы ***** (любая последовательность символов в данной позиции) и **?** (любой одиночный символ в данной позиции). Это позволяет определять искомый текст не по точному соответствию, а по частичному, например: `О программе *`	Если параметр не задан или задана пустая строка, то текст создаваемого окна не определен.

OnCreationText	Если для окна параметром Text задан некоторый текст, то данный параметр конкретизирует, должен ли заданный текст присутствовать в окне в момент создания, или может быть занесен в окно позже.	Значения Yes, Y или 1 определяют, что текст в окне должен присутствовать в момент создания. Значения No, N или 0 допускают, что текст в окно может быть занесен позже.	Если параметр не задан, то принимается значение No. **Примечание**: Если параметр имеет значение No, и при создании окна его текст не соответствует заданному, то окно становится ***кандидатом*** на соответствие. Если далее текст в окне изменится и будет соответствовать заданному, то GUI-событие будет выполнено.
ID	Идентификатор окна (для дочерних окон). Число. Родительские окна обычно задают для создаваемых дочерних окон различные идентификаторы.	Для окон диалога, содержащих элементы управления в виде кнопок, принято (но не является обязательным) задание для кнопок стандартных значений ID. Например, кнопки: 1 – OK, 2 – Cancel, 3 – Abort и т.д.	Если параметр не задан или задано значение 0, то идентификатор создаваемого окна не определен.

Style	Это узел, содержащий стиль окна. Стиль это 32-х битное число. Каждый бит задает определенное свойство окна, которое может зависеть от класса окна. Есть 2 способа определить стиль окна. Проще всего задать его шестнадцатеричным числом. В тексте шаблона это будет выглядеть следующим образом:
	`<Style>`
	`  <Hex>80c801cc</Hex>`
	`</Style>`
	Другой способ, это указание
		Если стиль окна задается шестнадцатеричным числом, то оно может принимать значения от 00000000 до ffffffff. Возможные значения битов стиля даны в приложении к данному документу.	Если стиль окна не задан, а маска стиля задана (отлична от 0), то принимается значение стиля равное 0.
			Если маска стиля не задана, то значение стиля игнорируется, и создаваемое окно не проверяется на соответствие стилю.

	отдельных битов стиля класса:
	`<Style>`
	`  <Bit>WS_CAPTION</Bit>`
	`  <Bit>WS_POPUP</Bit>`
	`  <Bit>WS_CHILD</Bit>`
	`  <Bit>WS_VISIBLE</Bit>`
	`  ...`
	`</Style>`		

StyleMask	Это узел, содержащий маску значимых битов стиля окна. Как и стиль окна, маска стиля может быть задана шестнадцатеричным числом или набором отдельных битов. Если для окна задан стиль и маска стиля то: На стиль созданного окна накладывается маска стиля (побитная операция AND) и сравнивается с заданным стилем, на который также наложена маска стиля. Если результаты совпадают, то созданное окно удовлетворяет заданному стилю.	Если маска стиля задается шестнадцатеричным числом, то оно может принимать значения от `00000000` до `ffffffff`. Возможные значения битов стиля и маски даны в приложении к данному документу.	Если маска стиля не задана или задана значением 0, то создаваемое окно не проверяется на соответствие стилю.
ExtStyle	Это узел, содержащий расширенный стиль окна. Расширенный стиль, как и просто стиль окна является 32-х битным число. Каждый бит задает определенное свойство окна, которое определяется классом окна. Определение расширенного стиля производится точно также как и просто стиля, меняется лишь имя тэга: `<ExtStyle>` вместо `<Style>`.	Если расширенный стиль окна задается шестнадцатеричным числом, то оно может принимать значения от `00000000` до `ffffffff`. Возможные значения битов расширенного стиля даны в приложении к данному документу.	Если расширенный стиль окна не задан, а маска задана (отлична от 0), то принимается значение расширенного стиля равное 0. Если маска не задана, то значение расширенного стиля игнорируется, и создаваемое окно не проверяется на соответствие расширенному стилю.
ExtStyleMask	Узел, содержащий маску значимых битов расширенного стиля окна. Если для окна задан расширенный стиль и маска расширенного стиля то: На расширенный стиль созданного окна накладывается маска расширенного стиля и сравнивается с заданным расширенным стилем, на который также наложена маска. Если	Если маска задается шестнадцатеричным числом, то оно может принимать значения от `00000000` до `ffffffff`. Возможные значения битов расширенного	Если маска расширенного стиля не задана или задана значением 0, то создаваемое окно не проверяется на соответствие расширенному стилю.

	результаты совпадают, то созданное окно удовлетворяет заданному расширенному стилю.	стиля и маски даны в приложении к данному документу.	
NoClose	Если данный параметр задан в значение Yes, то предполагается, что созданное окно не должно закрываться (уничтожаться). Чаще всего этот параметр задается в предусловии или в первом блоке (GUI-событии) основного сценария транзакции. Если параметр задан в одном из GUI-событий предусловия и окно закрывается, то, как правило, предусловие становится невыполненным, а начатая транзакция снимается. Если параметр задан в GUI-событии основного сценария и окно закрывается до момента полного выполнения сценария цикла транзакции, то блок создания окна становится невыполненным.	Значения Yes, Y или 1 определяют, что окно не должно закрываться. Значения No, N или 0 допускают закрытие созданного окна.	Если параметр не задан, то принимается значение No.
OnClose	Параметр определяет действие, которое будет произведено, если окно с параметром NoClose установленным в Yes будет закрыто.	Может иметь значения: Nothing – не производить никаких действий, UnusualCase – увеличить на 1 счетчик нетипичного поведения в транзакции, Error – завершает транзакцию по ошибке.	Если параметр не задан, то принимается значение Nothing
ParentAlias	Задает псевдоним GUI-события создания окна, являющегося родительским окном для данного. Очевидно, что GUI-событие создания родительского окна должно быть выполнено прежде, чем будет выполнено GUI-событие создания текущего окна. Проверяется также, что создаваемое окно действительно	Строка, содержащая псевдоним GUI-события создания родительского окна (не путать с псевдонимом блока создания родительского окна).	Если параметр не задан, то при создании окна не проверяется его родитель.

	является прямым потомком окна, созданного в блоке родительского окна. Если блок GUI-события находится в предусловии, то он может ссылаться на GUI-событие создания родительского окна, размещенное только в предусловии. Блоки создания окон, находящиеся в основном сценарии могут ссылаться как на блоки основного сценария, так и на блоки предусловия.		
AncestorAlias	Задает псевдоним GUI-события создания окна, являющегося предком данного окна. В отличие от ссылки на родительское окно, ссылка на окно предка «мягче». Требуется только, чтобы блок создания окна предка был выполнен, но не производится реальная проверка – является ли создаваемое окно потомком. Правила размещения блока создания окна предка такие же, как и для блока создания окна родителя.	Строка, содержащая псевдоним GUI-события создания окна предка.	Если параметр не задан, то при создании окна не проверяется выполнение блока создания окна предка.

Таблица 3. Параметры GUI-события создания окна (WindowCreated)

Несмотря на то, что параметров GUI-события создания окна достаточно много, в реальной практике нет необходимости использовать их все. При описании в шаблоне события создания окна, всегда думайте, является ли абсолютно необходимым использование каждого из перечисленных параметров.

Вот некоторые соображения по использованию параметров при создании окон.

1. Обязательным является задание хотя бы одного из трех параметров: атом класса, имя класса или текст окна. Атом класса является числовым эквивалентом имени класса. Имя класса является строкой. Как правило, имя класса дает некоторое представление о создаваемом окне и запоминается легче. Поэтому не задавайте атом класса, если можно задать имя класса окна.

2. В некоторых случаях можно обойтись без задания текста окна. Например, если при нажатии кнопки OK в окне диалога может появиться сообщение о том что, диалог не может быть закрыт по каким-либо причинам, то создание окна такого сообщения можно описать только с использованием класса окна:

```
<BlockN type="GuiEvent">
  <GuiEvent type="WindowCreated" alias="Не все введено в диалоге">
    <ClassName>#32770</ClassName>
  </GuiEvent>
```

```
    </BlockN>
```

3. Если вы не уверены, что нужный текст окна появляется в нем сразу в момент создания окна, то не используйте параметр <OnCreationText>Yes</OnCreationText>. Если текст окна может содержать некоторые изменяемые части, то используйте символ * в нужных позициях текста. Например, текст окна сообщения об ошибке может выглядеть так:

```
    <Text>Ошибка * при подключении к базе данных. *</Text>
```

4. Параметр идентификатора ID создаваемого окна есть смысл использовать, например, в ситуации, когда создается несколько однотипных окон, имеющих один класс, но различающихся идентификаторами, и в дальнейшем вам нужно следить за событиями в каждом или в отдельном окне:

```
    <BlockN type="GuiEvent">
      <GuiEvent type="WindowCreated" alias="Создание окна X из главного окна">
        <ClassName>MYPROGRAM3245:ES14523</ClassName>
        <ID>1</ID>
      </GuiEvent>
    </BlockN>
    <BlockN+1 type="GuiEvent">
      <GuiEvent type="WindowCreated" alias="Создание окна Y из главного окна">
        <ClassName>MYPROGRAM3245:ES14523</ClassName>
        <ID>2</ID>
      </GuiEvent>
    </BlockN+1>
```

Другая ситуация. В окне диалога создаются кнопки OK и Cancel. Вам необходимо отследить момент, когда клавиша OK станет разрешенной. Для этого вам понадобится ссылка на событие создания окна кнопки.

```
    <BlockN type="GuiEvent">
      <GuiEvent type="WindowCreated" alias="Создание кнопки OK в диалоге
SomeDlg">
        <ClassName>Button</ClassName>
        <ParentAlias>SomeDlg</ParentAlias>
        <ID>1</ID>
      </GuiEvent>
    </BlockN>
```

Зная, что идентификатор кнопки OK всегда 1, имя класса кнопки Button, вы точно отследите ее создание. Заметьте, что в примере использован параметр ParentAlias с некоторым условным псевдонимом GUI-события создания окна родителя – окна диалога, в котором создается кнопка. Здесь использование этого параметра гарантирует, что кнопка создается именно в окне нужного диалога.

5. Параметры `Style`, `ExtStyleMask`, `ExtStyle` и `ExtStyleMask`. Анализ стиля или расширенного стиля создаваемого окна является «экзотикой». Можно сказать, что он вам не потребуется никогда. Можно конечно придумать ситуацию, когда логикой работы программы заложено создание окна с разным набором стилей в зависимости от ситуации, и для вас важно понимать, с каким именно стилем создано окно. Ну, в таком случае и попытаетесь использовать эти параметры. Например, описываем создание дочернего окна со стилем: `WS_CHILD` - дочернее, `WS_POPUP` – всплывающее, `WS_CAPTION` – имеет заголовок, `WS_SYSMENU` – имеет системное меню, `WS_MINIMIZEBOX` – может быть минимизировано, `WS_MAXIMIZEBOX` – может быть максимизировано, `WS_THICKFRAME` – можно менять размер окна.

```
<BlockN type="GuiEvent">
  <GuiEvent type="WindowCreated" alias="ChildWnd">
    <ParentAlias>MainWnd</ParentAlias>
    <ClassName>MYPROG356:ES1563</ClassName>
    <StyleMask>
      <Bit>WS_CHILD</Bit>
      <Bit>WS_POPUP</Bit>
      <Bit>WS_CAPTION</Bit>
      <Bit>WS_SYSMENU</Bit>
      <Bit>WS_MINIMIZEBOX</Bit>
      <Bit>WS_MAXIMIZEBOX</Bit>
      <Bit>WS_THICKFRAME</Bit>
    </StyleMask>
    <Style>
      <Bit>WS_CHILD</Bit>
      <Bit>WS_POPUP</Bit>
      <Bit>WS_CAPTION</Bit>
      <Bit>WS_SYSMENU</Bit>
      <Bit>WS_MINIMIZEBOX</Bit>
      <Bit>WS_MAXIMIZEBOX</Bit>
      <Bit>WS_THICKFRAME</Bit>
    </Style>
  </GuiEvent>
</Block0>
```

В приведённом примере параметры `StyleMask` и `Style` содержат одинаковый набор бит, т.е. равны. В этом случае, если создаваемое окно будет иметь еще какие либо дополнительные биты стиля, то их наличие никак не отразится на условии создания окна, т.к. в маске эти биты будут отсутствовать. Если вы желаете строго ограничить набор бит стиля создаваемого окна только указанными битами (наличие других битов стиля не допускается), то задавайте маску ffffffff.

```
<StyleMask>
  <Hex>ffffffff</Hex>
</StyleMask>
```

6. Используйте параметр `NoClose` только в случае, когда это действительно необходимо. В GUI-событиях создания окна расположенных в предусловии транзакции параметр `NoClose` используется для того, чтобы гарантировать, что основной сценарий будет выполняться, только

если созданное окно будет существовать. Если окно закрывается, то предусловие становится невыполненным и транзакция, если она начата, снимается. Начало новой транзакции будет возможно, только если предусловие вновь будет выполнено. В GUI-событиях создания окна в основном сценарии транзакции параметр `NoClose` часто используется для первого (начального) блока сценария. Созданное окно не должно закрываться до полного выполнения сценария транзакции. Если по каким-либо причинам это произойдет, то блок создания окна станет невыполненным и если сценарий последовательный `type="Chain"` с порядком следования `order="sequenced"`, то все боки сценария станут невыполненными, т.к. они все следуют за блоком создания окна. В этом случае, кроме задания параметра `NoClose` в блоке создания окна, вы можете дополнительно задать и параметр `OnClose`, который в зависимости от его значения может увеличить счетчик ситуаций нетипичного поведения или завершить транзакцию по ошибке.

7. Существует один нюанс использования параметра `OnCreationText` со значением `No`. В случае отсутствия данного параметра также принимается это значение. Когда создается окно, подходящее по всем параметрам, кроме текста окна, то оно становится «потенциальным кандидатом» для GUI-события создания окна. Если далее в этом окне текст будет изменен на заданный, то условие создания окна будет полностью выполнено (GUI-событие и его блок станет выполненным). Если же до момента изменения текста на заданный, будет создано другое окно, также подходящее по параметрам условию создания окна (соответствие текста не обязательно), то вновь созданное окно станет «кандидатом» взамен предыдущего.

8. Параметры GUI-событий типа WindowClosed

Тип GUI-события закрытия (уничтожения) окна имеет единственный, но обязательный параметр – ссылку на GUI-событие создания окна.

Параметр	Описание
Window	Задает псевдоним GUI-события создания окна, которое должно быть закрыто.

Таблица 4. Параметры GUI-события закрытия окна (WindowCreated)

GUI-событие создания окна, на которое ссылается параметр Window события закрытия окна, может находится как в предусловии, так и в основном сценарии транзакции, если событие закрытия окна находится в основном сценарии. Если же GUI-событие закрытия окна находится в предусловии транзакции, то оно может ссылаться только на событие также находящееся в предусловии.
Когда закрывается окно, на которое ссылается параметр Window, то GUI-событие закрытия окна становится выполненным. Здесь всё с первого взгляда просто.
На самом деле, обработка события закрытия окна в сценарии производится в два этапа:

1. В первую очередь просматриваются GUI-события type="WindowClosed", которые ждут закрытия данного окна. Каждое такое событие (его блок) становится выполненным, но только в том случае, если событие закрытия окна не нарушает разрешенную последовательность событий в сценарии. Для каждого выполненного блока GUI-события закрытия окна делается проверка – как повлияло выполнение данного блока на цепочку событий, в которую входит блок. Если в результате цепочка событий также становится выполненной, то выполненным становится и сам родительский блок. Проверка продолжается уже для родительского блока и т.д. Если в результате выполнения GUI-события закрытия окна становится выполненным корневой блок (предусловие или основной сценарии транзакции) то дальнейшая проверка (этап 2) не производится.

2. Независимо от того, существовали ли GUI-события, ожидающие закрытия данного окна, производится просмотр всех GUI-событий, которые описывали создание окна (которое сейчас закрывается). Понятно, что на данный момент такие события выполнены. Далее для каждого такого GUI-события (создания окна), просматриваются все GUI-события сценария, которые на него ссылаются. Например, имеется GUI-событие создания окна диалога, далее в одном последовательном блоке с ним (не обязательно) имеется событие нажатия кнопки окна диалога и событие изменение текста в заголовке окна диалога:

```
<BlockN type="Chain" order="sequenced">
<!-- Создания диалога -->
  <Block0 type="GuiEvent">
    <GuiEvent type="WindowCreated" alias="CreateDlg">
      <ClassName>FNWNS3115</ClassName>
      <Text>Документ * (новый)</Text>
    </GuiEvent>
  </Block0>
```

```
    <!-- Нажатие кнопки OK -->
    <Block1 type="GuiEvent">
      <GuiEvent type="Command" alias="OKButtonPressed">
        <Receiver>CreateDlg</Receiver>
        <ID>1</ID>
        <NotifyCode>BN_CLICKED</NotifyCode>
      </GuiEvent>
    </Block1>
    <!-- Изменение текста заголовка окна диалога -->
    <Block2 type="GuiEvent">
      <GuiEvent type="WindowSetText">
        <Text>Документ * (редактирование)</Text>
        <Window>CreateDlg</Window>
      </GuiEvent>
    </Block2>
  </BlockN>
```

Допустим, что окно диалога было создано - `Block0` выполнен. Была нажата кнопка OK – `Block1` выполнен. Текст окна заголовка не был изменен на ожидаемый – `Block2` остался не выполнен. Далее окно диалога закрывается. GUI-событие в котором описано создание закрываемого окна это `Block0` – он выполнен. Какие GUI-события ссылаются на `Block0`? Это GUI-события в `Block1` и `Block2`. Если бы `Block2` был ранее выполнен, то выполнилась бы и вся цепочка блоков в последовательном блоке `BlockN`. Сам `BlockN` также стал бы выполненным. В таком случае, при закрытии окна диалога, не осталось бы ни одного невыполненного GUI-события, которое на него ссылается.

Но, так как в нашем примере `Block2` не был выполнен, то при закрытии окна осталось невыполненное GUI-событие, ссылающееся на его создание. В этом случае блок создания окна `Block0` **будет переведен в состояние - не выполнен**. Невыполненными станут и все блоки GUI-событий, которые на него ссылаются – это `Block1`. На `Block2` это не влияет, т.к. он и был не выполнен.

Примечание. Когда проверяются блоки, ссылающиеся на GUI-событие создания окна при его закрытии (с целью понять остались ли невыполненные ссылки), то возможны две ситуации: ссылающийся блок выполнен (нет проблем) и блок не выполнен. В последнем случае, вроде бы, невыполненный ссылающийся блок имеется, и блок создания окна надо переводить в состояние - не выполнен. Но здесь имеется одна тонкость. Если невыполненный ссылающийся блок входит в состав некоторого параллельного блока (`type="Alternative"`), который **на данный момент выполнен**, то невыполненный ссылающийся блок не оказывает влияние на блок создания окна, и его **можно не учитывать** в списке невыполненных ссылок. Сказанное можно пояснить на простом примере. Сценарий содержим фрагмент – последовательный блок, содержащий в свою очередь несколько блоков, два первых из которых: блок создания окна диалога и параллельный блок (`type="Alternative"`).

Последний содержит два блока событий: нажатие кнопки OK и нажатие кнопки Cancel:

```
  <BlockN type="Chain" order="sequenced">
  <!-- Создания диалога -->
    <Block0 type="GuiEvent">
      <GuiEvent type="WindowCreated" alias="CreateDlg">
```

```
          <ClassName>FNWNS3115</ClassName>
          <Text>Документ * (новый)</Text>
        </GuiEvent>
      </Block0>
      <!—Параллельный блок -->
      <Block1 type="Alternative">
        <!-- Нажатие кнопки OK -->
        <Block0 type="GuiEvent">
          <GuiEvent type="Command" alias="OKPressed">
            <Receiver>CreateDlg</Receiver>
            <ID>1</ID>
            <NotifyCode>BN_CLICKED</NotifyCode>
          </GuiEvent>
        </Block0>
        <!-- Нажатие кнопки Cancel -->
        <Block1 type="GuiEvent">
          <GuiEvent type="Command" alias="CancelPressed">
            <Receiver>CreateDlg</Receiver>
            <ID>2</ID>
            <NotifyCode>BN_CLICKED</NotifyCode>
          </GuiEvent>
        </Block1>
      </Block1>
      ...
      ...
    </BlockN>
```

После создания окна диалога (`Block0` входящий в `BlockN` выполнен), пользователь нажимает либо кнопку OK (выполнен `Block0` блока `Block1`) либо кнопку Cancel (выполнен `Block1` блока `Block1`). После нажатия одной из двух кнопок окно закрывается. Проверяем выполнение блоков, ссылающихся на блок создания окна. В любом случае один из двух блоков (либо блок кнопки OK, либо блок кнопки Cancel) окажется невыполненных. Но невыполненных ссылок нет, т.к. при нажатии любой из кнопок станет выполненным `Block1` входящий в `BlockN`. Т.к. этот блок является параллельным, то находящиеся в нем невыполненные блоки, ссылающиеся на создание окна, не будут учитываться.

9. Параметры GUI-событий типа WindowSetText

GUI-события с этим типом ссылаются на GUI-события создания окна и ожидают задания или изменения текста окна на заданный. закрытия (уничтожения) окна имеет единственный, но обязательный параметр – ссылку на GUI-событие создания окна.

Параметр	Описание	Значения	По умолчанию
`Window`	Задает псевдоним GUI-события создания окна, изменение или задание текста которого ожидается.		Параметр является обязательным.
`Text`	Текст окна. Строка. В зависимости от класса окна это может быть текст заголовка окна, текст в строке редактирования, текст в элементах управления, статических окнах и т.п.	В строке могут присутствовать символы ***** (любая последовательность символов в данной позиции) и **?** (любой одиночный символ в данной позиции).	Если параметр отсутствует, то текст окна содержит пустую строку.
`KeepText`	Опциональный параметр. Если параметр задан и имеет значение `Yes`, то GUI-событие становится невыполненным, если текст окна изменяется на недопустимый после того, как событие было выполнено (в окне был задан нужный текст). Значение `No`, допускает изменение текста после того как событие было выполнено.	Значения `Yes`, `Y` или `1` запрещают изменение текста окна. Значения `No`, `N` или `0` допускают изменения текста.	Если параметр отсутствует, то принимается значение `No`.
`OnTextChanged`	Параметр используется, если параметр `KeepText` задан в значение `Yes`. Определяет действие, которое будет произведено, если текст окна изменится на недопустимый после того как событие было выполнено.	Может иметь значения: `Nothing` – не производить никаких действий, `UnusualCase` – увеличить на 1 счетчик нетипичного поведения в транзакции, `Error` – завершает транзакцию по ошибке.	Если параметр не задан, то принимается значение `Nothing`

Таблица 5. Параметры GUI-события изменения/задания теста окна (WindowSetText)

10. Параметры GUI-событий типа WindowEnabled

GUI-события с этим типом ссылаются на GUI-событие создания окна и ожидают разрешения или запрещения окна в интерфейсе программы.

Параметр	Описание	Значения	По умолчанию
`Window`	Задает псевдоним GUI-события создания окна, изменение состояния которого ожидается.		Параметр является обязательным.
`Enabled`	Задает состояние, в которое переводится окно.	Значения `Yes`, `Y` или `1` ожидают разрешения окна. Значения `No`, `N` или `0` запрещения окна в интерфейсе.	Если параметр отсутствует, то принимается значение `No`.
`KeepState`	Опциональный параметр. Если параметр задан и имеет значение `Yes`, то GUI-событие становится невыполненным, если состояние окна меняется на недопустимое после того, как событие было выполнено. Значение `No`, допускает изменение состояния после того как событие было выполнено.	Значения `Yes`, `Y` или `1` запрещают изменение состояния. Значения `No`, `N` или `0` допускают.	Если параметр отсутствует, то принимается значение `No`.
`OnStateChanged`	Параметр используется, если параметр `KeepState` задан в значение `Yes`. Определяет действие, которое будет произведено, если состояние окна изменится на недопустимое после того как событие было выполнено.	Может иметь значения: `Nothing` – не производить никаких действий, `UnusualCase` – увеличить на 1 счетчик нетипичного поведения в транзакции, `Error` – завершает транзакцию по ошибке.	Если параметр не задан, то принимается значение `Nothing`

Таблица 6. Параметры GUI-события разрешения/запрещения окна (WindowEnabled)

11. Параметры GUI-событий типа ShowWindow

GUI-события с этим типом ссылаются на GUI-событие создания окна и ожидают изменение видимости окна (показ или скрытие) в интерфейсе программы.

Параметр	Описание	Значения	По умолчанию
`Window`	Задает псевдоним GUI-события создания окна, изменение видимости которого ожидается.		Параметр является обязательным.
`Show`	Задает ожидаемое состояние видимости окна.	Значения `Yes`, `Y` или `1` ожидают появления видимого окна. Значения `No`, `N` или `0` скрытия окна.	Если параметр отсутствует, то принимается значение `No`.
`KeepState`	Опциональный параметр. Если параметр задан и имеет значение `Yes`, то GUI-событие становится невыполненным, если состояние видимости окна меняется на недопустимое после того, как событие было выполнено. Значение `No`, допускает изменение состояния видимости после того как событие было выполнено.	Значения `Yes`, `Y` или `1` запрещают изменение состояния. Значения `No`, `N` или `0` допускают.	Если параметр отсутствует, то принимается значение `No`.
`OnStateChanged`	Параметр используется, если параметр `KeepState` задан в значение `Yes`. Определяет действие, которое будет произведено, если состояние видимости окна изменится на недопустимое после того как событие было выполнено.	Может иметь значения: `Nothing` – не производить никаких действий, `UnusualCase` – увеличить на 1 счетчик нетипичного поведения в транзакции, `Error` – завершает транзакцию по ошибке.	Если параметр не задан, то принимается значение `Nothing`

Таблица 7. Параметры GUI-события показа/скрытия окна (ShowWindow)

12. Параметры GUI-событий типа InitDialog

Тип GUI-события инициализации окна диалога имеет единственный, но обязательный параметр – ссылку на GUI-событие создания окна диалога.

Параметр	Описание
`Window`	Задает псевдоним GUI-события создания окна диалога

Таблица 8. Параметры GUI-события инициализации окна диалога (InitDialog)

Основное назначение GUI-события этого типа – быть уверенным, что событие инициализации диалога имело место быть. Инициализация диалога производится после создания всех дочерних окон окна диалога. Инициализация не будет произведена, если в процессе создания дочерних окон потерпи неудачу.

13. Параметры GUI-событий типа MDIActivated

GUI-события с этим типом ссылаются на GUI-событие создания дочернего MDI окна и ожидают получения этим окном активности.

Параметр	Описание	Значения	По умолчанию
`Window`	Задает псевдоним GUI-события создания дочернего MDI окна. GUI-событие становится выполненным, как только данное окно становится активным (только одно дочернее MDI окно может быть активно в любой момент).		Параметр является обязательным.
`KeepActive`	Опциональный параметр. Если параметр задан и имеет значение `Yes`, то GUI-событие становится невыполненным, если дочернее MDI окно теряет активность после его получения. Значение `No`, допускает потерю окном активности.	Значения `Yes`, `Y` или `1` запрещают потерю активности дочерним MDI окном. Значения `No`, `N` или `0` допускают.	Если параметр отсутствует, то принимается значение `No`.
`OnActiveChanged`	Параметр используется, если параметр `KeepActive` задан в значение `Yes`. Определяет действие, которое будет произведено, если дочернее MDI окно потеряет активность после его получения.	Может иметь значения: `Nothing` – не производить никаких действий, `UnusualCase` – увеличить на 1 счетчик нетипичного поведения в транзакции, `Error` – завершает транзакцию по ошибке.	Если параметр не задан, то принимается значение `Nothing`

Таблица 9. Параметры GUI-события изменения активности дочернего MDI окна (MDIActivated)

14. Параметры GUI-событий типа WindowStyleChanged

GUI-события с этим типом ссылаются на GUI-событие создания окна и ожидают изменения стиля или расширенного стиля окна.

Параметр	Описание	Значения	По умолчанию
Window	Задает псевдоним GUI-события создания окна, изменение стиля которого ожидается.		Параметр является обязательным.
Flag	Определяет изменение стиля или расширенного стиля окна ожидается.	Значение GWL_EXSTYLE – изменяется расширенный стиль окна, GWL_STYLE – изменяется стиль окна	Если параметр отсутствует, то принимается значение GWL_STYLE.
Style	Это узел, содержащий ожидаемый стиль или расширенный стиль окна, в зависимости от значения параметра Flag. Подробности задания параметров узла смотрите в таблице параметров GUI-события создания окна.	Если стиль или расширенный стиль окна задается шестнадцатеричным числом, то оно может принимать значения от 00000000 до ffffffff. Возможные значения битов даны в приложении к данному документу.	Если стиль/расширенный стиль окна не задан, а маска задана (отлична от 0), то принимается значение стиля/расширенного стиля равное 0. Если маска не задана, то значение стиля/расширенного стиля игнорируется. Т.е. любое изменение стиля приведет к выполнению данного GUI-события.
StyleMask	Это узел, содержащий маску значимых битов стиля/расширенного стиля окна. Подробности задания параметров узла смотрите в таблице параметров GUI-события создания окна.	Если маска стиля задается шестнадцатеричным числом, то оно может принимать значения от 00000000 до ffffffff. Возможные значения битов даны в	Если маска стиля не задана или задана значением 0, то любое изменение стиля приведет к выполнению данного GUI-события.

		приложении.	
KeepStyle	Опциональный параметр. Если параметр задан и имеет значение Yes, то GUI-событие становится невыполненным, если стиль или расширенный стиль окна меняется на недопустимый, после того, как событие было выполнено. Значение No, допускает изменение стиля после того как событие было выполнено.	Значения Yes, Y или 1 запрещают изменение стиля/расширенного стиля окна. Значения No, N или 0 допускают.	Если параметр отсутствует, то принимается значение No.
OnStyleChanged	Параметр используется, если параметр KeepStyle задан в значение Yes. Определяет действие, которое будет произведено, если стиль или расширенный стиль окна изменится на недопустимый после того как событие было выполнено.	Может иметь значения: Nothing – не производить никаких действий, UnusualCase – увеличить на 1 счетчик нетипичного поведения в транзакции, Error – завершает транзакцию по ошибке.	Если параметр не задан, то принимается значение Nothing

Таблица 10. Параметры GUI-события изменения стиля окна (WindowStyleChanged)

15. Параметры GUI-событий типа Command

GUI-событие этого типа ссылается на GUI-событие создания окна (получателя и/или отправителя команды) и ожидает команды от меню, акселератора или элемента управления.

Параметр	Описание	Значения	По умолчанию
Receiver	Задает псевдоним GUI-события создания окна, получающего команду.		Параметр является обязательным, если не задан параметр ControlWindow.
ControlWindow	Задает псевдоним GUI-события создания окна элемента управления. Для команд от меню и акселератор параметр не используется.		Параметр является обязательным, если не задан параметр Receiver.
FromControl	Опциональный параметр. Уточняет, что является источником команды: меню/акселератор, элемент управления (Control) или оба.	Yes, Y или 1 - источником команды является элемент управления (Control). No, N или 0 - источником команды является меню или акселератор.	Если параметр отсутствует, или задано любое другое значение, то источником команды может что угодно.
ID	Идентификатор команды меню, акселератора или элемента управления (Control). Для меню и акселератора идентификатор команды указывать обязательно. Для элемента управления этот параметр необязателен.	Целое число, отличное от 0. Диапазон возможных значений – от 1 до 65535.	Если параметр отсутствует, то идентификатор не проверяется.
NotifyCode	Обязательный параметр, если параметр FromControl имеет значение Yes, т.е. ожидается команда посылаемая элементом управления (Control). Уточняет код нотификации.	В приложении к данному документу приводится полный список возможных значений кодов нотификации команд. Пример: BN_CLICKED – нажатие кнопки, EN_CHANGE – изменение текста в	Если параметр отсутствует, то код нотификации принимается равным 0.

		окне редактирования	
NotifyCodeHex	Является альтернативой параметру NotifyCode. Задает код нотификации в виде шестнадцатеричного числа.	Число от 0x0000 до 0xFFFF	Если параметр отсутствует, то код нотификации принимается равным 0.
ControlClassAtom	Если параметр FromControl имеет значение Yes, то задает атом класса окна элемента управления, посылающего команду. Шестнадцатеричное число.	От 0x0001 до 0xFFFF	Если параметр отсутствует, то проверка на атом класса не выполняется.
ClassName	Если параметр FromControl имеет значение Yes, то задает имя класса окна элемента управления, посылающего команду.	Строка, содержащая имя класса, например Button	Если параметр отсутствует, то проверка на имя класса не выполняется.
ControlText	Если параметр FromControl имеет значение Yes, то задает текст окна элемента управления, посылающего команду.		Если параметр отсутствует, то проверка на тест окна элемента управления не выполняется.

Таблица 11. Параметры GUI-события получения окном команды (Command)

16. Параметры GUI-событий типа Notify

GUI-событие этого типа ссылаются на GUI-событие создания окна (получающего и/или отправляющего нотификацию) и ожидает кода нотификации от элемента управления (Control).

Параметр	Описание	Значения	По умолчанию
`Receiver`	Задает псевдоним GUI-события создания окна. Данное окно получает нотификацию.		Параметр является обязательным, если не задан параметр `ControlWindow`.
`ControlWindow`	Задает псевдоним GUI-события создания окна элемента управления, отправляющего нотификацию.		Параметр является обязательным, если не задан параметр `Receiver`.
`idFrom`	Идентификатор элемента управления, посылающий нотификацию.	Целое число, отличное от 0. Диапазон возможных значений – от 1 до 65535.	Если параметр отсутствует, то проверка идентификатора элемента управления не выполняется.
`NotifyCode`	Обязательный параметр. Задает код нотификации.	В приложении к данному документу приводится полный список возможных значений кодов нотификации. Пример: `NM_DBLCLK` – двойной щелчок мышью на элементе управления, `LVN_ITEMCHANGED` – изменен выбор элементе управления типа ListView.	Параметр является обязательным и не имеет значения по умолчанию.
`NotifyCodeHex`	Является альтернативой параметру `NotifyCode`. Задает код нотификации в виде шестнадцатеричного числа.	Число от `0x00000001` до `0xFFFFFFFF`	Параметр является обязательным и не имеет значения по умолчанию.
`ControlClassAtom`	Задает атом класса окна элемента управления, посылающего нотификацию. Шестнадцатеричное число.	От `0x0001` до `0xFFFF`	Если параметр отсутствует, то проверка на атом класса не

			выполняется.
ClassName	Задает имя класса окна элемента управления, посылающего нотификацию.	Строка, содержащая имя класса окна элемента управления, например SysListView32	Если параметр отсутствует, то проверка на имя класса не выполняется.
ControlText	Если параметр FromControl имеет значение Yes, то задает текст окна элемента управления, посылающего нотификацию.		Если параметр отсутствует, то проверка на тест окна элемента управления не выполняется.
ListViewRequired Selection	Используется для класса элемента управления **SysListView32** (известен как просто ListView). Если параметр имеет значение Yes, то при поступлении ожидаемого кода нотификации, производится дополнительный анализ: сделан ли в ListView выбор элемент списка. В этом случае параметр NotifyCode должен быть задан со значением NM_DBLCLK или LVN_ITEMCHANGED, а параметр ClassName должен иметь значение SysListView32.	Yes, Y или 1 – требует выбора элемента списка.	Если параметр имеет другое значение или отсутствует, то при поступлении кода нотификации не делается проверка – выбран ли элемент списка или нет.

Таблица 12. Параметры GUI-события получения окном нотификации (Notify)

17. Атрибуты блоков

При описании различных типов блоков, был упомянут важнейший (и обязательный) атрибут `type`, который может принимать значения `GuiEvent`, `Chain` и `Alternative`. Блоки могут иметь и другие атрибуты, часть которых применяется для всех типов блоков, другие – только для конкретных типов блоков:

Имя атрибута	Тип блока	Описание	Значения атрибута
`Alias`	Для всех типов	Псевдоним блока. Не путать с псевдонимом GUI-события. Может содержать произвольный текст и является обязательным, если другие блоки ссылаются на этот через значение атрибута `ClearBlock` (описание дано ниже). Вы можете без ограничений задавать псевдонимы любому блоку в сценарии. Псевдонимы блоков отображаются в интерфейсе программы EPM-Agent.	Произвольная строка символов.
`Order`	Только для типа `Chain`	Задает последовательность выполнения внутри блока.	Значение `sequenced` требует выполнения всех блоков, входящих в данный, в строгом порядке, соответствующем номерам блоков. Значение `any` допускает выполнение блоков в любом порядке. При отсутствии данного атрибута принимается значение `sequenced`.
`ClearRoot`	Для всех типов блоков. Корневые блоки не должны иметь этого атрибута.	Если атрибут имеет значения `Yes`, и если данный блок будет выполнен, то произойдет очистка корневого блока. Если блок, с этим атрибутом находится в основной части сценария, то все блоки основного сценария станут невыполненными. Практически, это означает, что транзакция начинается сначала. Если блок с данным атрибутом находится в предусловии, то будет сброшено и предусловие и основной сценарий транзакции.	Значения `Yes`, `Y` или `1` - выполнять очистку корневого блока, при выполнении данного блока.
`ClearParent`	Для всех типов блоков. Корневые блоки не должны	Если атрибут имеет значения `Yes`, и если данный блок будет выполнен, то	Значения `Yes`, `Y` или `1` - выполнять очистку

	иметь этого атрибута.	произойдет очистка родительского блока. Если родительским блоком является корневой блок, то действие аналогично действию, описанному для параметра `ClearRoot`. Если родительский блок не является корневым, то сам родительский блок и все входящие в него блоки становятся невыполненными. Невыполненными также становятся блоки, ссылающиеся на любой блок, входящий в родительский.	родительского блока, при выполнении данного блока.
`ClearBlock`	Для всех типов блоков. Корневые блоки не должны иметь этого атрибута.	В значении данного атрибута задается псевдоним блока, который будет очищен, если данный блок будет выполнен. В остальном, производимые действия аналогичны описанным в атрибуте `ClearParent`.	
`Exception`	Для всех типов блоков. Корневые блоки не должны иметь этого атрибута.	Если атрибут имеет значения `Yes`, и если данный блок будет выполнен, то транзакция будет завершена по ошибке.	Значения `Yes`, `Y` или `1` – завершать транзакцию по ошибке.
`UnusualCase`	Для всех типов блоков. Корневые блоки не должны иметь этого атрибута.	Если атрибут имеет значения `Yes`, и если данный блок будет выполнен, то буде увеличен на 1 счетчик нетипичного поведения в транзакции.	Значения `Yes`, `Y` или `1` – увеличивать счетчик нетипичного поведения в транзакции.
`SendHelpMeMsg`	Для всех типов блоков. Корневые блоки не должны иметь этого атрибута.	Если атрибут имеет значения `Yes`, и если данный блок будет выполнен, то на ProLAN зонд будет отправлено сообщение HelpMe. Параметры отправляемого сообщения даются в узле `HelpMeMsg` внутри блока.	Значения `Yes`, `Y` или `1` – посылать сообщение HelpMe при выполнении данного блока.
`TimingStart`	Для всех типов блоков. Корневые блоки и блоки предусловия не должны иметь этого атрибута.	Если атрибут имеет значения `Yes`, и если данный блок будет выполнен, то включается таймер отсчета времени выполнения транзакции.	Значения `Yes`, `Y` или `1` – включить таймер.
`TimingStop`	Для всех типов блоков. Корневые блоки и блоки предусловия не должны иметь	Если атрибут имеет значения `Yes`, и если данный блок будет выполнен, то таймер отсчета времени выполнения транзакции выключается.	Значения `Yes`, `Y` или `1` – выключить таймер.

	этого атрибута.		
ResetTime	Для всех типов блоков. Корневые блоки и блоки предусловия не должны иметь этого атрибута.	Если атрибут имеет значения `Yes`, и если данный блок будет выполнен, то время выполнения транзакции, замеренное таймером обнуляется.	Значения `Yes`, `Y` или `1` – обнулить время выполнения транзакции.

Таблица 13. Атрибуты блоков сценария

18. Параметры отсылки сообщений HelpMe

Отсылка сообщений HelpMe – это опциональная интеграция решения Пятый Уровень с решением Красная Кнопка, которая позволяет автоматически регистрировать в базе данных снимок инцидента в момент появления инцидента. Инцидент фиксируется в момент выполнения блока сценария с атрибутом `SendHelpMeMsg="Yes"`. На компьютере пользователя должна быть установлена и настроена программа HelpMe Pro (см. руководство по развертыванию решения Красная Кнопка). Параметры отсылаемого сообщения размещаются между открывающим и закрывающим тегом блока в узле `HelpMeMsg`, например:

```
    <BlockN type="GuiEvent" SendHelpMeMsg="Yes">
      <!-- Это параметры сообщения HelpMe -->
      <HelpMeMsg>
        <WaitBefore>0</WaitBefore>
        <HelpMessage>Нажата кнопка Cancel при копировании файла или
папки</HelpMessage>
        <IncidentTypeIndex>1</IncidentTypeIndex>
      </HelpMeMsg>
      <!-- Это описание GUI-события блока BlocN -->
      <GuiEvent type="Command" alias="CancelBtnClick">
        <Receiver>MainDlg</Receiver>
        <FromControl>Yes</FromControl>
        <NotifyCode>BN_CLICKED</NotiyfyCode>
        <ID>2</ID>
      </GuiEvent>
    </BlockN>
```

В таблице приводится список параметров узла `HelpMeMsg`, описывающих отправляемое сообщение.

Параметр	Описание	Значения	По умолчанию
`WaitBefore`	Задает время (в миллисекундах) задержки от момента выполнения блока, перед отправкой сообщения. Применение данного параметра вызвано тем, что в снимок инцидента включается образ экрана компьютера пользователя (screenshot). Допустим, возникла некоторая ошибочная ситуация, и в интерфейсе программы появляется окно диалога с сообщением об ошибке. Окно содержит заголовок, по тексту которого мы понимаем, что произошла ошибка. В этот момент блок сценария становится выполненным. Если описание ошибки высвечивается в одном из дочерних окон диалога, то если мы сделаем снимок экрана сразу при выполнении блока, то текст описания ошибки может еще не появится. Если же мы сделаем небольшую паузу, скажем в 50 миллисекунд, то все содержимое диалога	От 0 до 10000.	Если параметр не задан, то принимается значение 0.

	уже будет присутствовать. В других случаях наоборот, необходимо делать снимок экрана без паузы, пока содержимое экрана не изменилось.		
`HelpMessage`	Задает текст сообщения HelpMe.	Строка с произвольным содержимым.	Значение не должно быть пустой строкой.
`IncidentTypeIndex`	Задает числовой эквивалент типа инцидента: 0 – сервис работает слишком медленно, 1 – сервис недоступен, 2 – сервис доступен, 3 – нужна помощь, 4 – спасибо за помощь.	Число от 0 до 4	Если параметр не задан, то принимается значение 0.

Таблица 14. Параметры сообщения HelpMe

19. Замер времени выполнения транзакции

В разделе 2, в таблице атрибутов тега **Transaction**, присутствует атрибут `Group`, который может принимать значения `None`, `Task`, `Task Chain` или `Process`. Значение атрибута поясняет, как в транзакции производится замер времени ее выполнения. Для значения `Process` производится замер общего времени выполнения транзакции, от момента выполнения первого блока до момента выполнения всего основного сценария транзакции. Значение `Task` говорит о том, что внутри транзакции замеряется некоторый отрезок времени, от момента включения таймера (`TimingStart="Y"`) до момента выключения таймера (`TimingStop="Y"`) или окончания транзакции. Значение `Task Chain` предполагает наличие нескольких замеряемых отрезков времени, т.е. таймер несколько раз может включаться и выключаться.

Примечание. Значение атрибута `Group` носит поясняющий характер. Вне зависимости от значения атрибута и также при его отсутствии управление временем замера транзакции осуществляется непосредственно по инструкциям сценария. Правила следующие:

- В момент окончания транзакции определяется, включен ли на данный момент таймер или нет?
- Если таймер выключен, то смотрится время, которое уже насчитал таймер. Если оно равно нулю (таймер не был включен или его замер был сброшен), то берется общее время выполнения транзакции. Если таймер насчитал некоторое время, то это и есть время выполнения транзакции.
- Если таймер в момент окончания транзакции еще включен, то он останавливается, вычисляется время, прошедшее с момента его последнего включения. К этому времени добавляется время, которое ранее насчитал таймер (т.к. возможно было несколько включений и выключений). Сумма времен будет временем выполнения транзакции.

Явно управляют таймером с помощью атрибутов блоков - `TimingStart="Y"`, `TimingStop="Y"` и `ResetTime="Y"`.
Атрибут `TimingStart="Y"` включает таймер. Момент включения таймера запоминается. Если таймер уже на этот момент уже был включен, то никаких действий не производится.
Атрибут `TimingStop="Y"` выключает таймер, если он на этот момент включен. Вычисляется время от момента последнего включения таймера и суммируется со временем, которое насчитал таймер ранее (если уже были включения/выключения).
Атрибут `ResetTime ="Y"` обнуляет время, которое уже насчитал таймер. Если таймер на этот момент включен, то он не останавливается. Для его остановки используйте атрибут `TimingStop="Y"` .

Если вы замеряете общее время выполнения транзакции, то можете забыть об атрибутах управления таймером. Если вы управляете таймером явно, то внимательно просмотрите логику сценария, т.е. все возможные варианты включения и выключения таймера, чтобы при окончании транзакции не получить неожиданное значение времени ее выполнения.

20. Управление откатами блоков

Атрибуты `ClearRoot="Y"`, `ClearParent="Y"`, `ClearBlock="ПсевдонимБлока"` и `Exception="Y"` дают возможность «откатывать» (чистить, делать невыполненными) блоки сценария.

Атрибут `ClearRoot="Y"` блока, расположенного в основном сценарии транзакции практически возвращает транзакцию к ее началу. Если бы в лабиринте комнат, показанном на рисунок 1, какая либо комната имела бы такой атрибут, то при попадании в нее (включении в ней света), человек оказался бы вновь перед входом в лабиринт (начинай сначала). При этом свет во всех комнатах лабиринта погаснет – все блоки транзакции станут невыполненными. Таймер, если он был включен, будет остановлен. Обнулено будет и время, которое уже насчитал таймер. Если блок с атрибутом `ClearRoot="Y"` принадлежит предусловию, то кроме действий описанных выше будет также выполнена очистка и корневого блока предусловия. В этом случае, для начала новой транзакции, необходимо будет, чтобы предусловие вновь стало бы выполненным.

Атрибут `ClearParent="Y"` вызывает очистка родительского блока - блока, в который непосредственно входит данный блок. Если родительским блоком является корневой блок, то действие аналогично действию параметра `ClearRoot`. Если родительский блок не является корневым, то сам родительский блок и все входящие в него блоки становятся невыполненными. Невыполненными также становятся блоки, ссылающиеся на любой блок, входящий в родительский. Если у комнаты **F**, показанной на рисунке 1, имеется атрибут `ClearParent="Y"`, то человек будет изгнан из всех комнат, входящих в Блок 6. Свет погаснет в комнатах **E** и **F**.
Если блок с атрибутом `ClearParent="Y"` находится в предусловии, и предусловие на этот момент было выполнено, то после очистки родительского блока производится анализ – не стало ли после этого предусловие невыполненным. Если стало, то транзакция снимается.

Атрибут `ClearBlock="ПсевдонимБлока"` блока вызывает очистку блока с заданным в значении псевдонимом блока. В остальном действие аналогично действию, описанному для параметра `ClearParent`. Если у комнаты **F**, показанной на рисунке 1, имеется атрибут `ClearBlock="Блок2"`, то человек окажется в комнате **A**. Свет погаснет в комнатах **B, C, D, E, F, G, H** и **I**. Свет погаснет также и в **Блоке 3**, т.к. он входит в последовательную цепь блоков лабиринта и следует за **Блоком 2**. Объяснение простое – в **Блок 3** нельзя попасть, не пройдя одну из галерей **Блока 2**. Если **Блок 2** погружается во тьму, то и следующие за ним блоки тоже. Заметим, что свет в **Блоке 4** погаснуть не может – он там не был включен. Ибо, если он там был включен, то человек должен был пройти лабиринт.

Если блок имеет атрибут `Exception="Y"`, то при выполнении этого блока транзакция будет завершена по ошибке. У комнаты **I**, показанной на рисунке 1, имеется этот атрибут. При включении света в этой комнате взрывается бомба. Отличие атрибута `Exception`, от атрибута `ClearRoot` состоит в том, что атрибут `ClearRoot` не завершает транзакцию, а только возвращает ее к началу. `Exception` используется в сценарии для обеспечения принудительного завершения, аварийного выхода из транзакции.

21. «Цепная реакция» в блоках

Кроме аналогии с человеком, проходящим лабиринт, для сценария транзакции можно привести и другие сравнения, для пояснения взаимного влиянием блоков при изменении их состояния – с выполнен в не выполнен, и наоборот. Представим, что в транзакции в данный момент часть блоков выполнена, а часть нет, т.е. транзакция начата, но не завершена. В результате некоторого события выполненный блок становится невыполненным. А теперь представим, что транзакция это множество косточек домино, и для ее выполнения необходимо поставить их на ребра. Если некоторая кость домино падает, то это может никак не повлиять на другие кости, если они стоят достаточно далеко. Если же одна или несколько костей стоит близко, то они упадут. Может начаться цепная реакция падения, которая может закончиться либо полным разрушением, либо частичным. Эта аналогия здесь приводится для понимания того, что проектируя сценарий транзакции, необходимо всегда учитывать взаимное влияние блоков и прорабатывать все варианты возможного поведения.

Факторов взаимного влияния блоков при переходе из состояния **выполнен** в состояние **не выполнен** всего три:

1. Нахождение блока, становящегося невыполненным в родительском блоке. Если родительский блок до этого момента был выполнен, то он **всегда станет невыполненным**, если родительский блок является последовательным блоком. Родительский блок параллельного типа **может стать невыполненным**, если его выполненность ранее обеспечивал блок, который становится невыполненным.
   Если невыполненным становится родительский блок, то это может повлиять на выполненность родительского, по отношению уже к нему, блока. Может пойти цепная реакция падений вверх по сценарию. Остановить падение может только параллельный блок.
   Если же ранее выполненный блок, входящий в последовательный блок, становится невыполненным, а родительский блок не был выполнен, то цепной реакции не возникнет – кости отстояли далеко.

2. Если блок, становящийся невыполненным, находится в последовательном блоке со строго последовательным порядком выполнения, входящих в него блоков (`type="Chain" order="sequenced"`), то все **блоки, следующие за блоком, становящимся невыполненным, также станут невыполненными**. Правда может оказаться, что таких блоков нет.

3. Если блок GUI-события создания окна становится невыполненным, то становятся невыполненными и все блоки GUI-событий, которые на него ссылаются. Так как такие блоки могут находиться в своих родительских блоках, выполненных на этот момент, то это тоже может привести к эффекту цепной реакции падения вверх по блокам родителей.

Отдельно рассмотрим вопрос перехода блока из состояния не выполнен в состояние выполнен. Здесь также возможна цепная реакция, но уже не падения костей, а (чудесным образом) их подъема. При этом могу подниматься соседние кости.

1. Если блок, становящийся выполненным, находится в последовательном блоке с порядком следования `order="sequenced"`, и этот блок является последним в цепочке, то выполненным становится и родительский блок.
   Если блок, становящийся выполненным, находится в последовательном блоке с порядком

следования `order="any"`, и другие блоки в цепочке уже выполнены, то родительский блок также станет выполненным.
То, что родительский блок станет выполненным, может повлиять на его родительский блок и т.д. Если в результате процесс дойдет до корневого блока, то транзакция или предусловие станут выполненными.

2. Если блок, становящийся выполненным, находится непосредственно в параллельном блоке, то параллельный (родительский) блок становится выполненным. Далее это может повлиять на родительский блок параллельного блока и т.д.

22. Как узнать детали GUI-событий в приложении?

После знакомства с правилами создания шаблонов транзакций у вас может возникнуть резонный вопрос:

- Допустим, какие действия выполняет пользователь в интерфейсе приложения в ходе транзакции, я знаю. Порядок следования событий и варианты различного поведения в транзакции мне тоже известны. Но, как мне узнать все эти нужные для написания сценария вещи: имена классов окон, тексты окон, идентификаторы, коды нотификации и прочее?

Для решения этой задачи есть 2 варианта, которые лучше использовать совместно:

1. **Вы можете использовать какую-либо программу, позволяющую получать эту информацию при работе приложения.** Например, я использую программу Spy++ от Microsoft. Она входит в состав всех редакций Visual Studio. Файл программы называется SPYXX.EXE. Программа позволяет получать статическую информацию об окнах и элементах управления (имя и атом класса, текст, идентификатор, окно родителя, иерархию дочерних окон и полную информацию о них). Для этого из программы Spy++, мышью вы указываете на нужное вам окно и наслаждаетесь результатом. Другой важный плюс – возможность следить за сообщениями, поступающими в выбранное окно. Вы настраиваете при необходимости фильтр сообщение, чтобы отсечь поток ненужных вам типов сообщений, и далее выполняя действия непосредственно в интерфейсе бизнес приложения, будете иметь в программе Spy++ стенографически полную последовательность всего, что происходило с окном. Стенограмма здесь упомянута потому, что ее результаты частенько приходится еще расшифровывать, обращаясь за дополнительной информацией в Windows Platform SDK, а иногда и h-файлы Visual Studio приходится смотреть. Но не хочется вас пугать - такое происходит редко, для очень заковыристых сценариев. Да…, для наблюдения за работой 64-х разрядных приложений необходимо поискать 64-х разрядную версию программы Spy++. У вас могут быть другие предпочтения. Если вы пользуетесь другой, удобной для вас программой, которая позволяет получать нужные сведения – замечательно.

2. **Сама программа EPM-Agent позволяет вести лог GUI-событий**. В настройках программы, на закладке «Лог программы», вы можете включить запись в лог нужных вам GUI-событий при работе нужного вам приложения (см. рисунок 2). Путь к файлу и имя лог файла отображается в поле «Размещение лог-файла». В любом текстовом редакторе вы можете смотреть содержимое этого файла. Для очистки содержимого лога просто удалите этот файл с диска - он будет создан заново.

   - Выключите флажок «Записывать события в транзакциях» (Этот режим используется при выполнении уже созданных и загруженных в программу транзакций).
   - Включите флажок «Записывать треки ловушек системы (уровень отладки)». Треки ловушек системы – это и есть GUI-события, происходящие в системе при работе приложений.
   - Ниже опции «Записывать треки ловушек системы» находится группа флажков с именами, соответствующими типам GUI-событий, информацию о которых необходимо записывать в лог:
     - **PROCESS_DETACH**. Событие этого типа не используется в сценариях транзакций. Запись этого типа в лог-файле говорит о том, что некоторый процесс (приложение) завершил свою работу.
     - **WM_CREATE**. Соответствует GUI-событиям типа `WindowCreated`. Запись в логе содержит информацию о дескрипторе окна, дескрипторе родительского окна, атоме и имени класса, тексте, идентификаторе, стиле и расширенном стиле создаваемого окна.

- **WM_DESTROY.** Соответствует GUI-событиям типа `WindowClosed`. Запись в логе содержит дескриптор закрываемого окна.
- **WM_SETTEXT.** Соответствует GUI-событиям типа `WindowSetText`. Запись в логе содержит дескриптор окна и новый текст окна.
- **WM_SHOWWINDOW.** Соответствует GUI-событиям типа `ShowWindow`. Запись в логе содержит дескриптор окна и состояние видимости окна (0 – скрыто, 1 - видимо).
- **WM_ENABLE.** Соответствует GUI-событиям типа `WindowEnabled`. Запись в логе содержит дескриптор окна и состояние в интерфейсе (0 – запрещено, 1 - разрешено).
- **WM_INITDIALOG.** Соответствует GUI-событиям типа `InitDialog`. Запись в логе содержит дескриптор окна диалога, инициализация которого выполнена.
- **WM_MDIACTIVATE.** Соответствует GUI-событиям типа `MDIActivated`. Запись в логе содержит дескриптор окна MDI клиента и два дескриптора дочерних MDI окон – теряющего активность и получающего активность.
- **WM_STYLECHANGED.** Соответствует GUI-событиям типа `WindowStyleChanged`. Запись в логе содержит дескриптор окна, флаг `GWL_EXSTYLE` или `GWL_STYLE` для изменения стиля или расширенного стиля соответственно, и два шестнадцатеричных числа – старый и новый стиль окна.
- **WM_COMAND.** Соответствует GUI-событиям типа `Command`. Запись в логе содержит дескриптор окна, идентификатор команды, кто является источником команды – меню, акселератор или control. Если источником является control, то далее содержится дескриптор, атом, имя класса, и текст его окна, и код нотификации команды.
- **WM_NOTIFY.** Соответствует GUI-событиям типа `Notify`. Запись в логе содержит дескриптор окна получателя нотификации, дескриптор окна Control, идентификатор Control, код нотификации, атом, имя класса и текст Control.

- Включите флажок «Записывать треки только заданного процесса» и введите имя файла процесса (вашего бизнес-приложения).

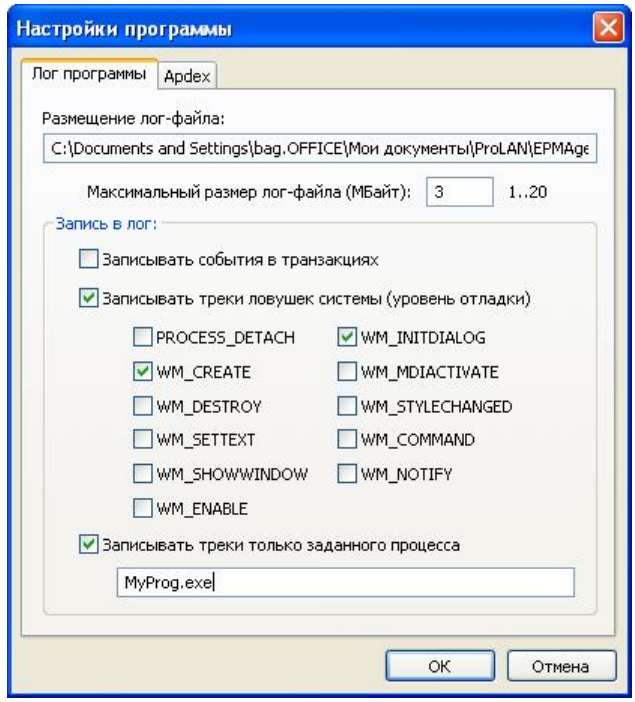


Рисунок 2. Настройка параметров лога для записи GUI-событий при работе приложения.

23. Практические упражнения

В качестве примера возьмем транзакцию некоторого гипотетического приложения «ABC Client» компании «ABC Soft». Допустим, транзакция выглядит следующим образом:

1. Пользователь в главном меню программы выбирает пункт «Новый платежный документ». В качестве альтернативы пользователь может нажать некоторую кнопку панели инструментов или нажать комбинацию клавиш акселератора. В любом случае, главное окно программы получит сообщение WM_COMMAND с идентификатором 40001.

2. Создается модальное окно диалога класса DS1405PD и текстом заголовка «Новый платежный документ». В окне диалога присутствует множество элементов управления - поля ввода, списки, закладки и т.д. В нижней части окна присутствуют три кнопки (окна класса Button) с текстами «Применить», «Закрыть», «Отмена». Допустим, идентификаторы кнопок формируются динамически – мы заранее не знаем их значения. При открытии окна диалога кнопка «Закрыть» недоступна (Disabled).

3. При нажатии кнопки «Применить», производится передача введенных данных на сервер базы данных, где специальная процедура проверяет корректность данных и если они удовлетворительны, пытается сформировать в БД новый документ.

4. Если данные некорректны или документ в БД не удается создать, то в интерфейсе программы появляется сообщение об ошибке. Пользователь может отредактировать данные и вновь нажать кнопку «Применить» для повторной передачи данных на сервер.

5. Если документ в БД создан, то в окне диалога текст меняется на «Создан документ NNNNN» и разрешается кнопка «Закрыть». При нажатии не кнопку «Закрыть» транзакция завершается.

6. Если пользователь в любой момент нажимает кнопку «Отмена», то считаем, что данная транзакция не завершена – производится возврат к началу транзакции.

Приложения

Стили окон Windows

В таблице представлены символьные и соответствующие им числовые значения битов стиля окон Windows. Символьные значения могут быть использованы в параметрах узлов тегов `Style` и `StyleMask` при описании GUI-событий создания окна (см. раздел 7) и изменения стиля окна (см. раздел 14).

Стиль	Числовое значение	Стиль	Числовое значение
WS_OVERLAPPED	0x00000000	WS_POPUP	0x80000000
WS_CHILD	0x40000000	WS_MINIMIZE	0x20000000
WS_VISIBLE	0x10000000	WS_DISABLED	0x08000000
WS_CLIPSIBLINGS	0x04000000	WS_CLIPCHILDREN	0x02000000
WS_MAXIMIZE	0x01000000	WS_CAPTION	0x00C00000
WS_BORDER	0x00800000	WS_DLGFRAME	0x00400000
WS_VSCROLL	0x00200000	WS_HSCROLL	0x00100000
WS_SYSMENU	0x00080000	WS_THICKFRAME	0x00040000
WS_GROUP	0x00020000	WS_TABSTOP	0x00010000
WS_MINIMIZEBOX	0x00020000	WS_MAXIMIZEBOX	0x00010000
WS_OVERLAPPEDWINDOW	0x00cf0000	WS_POPUPWINDOW	0x80880000
DS_ABSALIGN	0x01	DS_SYSMODAL	0x0002
DS_LOCALEDIT	0x20	DS_SETFONT	0x0040
DS_MODALFRAME	0x80	DS_NOIDLEMSG	0x100
DS_SETFOREGROUND	0x200	DS_3DLOOK	0x0004
DS_FIXEDSYS	0x0008	DS_NOFAILCREATE	0x0010
DS_CONTROL	0x0400	DS_CENTER	0x0800
DS_CENTERMOUSE	0x1000	DS_CONTEXTHELP	0x2000
DS_SHELLFONT	0x0048		
CCS_TOP	0x00000001	CCS_NOMOVEY	0x00000002
CCS_BOTTOM	0x00000003	CCS_NORESIZE	0x00000004
CCS_NOPARENTALIGN	0x00000008	CCS_ADJUSTABLE	0x00000020
CCS_NODIVIDER	0x00000040	CCS_VERT	0x00000080

CCS_LEFT	0x00000081
CCS_NOMOVEX	0x00000082
SS_LEFT	0x00000000
SS_RIGHT	0x00000002
SS_BLACKRECT	0x00000004
SS_WHITERECT	0x00000006
SS_GRAYFRAME	0x00000008
SS_USERITEM	0x0000000A
SS_LEFTNOWORDWRAP	0x0000000C
SS_BITMAP	0x0000000E
SS_ETCHEDHORZ	0x00000010
SS_ETCHEDFRAME	0x00000012
SS_REALSIZECONTROL	0x00000040
SS_NOTIFY	0x00000100
SS_RIGHTJUST	0x00000400
SS_SUNKEN	0x00001000
SS_ENDELLIPSIS	0x00004000
SS_WORDELLIPSIS	0x0000C000
ES_LEFT	0x0000
ES_RIGHT	0x0002
ES_UPPERCASE	0x0008
ES_PASSWORD	0x0020
ES_AUTOHSCROLL	0x0080
ES_OEMCONVERT	0x0400
ES_WANTRETURN	0x1000
BS_PUSHBUTTON	0x00000000
BS_CHECKBOX	0x00000002
BS_RADIOBUTTON	0x00000004
BS_AUTO3STATE	0x00000006
BS_USERBUTTON	0x00000008

CCS_RIGHT	0x00000083
SS_CENTER	0x00000001
SS_ICON	0x00000003
SS_GRAYRECT	0x00000005
SS_BLACKFRAME	0x00000007
SS_WHITEFRAME	0x00000009
SS_SIMPLE	0x0000000B
SS_OWNERDRAW	0x0000000D
SS_ENHMETAFILE	0x0000000F
SS_ETCHEDVERT	0x00000011
SS_TYPEMASK	0x0000001F
SS_NOPREFIX	0x00000080
SS_CENTERIMAGE	0x00000200
SS_REALSIZEIMAGE	0x00000800
SS_EDITCONTROL	0x00002000
SS_PATHELLIPSIS	0x00008000
SS_ELLIPSISMASK	0x0000C000
ES_CENTER	0x0001
ES_MULTILINE	0x0004
ES_LOWERCASE	0x0010
ES_AUTOVSCROLL	0x0040
ES_NOHIDESEL	0x0100
ES_READONLY	0x0800
ES_NUMBER	0x2000
BS_DEFPUSHBUTTON	0x00000001
BS_AUTOCHECKBOX	0x00000003
BS_3STATE	0x00000005
BS_GROUPBOX	0x00000007
BS_AUTORADIOBUTTON	0x00000009

BS_OWNERDRAW	0x0000000B
BS_LEFTTEXT	0x00000020
BS_ICON	0x00000040
BS_LEFT	0x00000100
BS_CENTER	0x00000300
BS_BOTTOM	0x00000800
BS_PUSHLIKE	0x00001000
BS_NOTIFY	0x00004000
LBS_NOTIFY	0x0001
LBS_NOREDRAW	0x0004
LBS_OWNERDRAWFIXED	0x0010
LBS_HASSTRINGS	0x0040
LBS_NOINTEGRALHEIGHT	0x0100
LBS_WANTKEYBOARDINPUT	0x0400
LBS_DISABLENOSCROLL	0x1000
LBS_NOSEL	0x4000
LBS_STANDARD	0x00A00003
CBS_SIMPLE	0x0001
CBS_DROPDOWNLIST	0x0003
CBS_OWNERDRAWVARIABLE	0x0020
CBS_OEMCONVERT	0x0080
CBS_HASSTRINGS	0x0200
CBS_DISABLENOSCROLL	0x0800
CBS_LOWERCASE	0x4000
SBS_VERT	0x0001
SBS_LEFTALIGN	0x0002
SBS_RIGHTALIGN	0x0004
SBS_SIZEBOXBOTTOMRIGHTALIGN	0x0004
SBS_SIZEGRIP	0x0010
HDS_BUTTONS	0x0002

BS_TYPEMASK	0x0000000F
BS_TEXT	0x00000000
BS_BITMAP	0x00000080
BS_RIGHT	0x00000200
BS_TOP	0x00000400
BS_VCENTER	0x00000C00
BS_MULTILINE	0x00002000
BS_FLAT	0x00008000
LBS_SORT	0x0002
LBS_MULTIPLESEL	0x0008
LBS_OWNERDRAWVARIABLE	0x0020
LBS_USETABSTOPS	0x0080
LBS_MULTICOLUMN	0x0200
LBS_EXTENDEDSEL	0x0800
LBS_NODATA	0x2000
LBS_COMBOBOX	0x8000
CBS_DROPDOWN	0x0002
CBS_OWNERDRAWFIXED	0x0010
CBS_AUTOHSCROLL	0x0040
CBS_SORT	0x0100
CBS_NOINTEGRALHEIGHT	0x0400
CBS_UPPERCASE	0x2000
SBS_HORZ	0x0000
SBS_TOPALIGN	0x0002
SBS_BOTTOMALIGN	0x0004
SBS_SIZEBOXTOPLEFTALIGN	0x0002
SBS_SIZEBOX	0x0008
HDS_HORZ	0x0000
HDS_HOTTRACK	0x0004

HDS_HIDDEN	0x0008
HDS_FULLDRAG	0x0080
HDS_FLAT	0x0200
TBSTYLE_SEP	0x0001
TBSTYLE_GROUP	0x0004
TBSTYLE_AUTOSIZE	0x0010
BTNS_SHOWTEXT	0x0040
TBSTYLE_TOOLTIPS	0x0100
TBSTYLE_ALTDRAG	0x0400
TBSTYLE_LIST	0x1000
TBSTYLE_REGISTERDROP	0x4000
RBS_TOOLTIPS	0x0100
RBS_BANDBORDERS	0x0400
RBS_REGISTERDROP	0x0800
RBS_VERTICALGRIPPER	0x4000
TTS_ALWAYSTIP	0x01
TTS_NOANIMATE	0x10
TTS_BALLOON	0x40
SBARS_SIZEGRIP	0x0100
TBS_AUTOTICKS	0x0001
TBS_HORZ	0x0000
TBS_BOTTOM	0x0000
TBS_RIGHT	0x0000
TBS_NOTICKS	0x0010
TBS_FIXEDLENGTH	0x0040
TBS_TOOLTIPS	0x0100
TBS_DOWNISLEFT	0x0400
UDS_WRAP	0x0001
UDS_ALIGNRIGHT	0x0004
UDS_AUTOBUDDY	0x0010

HDS_DRAGDROP	0x0040
HDS_FILTERBAR	0x0100
TBSTYLE_BUTTON	0x0000
TBSTYLE_CHECK	0x0002
TBSTYLE_DROPDOWN	0x0008
TBSTYLE_NOPREFIX	0x0020
BTNS_WHOLEDROPDOWN	0x0080
TBSTYLE_WRAPABLE	0x0200
TBSTYLE_FLAT	0x0800
TBSTYLE_CUSTOMERASE	0x2000
TBSTYLE_TRANSPARENT	0x8000
RBS_VARHEIGHT	0x0200
RBS_FIXEDORDER	0x0800
RBS_AUTOSIZE	0x2000
RBS_DBLCLKTOGGLE	0x8000
TTS_NOPREFIX	0x02
TTS_NOFADE	0x20
TTS_CLOSE	0x80
SBARS_TOOLTIPS	0x0800
TBS_VERT	0x0002
TBS_TOP	0x0004
TBS_LEFT	0x0004
TBS_BOTH	0x0008
TBS_ENABLESELRANGE	0x0020
TBS_NOTHUMB	0x0080
TBS_REVERSED	0x0200
UDS_SETBUDDYINT	0x0002
UDS_ALIGNLEFT	0x0008
UDS_ARROWKEYS	0x0020

UDS_HORZ	0x0040
UDS_HOTTRACK	0x0100
PBS_SMOOTH	0x01
PBS_MARQUEE	0x08
LVS_ICON	0x0000
LVS_SMALLICON	0x0002
LVS_TYPEMASK	0x0003
LVS_SHOWSELALWAYS	0x0008
LVS_SORTDESCENDING	0x0020
LVS_NOLABELWRAP	0x0080
LVS_EDITLABELS	0x0200
LVS_NOSCROLL	0x2000
LVS_ALIGNTOP	0x0000
LVS_ALIGNMASK	0x0c00
LVS_NOCOLUMNHEADER	0x4000
TVS_HASBUTTONS	0x0001
TVS_LINESATROOT	0x0004
TVS_DISABLEDRAGDROP	0x0010
TVS_RTLREADING	0x0040
TVS_CHECKBOXES	0x0100
TVS_SINGLEEXPAND	0x0400
TVS_FULLROWSELECT	0x1000
TVS_NONEVENHEIGHT	0x4000
TCS_SCROLLOPPOSITE	0x0001
TCS_RIGHT	0x0002
TCS_FLATBUTTONS	0x0008
TCS_FORCELABELLEFT	0x0020
TCS_VERTICAL	0x0080
TCS_BUTTONS	0x0100
TCS_MULTILINE	0x0200

UDS_NOTHOUSANDS	0x0080
PBS_VERTICAL	0x04
LVS_REPORT	0x0001
LVS_LIST	0x0003
LVS_SINGLESEL	0x0004
LVS_SORTASCENDING	0x0010
LVS_SHAREIMAGELISTS	0x0040
LVS_AUTOARRANGE	0x0100
LVS_OWNERDATA	0x1000
LVS_TYPESTYLEMASK	0xfc00
LVS_ALIGNLEFT	0x0800
LVS_OWNERDRAWFIXED	0x0400
LVS_NOSORTHEADER	0x8000
TVS_HASLINES	0x0002
TVS_EDITLABELS	0x0008
TVS_SHOWSELALWAYS	0x0020
TVS_NOTOOLTIPS	0x0080
TVS_TRACKSELECT	0x0200
TVS_INFOTIP	0x0800
TVS_NOSCROLL	0x2000
TVS_NOHSCROLL	0x8000
TCS_BOTTOM	0x0002
TCS_MULTISELECT	0x0004
TCS_FORCEICONLEFT	0x0010
TCS_HOTTRACK	0x0040
TCS_TABS	0x0000
TCS_SINGLELINE	0x0000
TCS_RIGHTJUSTIFY	0x0000

TCS_FIXEDWIDTH	0x0400	TCS_RAGGEDRIGHT	0x0800
TCS_FOCUSONBUTTONDOWN	0x1000	TCS_OWNERDRAWFIXED	0x2000
TCS_TOOLTIPS	0x4000	TCS_FOCUSNEVER	0x8000
MCS_DAYSTATE	0x0001	MCS_MULTISELECT	0x0002
MCS_WEEKNUMBERS	0x0004	MCS_NOTODAYCIRCLE	0x0008
MCS_NOTODAY	0x0010	DTS_UPDOWN	0x0001
DTS_SHOWNONE	0x0002	DTS_SHORTDATEFORMAT	0x0000
DTS_LONGDATEFORMAT	0x0004	DTS_SHORTDATECENTURYFORMAT	0x000C
DTS_TIMEFORMAT	0x0009	DTS_APPCANPARSE	0x0010
DTS_RIGHTALIGN	0x0020		
PGS_VERT	0x00000000	PGS_HORZ	0x00000001
PGS_AUTOSCROLL	0x00000002	PGS_DRAGNDROP	0x00000002

Таблица 15. Биты стиля окон Windows

Расширенные стили окон Windows

В таблице представлены символьные и соответствующие им числовые значения битов расширенного стиля окон Windows.

Символьные значения могут быть использованы в параметрах узлов тегов `ExtStyle` и `ExtStyleMask` при описании GUI-событий создания окна (см. раздел 7) и тегов `Style` и `StyleMask` при изменении стиля окна (см. раздел 14).

Стиль	Числовое значение
WS_EX_DLGMODALFRAME	0x00000001
WS_EX_TOPMOST	0x00000008
WS_EX_TRANSPARENT	0x00000020
WS_EX_TOOLWINDOW	0x00000080
WS_EX_CLIENTEDGE	0x00000200
WS_EX_RIGHT	0x00001000
WS_EX_LEFTSCROLLBAR	0x00004000
WS_EX_CONTROLPARENT	0x00010000
WS_EX_APPWINDOW	0x00040000
WS_EX_PALETTEWINDOW	0x00000188
WS_EX_NOINHERITLAYOUT	0x00100000
WS_EX_COMPOSITED	0x02000000
TBSTYLE_EX_DRAWDDARROWS	0x00000001
TBSTYLE_EX_HIDECLIPPEDBUTTONS	0x00000010
LVS_EX_GRIDLINES	0x00000001
LVS_EX_CHECKBOXES	0x00000004
LVS_EX_HEADERDRAGDROP	0x00000010
LVS_EX_ONECLICKACTIVATE	0x00000040
LVS_EX_FLATSB	0x00000100
LVS_EX_INFOTIP	0x00000400
LVS_EX_UNDERLINECOLD	0x00001000
LVS_EX_LABELTIP	0x00004000

Стиль	Числовое значение
WS_EX_NOPARENTNOTIFY	0x00000004
WS_EX_ACCEPTFILES	0x00000010
WS_EX_MDICHILD	0x00000040
WS_EX_WINDOWEDGE	0x00000100
WS_EX_CONTEXTHELP	0x00000400
WS_EX_RTLREADING	0x00002000
WS_EX_RIGHTSCROLLBAR	0x00000000
WS_EX_STATICEDGE	0x00020000
WS_EX_OVERLAPPEDWINDOW	0x00000300
WS_EX_LAYERED	0x00080000
WS_EX_LAYOUTRTL	0x00400000
WS_EX_NOACTIVATE	0x08000000
TBSTYLE_EX_MIXEDBUTTONS	0x00000008
TBSTYLE_EX_DOUBLEBUFFER	0x00000080
LVS_EX_SUBITEMIMAGES	0x00000002
LVS_EX_TRACKSELECT	0x00000008
LVS_EX_FULLROWSELECT	0x00000020
LVS_EX_TWOCLICKACTIVATE	0x00000080
LVS_EX_REGIONAL	0x00000200
LVS_EX_UNDERLINEHOT	0x00000800
LVS_EX_MULTIWORKAREAS	0x00002000
LVS_EX_BORDERSELECT	0x00008000

LVS_EX_DOUBLEBUFFER	0x00010000	LVS_EX_HIDELABELS	0x00020000
LVS_EX_SINGLEROW	0x00040000	LVS_EX_SNAPTOGRID	0x00080000
LVS_EX_SIMPLESELECT	0x00100000		
CBES_EX_NOEDITIMAGE	0x00000001	CBES_EX_NOEDITIMAGEINDENT	0x00000002
CBES_EX_PATHWORDBREAKPROC	0x00000004	CBES_EX_NOSIZELIMIT	0x00000008
CBES_EX_CASESENSITIVE	0x00000010		
TCS_EX_FLATSEPARATORS	0x00000001	TCS_EX_REGISTERDROP	0x00000002

Таблица 16. Биты расширенного стиля окон Windows

Коды нотификации GUI-событий типа Command

В таблице представлены символьные и соответствующие им числовые значения кодов нотификации, поступающих в родительское окно элемента управления по сообщению WM_COMMAND. Символьные значения могут быть использованы в параметре `NotifyCode` при описании GUI-событий посылки команды окну (см. раздел 15). В таблице приводятся только наиболее популярные (и полезные) значения кодов нотификации для стандартных элементов управления Windows. Тем не менее, при описании GUI-события, **вы можете задавать любое другое значение кода нотификации**, в шестнадцатеричном виде, используя параметр `NotifyCodeHex`.

Код нотификации	Числовое значение	Код нотификации	Числовое значение
ACN_START	0x01	ACN_STOP	0x02
BN_CLICKED	0x00	BN_SETFOCUS	0x06
BN_KILLFOCUS	0x07	CBN_DBLCLK	0x02
CBN_EDITCHANGE	0x05	CBN_KILLFOCUS	0x04
CBN_SELCHANGE	0x01	CBN_SETFOCUS	0x03
EN_CHANGE	0x0300	EN_KILLFOCUS	0x0200
EN_SETFOCUS	0x0100	STN_CLICKED	0x00
STN_DBLCLK	0x01	LBN_DBLCLK	0x02
LBN_KILLFOCUS	0x05	LBN_SELCHANGE	0x01
LBN_SETFOCUS	0x04		

Таблица 17. Коды нотификации GUI-события типа Command

Коды нотификации GUI-событий типа Notify

В таблице представлены символьные и соответствующие им числовые значения кодов нотификации, поступающих в родительское окно элемента управления (или отправляемые элементом управления окна) по сообщению WM_NOTIFY.
Символьные значения могут быть использованы в параметрах узлов тегов `NotifyCode` при описании GUI-событий посылки окну нотификации (см. раздел 16). В таблице приводятся только наиболее популярные (и полезные) значения кодов нотификации для стандартных элементов управления Windows. При описании GUI-события, **вы не можете задавать другие значение кода нотификации**, в шестнадцатеричном виде, используя параметр `NotifyCodeHex`.

Код нотификации	Числовое значение	Код нотификации	Числовое значение
NM_DBLCLK	0xfffffffd	NM_CLICK	0xfffffffe
NM_RETURN	0xfffffffc	LVN_ITEMCHANGED	0xffffff9b
TVN_SELCHANGED	0xfffffe6e	MCN_SELECT	0xfffffd16
DTN_DATETIMECHANGE	0xfffffd09	IPN_FIELDCHANGED	0xfffffca4
PSN_APPLY	0xffffff36	PSN_RESET	0xffffff35
PSN_WIZFINISH	0xffffff30		

Таблица 18. Коды нотификации GUI-события типа Notify